Encoding Polyphony from Medieval Manuscripts Notated in Mensural Notation

Karen Desmond
Brandeis University, USA
kdesmond@brandeis.edu

Laurent Pugin
RISM Digital Center, Switzerland
laurent.pugin@rism-ch.org

Juliette Regimbal
McGill University, Canada
juliette.regimbal@mail.mcgill.ca

David Rizo
Universidad de Alicante, ISEA.CV, Spain
drizo@dlsi.ua.es

Craig Sapp
Stanford University, USA
craigsapp@gmail.com

Martha E. Thomae
McGill University, Canada
martha.thomaeelias@mail.mcgill.ca

Abstract

This panel submission for the 2021 Music Encoding Conference brings together five short papers that focus on the making of computer-readable encodings of polyphony in the notational style – mensural notation – in which it was originally copied. Mensural notation was used in the medieval West to encode polyphony from the late thirteenth to sixteenth centuries. The Measuring Polyphony (MP) Online Editor, funded by an NEH Digital Humanities Advancement Grant, is a software that enables non-technical users to make Humdrum and MEI encodings of mensural notation, and links these encodings to digital images of the manuscripts in which these compositions were first notated. Topics explored by the authors include: the processes of, and the goals informing, the linking of manuscript images to music encodings; choices and compromises made in the development process of the MP Editor in order to facilitate its rapid deployment; and the implications of capturing dual encodings – a parts-based encoding that reflects the layout of the original source, and a score-based encoding. Having two encodings of the music data is useful for a variety of activities, including performance and analysis, but also within the editorial process, and for sharing data with other applications. The authors present two case studies that document the possibilities and potential in the interchange of music data between the MP Editor and other applications, specifically, MuRET, an optical music recognition (OMR) tool, and Humdrum analysis tools.

Introduction

The availability of high-quality digital images of music manuscripts has fundamentally changed the way we encounter the music of the western Middle Ages. Instead of consulting modern printed editions, those interested can go directly to images of the original manuscript sources. Nonetheless, for both human and computer readers (such as OMR applications), issues with deciphering and understanding the manuscript sources persist– legibility, scribal copying quirks, and outright mistakes. While the nature of the interaction has changed due to the access to these images, the production of human- and computer-readable versions of what medieval scribes encoded in these manuscripts is still useful and often requires (human) editorial interventions.

What maybe is no longer needed, however, is the translation of medieval notations into modern notation, which does a poor job of communicating the nuance and simplicity of the notation in which the music was originally conceived. This panel presentation brings together five short papers that focus on the making of computer-readable encodings of polyphony in a notation used in the medieval West from the late thirteenth

to sixteenth centuries called mensural notation.[1] The Measuring Polyphony (MP) Online Editor,[2] funded by an NEH Digital Humanities Advancement Grant (Principal Investigator: Karen Desmond),[3] is a software that enables non-technical users to make Humdrum and MEI encodings of mensural notation [3]. Capitalizing on the availability of high-quality images documented by the International Image Interoperability Framework (IIIF) standards,[4] the encodings are linked directly to 'zones' within IIIF-documented manuscripts, allowing the simultaneous viewing and study of both the manuscript image and its encoding, but with the images stored and served by the manuscript's host institution. The link between image and encoding also facilitates the editing process: In §1, Juliette Regimbal introduces the data model design for the MP Editor, while in §2, Karen Desmond examines some of the early choices made in the development process, focusing on the intersection between the project's requirements and choices as to what elements get encoded.

One key feature of these encodings is that they encode both the content of the voice parts as they were copied in the original manuscripts, and encode the voice parts realized into score format. Laurent Pugin (in §3) focuses on one ubiquitous feature of mensural notation – the ligature, a glyph that ligates (that is, ties together) several note shapes – and how these ligatures are flexibly handled in the Verovio project (see §3), whether represented in parts-based or score-based encodings. In §4, David Rizo, Martha E. Thomae, and Regimbal examine the possibilities of interchange between computer applications – between MuRET, an optical music recognition (OMR) tool, and the MP Editor – so that the MP Editor's score-editing features can be used to refine MuRET's OMR transcriptions and produce digital editions. Finally, in the last section (§5), Craig Sapp, along with Regimbal and Thomae, continues this theme of interchange with a presentation of conversion tools between MEI-mensural and Humdrum **mens digital formats, which means Humdrum analysis tools can be used on MEI-mensural encodings.

1 Introducing the Data Model Design for the Measuring Polyphony Editor (Regimbal)

1.1 Overview of the Data Model and Its Requirements

All software is designed with decisions made about how its data are structured and transmitted across different components and, in many cases, outside of the software. The MP Editor is not an exception and primarily stores musical content, the locations of this content, and metadata. It has a modular design composed of three stages – the Input Editor, the Score Editor, and the Editorial Corrections mode – that structure data differently to better fit the tasks performed in each stage. Many specifics of these tasks only became clear later on, and the data model constructed for the MP Editor could not take them into account before development. Instead, a flexible, 'prototype-friendly' model was designed using MEI as a common interchange format to support the development team in implementing the different features of the editor. This model also was required to input and output MEI encodings to provide for interoperability with other tools and projects. An overview of this structure can be seen in Figure 1.1 below.

Each of the three stages uses a different format for storing and manipulating data in response to the scale of its primary editing task. The Input Editor stage either starts with no preexisting data or loads data from a parts-based mensural MEI file (on the structure of parts-based MEI see section 4.2.3 below). Data is formatted in an internal data format that stores data within a system (a single stave) in Humdrum. This permits Humdrum to be used as an intermediary to convert this internal data format, when needed, into a parts-based MEI file for the Score Editor stage. This parts file can be edited, exported, or 'scored up' to produce a score-based MEI file. The score-based MEI file is used to store data for the third and final stage of editorial corrections. Even here, the MEI encoding may pass through Humdrum again for additional processing before being exported.

1 For good introductions to mensural notation see [5, pp. 85–134] and [4].

2 https://editor.measuringpolyphony.org; open-source code available at https://measuringpolyphony.github.io/mp_editor/ (both accessed January 12, 2022).

3 https://securegrants.neh.gov/publicquery/main.aspx?f=1&gn=HAA-263800-19 (accessed January 12, 2022).

4 https://iiif.io/ (accessed January 12, 2022).

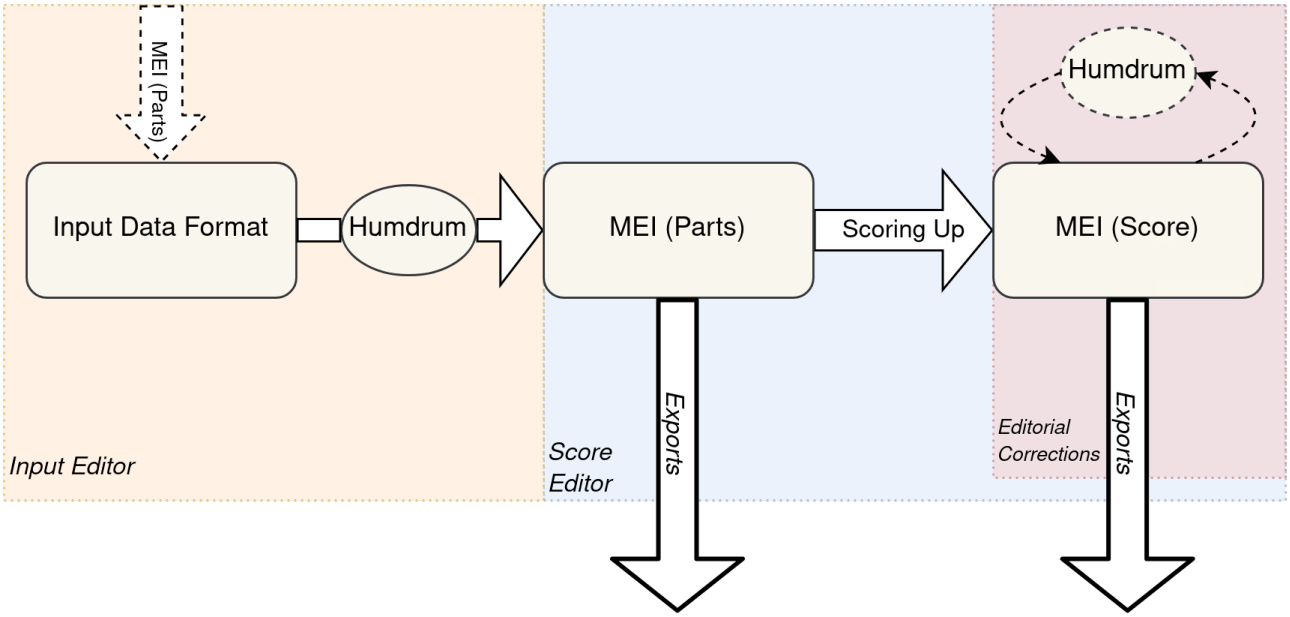


Figure 1.1: A top-level diagram of the data model. Long-lasting data formats are rounded gray rectangles while temporary data formats are gray ovals. Arrows show the flow of data across the model. Each stage of the editor is indicated by a different color rectangle in the background.

1.2 Stages of the MP Editor

The Input Editor stage uses its own internal format based on Humdrum. Each musical element that can be entered in the MP Editor is stored in a manner directly based on Humdrum's **mens format [9]. This internal input data format distinguishes itself in how it handles systems, the base unit of the Input Editor. A system is the first element that must be entered, preceding both the assignment of parts and the entry of notes, and defines the structure of the music across different pages. This structure does not necessarily emerge in order. A user can enter systems in any order they find most convenient and set the association of the systems to parts after all other content is entered. This means that the actual structure as it would be expressed as a single Humdrum or MEI document is only fixed once the user finishes this Input Editor pass.

To match the user's possible actions without risking the data being transformed into an invalid state, links between the system, its part, and its location within the manuscript and within a page are kept dynamic in the input data format. When the user signals the end of the Input Editor stage, the systems are then sorted with the links to other data resolved so they can be encoded in a parts-based MEI file as shown in Figure 1.2.

This parts-based MEI file is then transferred to the Score Editor stage. The user is presented with a view as a score, but data is edited at this stage in the parts format and converted to a score on each change. The Score Editor interface presents a pared down level of control: Changes can be made within systems and parts but the higher level actions that impact the structure of the document are not available. This ensures that maintaining valid data, in this case a valid MEI document, is straightforward. As such, changes are made directly to the parts-based MEI file using standard browser functions to manipulate the Document Object Model (DOM) on the web and XML standards like XPath queries [2]. This minimizes the overhead involved in data conversion and still allows for easily reusable outputs in the form of parts- and score-based MEI documents.

The Editorial Corrections stage keeps the same interface as the Score Editor, however the underlying task changes from fixing problems with entry in the Input Editor stage to fixing problems with how the piece was written on the original page. While this change is important to the user, the scale of the edits remains at the level of elements within systems. The same DOM and XML tools are used to enter edits directly into the score-based MEI document with the original and corrected data wrapped in `<choice>`, `<sic>`, and `<corr>` tags.

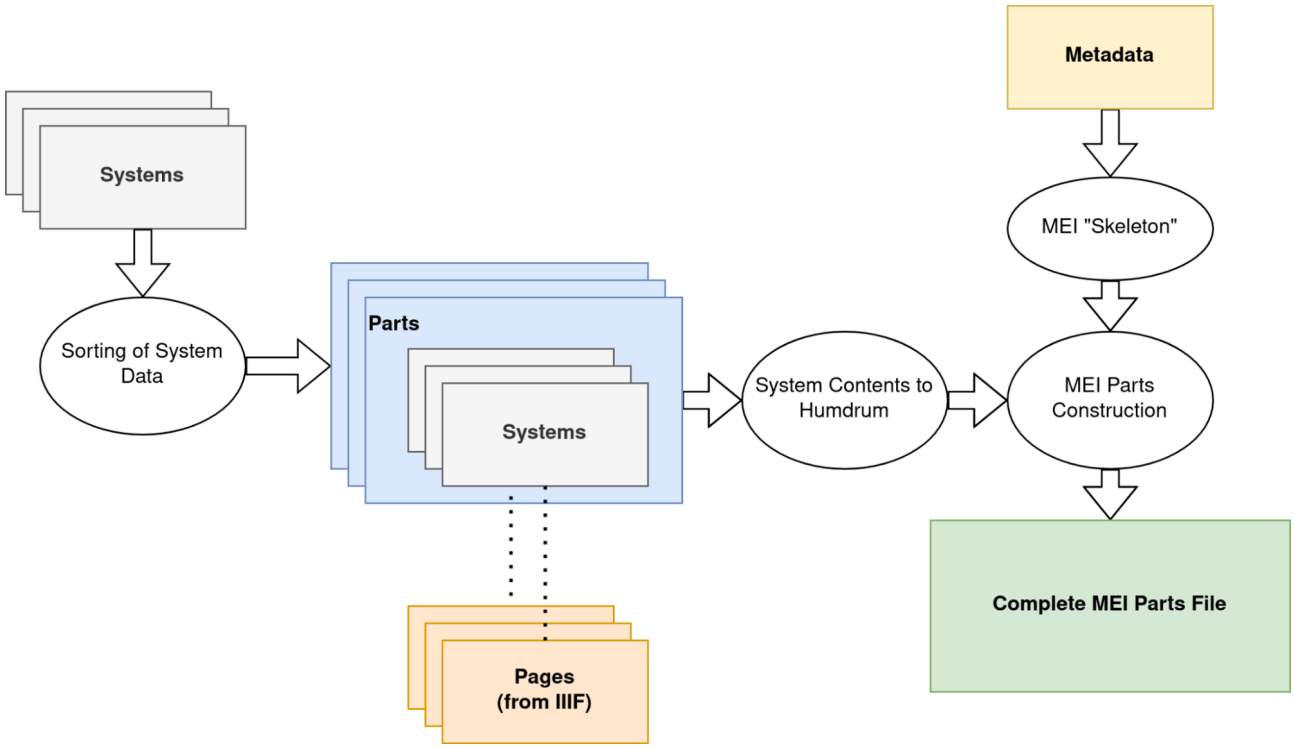


Figure 1.2: A diagram showing how a parts-based MEI file is created from the data collected in the Input Editor.

1.3 Impact of the Data Model

The data model designed for the MP Editor focuses on matching the scale of user edits through the web interface. While converting between formats adds some overhead and room for error, this choice allows for the rapid implementation of features. Now, this flexibility in the data model makes the decision easy to add other features that require conversion to a different format, rather than recreating the process ourselves. An example of this is discussed in more detail in section 5.

At this time, it appears that this approach to the data model worked well for a project like the MP Editor. Rather than spending our limited time focusing on a workable monolithic data structure, we were able to move quickly and reach more serious design challenges, like those discussed in the next section. Keeping the inputs and outputs of the editor as standard MEI also permitted its use in workflows involving other projects, like MuRET (see section 4).

2 What Gets Encoded: Early Choices, Decisions, Compromises (Desmond)

2.1 Fundamental Requirements of the Measuring Polyphony Editor

One of the fundamental requirements of the MP Editor is that it be simple: an easy-to-use tool for someone with no little to no experience in computer coding that allows them to rapidly enter pitches and rhythms through ASCII keystrokes. The user sees, in real-time, a visual representation of how the note shapes generated by their keystroke entry match the shapes in the medieval manuscript. The end goals are twofold: (1) the creation of encodings of medieval music repertoires that can be read by computers, but also (2) the generation of transcriptions and/or editions in score format that can be read by twenty-first-century musicians and scholars. A crucial by-product of the encoding process is that users of the MP Editor become intimately familiar with the palaeography and layout of medieval manuscripts: with how medieval notations work, and with how medieval scribes efficiently and elegantly encoded their music repertoires.

This requirement of simplicity influenced decisions regarding which aspects of the music must be encoded versus those whose encoding might add unnecessary complexity to the interface. One of our first decisions, which was intended to simplify and ensure the speed of the user input, but which had a significant impact on

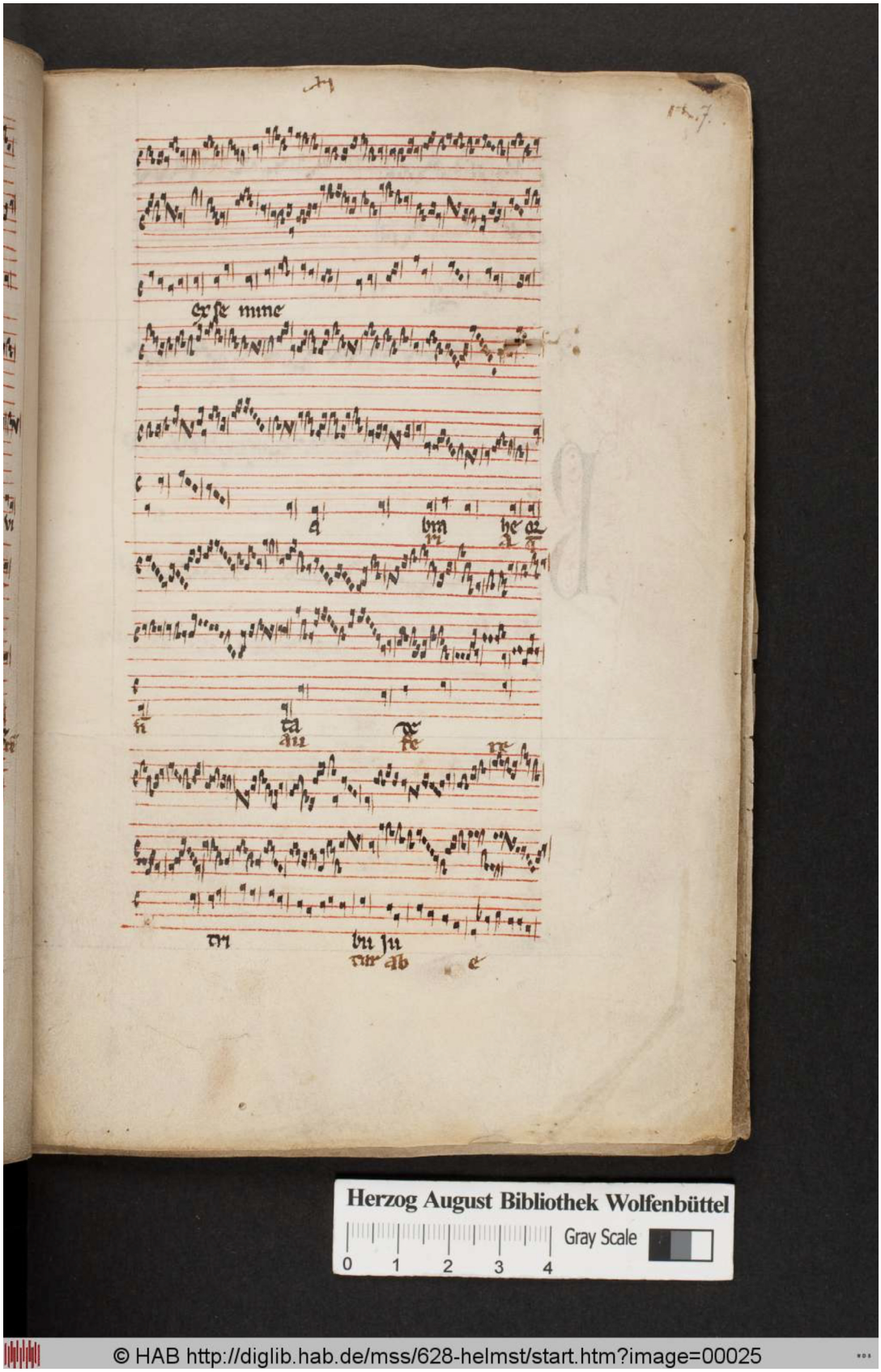


Figure 2.1: Excerpt from the organum *Alleluia. Nativitas gloriose* (excerpt), copied in score. Herzog August Bibliothek Wolfenbüttel, Cod. Guelf. 628 Helmst., fol. 7(11)r (detail). © Herzog August Bibliothek Wolfenbüttel, http://diglib.hab.de/mss/628-helmst/start.htm?image=00025 (CC BY-SA 3.0).

both the design of the interface and the encoding process, was that we chose the staff as the fundamental unit for the input interface. A compromise of this decision is that the resulting encodings do not have a one-to-one correspondence between the notational glyph and the manuscript 'zone' (as you might have in encodings generated through OMR), but rather the correspondence is between the individual staff (and its contents) and the manuscript 'zone'.

Another fundamental requirement that had a significant impact on the data model design was that we wanted to create two encodings: one that encodes the musical content as notated in parts in the original manuscript source, and a second encoding that scores up the voice parts. As part of the encoding process, the MP Score Editor allows the transcriber to quickly catch transcription mistakes because they can toggle views to see the alignment of the parts in the midst of transcribing. As described in section 1, the MP Editor was designed to accommodate these iterative processes of input, proofreading, and editing. In the following, I briefly outline some consequences that these early decisions – the staff-based encoding and the encoding of two versions of each composition (a parts-based version and a score-based version) – had on how certain structural aspects of late medieval repertoires are encoded with the MP Editor. Specifically, I outline how medieval encodings of formal structure intersect with manuscript page layout (the *mise-en-page*), and how a modern score-based layout has to realize musical content based on instructions often only implicitly conveyed in the parts-based layout.[5]

2.2 Tenor Repetitions in Motets

In the later medieval period –- that is, from about the third quarter of thirteenth century – most polyphonic repertoires were copied in a parts-based layout. This was a change from earlier polyphony: The Aquitanian and Notre Dame organum repertoires, for example, were encoded in score. The fundamental difference between organa and medieval motets is that for the duration of an entire motet, each voice part usually moves at a different rhythmic rate, and often have different texts. The top of the page shown in Figure 2.1, for example, shows a discant section from the organum *Alleluia. Nativitas gloriose*: The three parts, *tenor*, *duplum*, and *triplum* have roughly the same number of pitches, all moving at roughly the same rhythmic rate, and are copied in score. By contrast, Figure 2.2 shows a motet based on this same passage: Now the two upper voices (the *triplum* and *motetus*) have texts underlaid throughout, and the *tenor* takes up much less space on the page than either of the two upper parts. Later in the thirteenth century, and into the fourteenth, the uppermost voice frequently took up even more space on the page. Practically speaking, in terms of production costs – parchment was the most expensive cost for makers of music manuscripts since the skin of a single animal might make just two or four folios – scribes needed to conserve manuscript space. Copying medieval motets laid out in parts instead of the older score-based layout saved a lot of parchment.

In their desire to save time and parchment, medieval scribes became experts in techniques of abbreviation. Depending on the eventual readers and purpose of the manuscript, the degree of formality of text script and abbreviation differs. In medieval music manuscripts, motet tenors were frequently copied in an abbreviated manner. Most motet tenors were composed from repeated segments of preexistent melodies, taken from plainchants or songs. In Figure 2.3a, the melodic segment is written out once, and four short vertical strokes indicate the *tenor* sings it four times; a variant on this has dots surrounding the three strokes (Figure 2.3b). In Figure 2.3c, the repeat of the *tenor* part is indicated by a rubric or canon (literally an instruction to the performer) that is an abbreviation of the Latin word *iterum* ("it.") which means 'again'. Sometimes both the sign and the rubric are found together, as shown in Figure 2.3d. Most often the entire *tenor* is repeated, but sometimes the sign and/or rubric can be found in the middle of the *tenor* part: What comes before the sign is repeated and what comes after is sung at the very end (this is the case in the *tenor* of Figure 2.3d).

In development meetings for the MP Editor, we discussed at length how to encode these *tenor* repetitions, debating the applicability of attributes such as `@corresp`, `@sameas`, and `@copyof`, and the elements `<multiRpt>`, `<rptSign>`, `<dir>`, and `<expansion>`. We wanted our encoding to do two things: (1) in the parts-based file, encode the pitches of the *tenor* exactly as they appear in the manuscript; (2) in the score-based file, actually repeat the *tenor* segment the number of times indicated. We also wanted to avoid adding new MEI elements or attributes, even though the above-mentioned elements and attributes were conceived with Common Western Music Notation (CWMN) in mind. But, in addition, ideally we wanted to include, within the parts-based file, an

5 On the manuscript layout of late medieval polyphony, see [6, 7] and several essays collected in the volume [10].

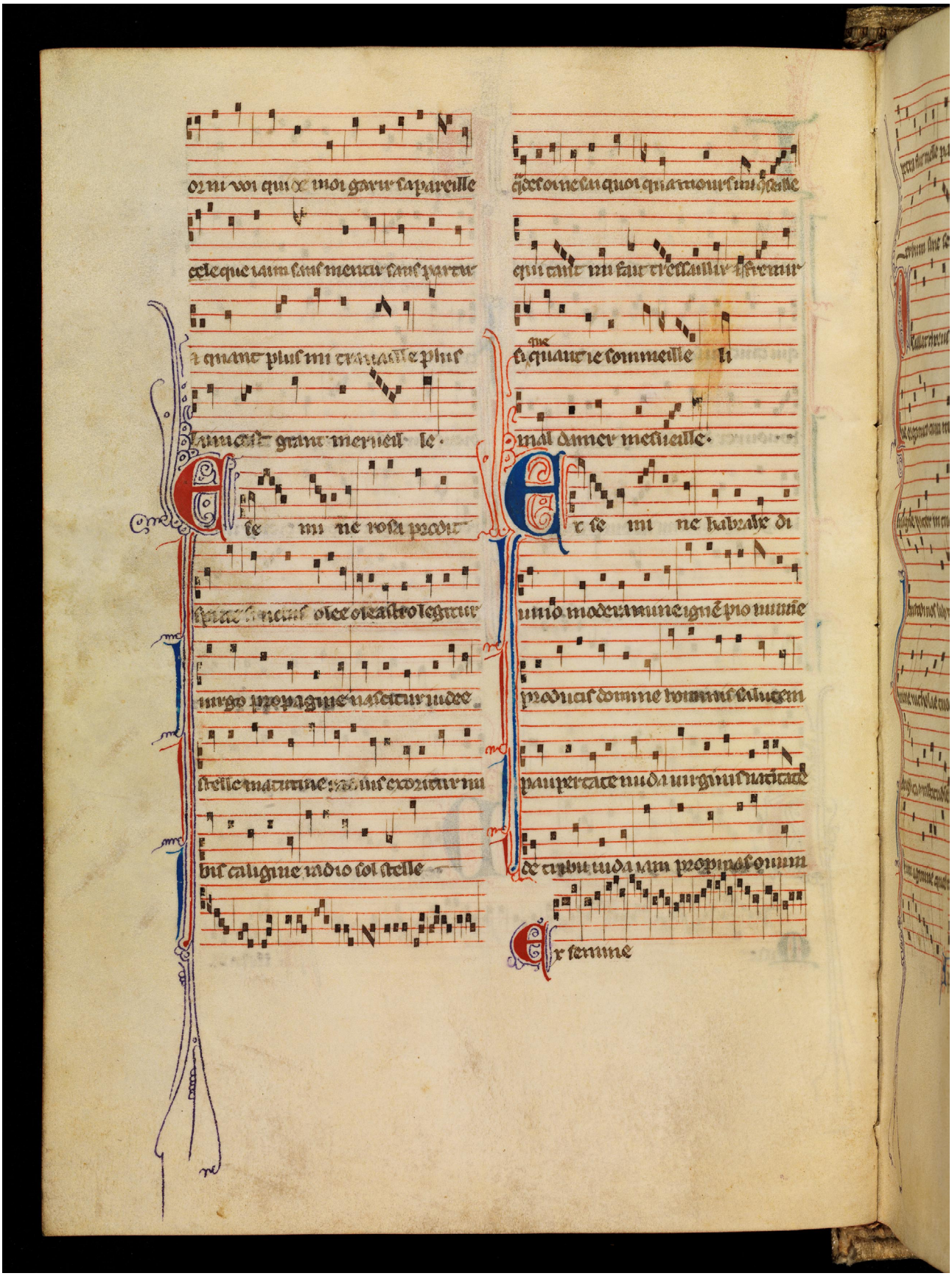


Figure 2.2: The beginning of the three voice motet *Ex semine rosa/Ex semine habrahe/EX SEMINE*, copied in parts. Staatsbibliothek Bamberg, Msc. Lit. 115, fol. 15v. Photo: Gerald Raab, http://digital.bib-bvb.de/webclient/DeliveryManager?custom_att_2=simple_viewer&pid=2957869&childpid=2957903 (CC BY-SA 4.0).

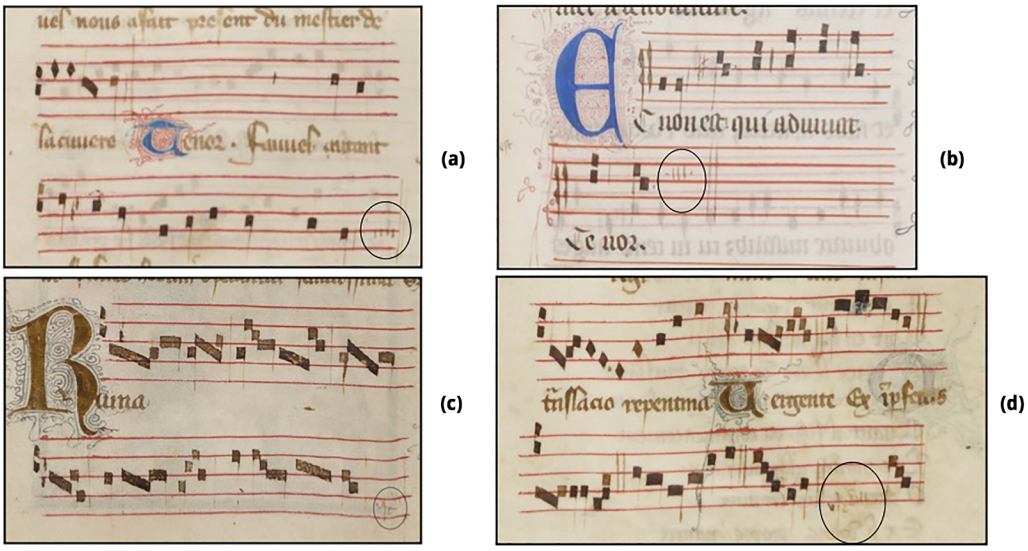


Figure 2.3: (a), (c), (d) are from Paris, Bibliothèque nationale de France, fr. 146, fols. 9v, 1v, and 3v (details), https://gallica.bnf.fr/ark:/12148/btv1b8454675g; (b) is from Paris, Bibliothèque nationale de France, fr. 1586, fol. 213v (detail), https://gallica.bnf.fr/ark:/12148/btv1b8449043q/f432.item. © gallica.bnf.fr / Bibliothèque nationale de France (license).

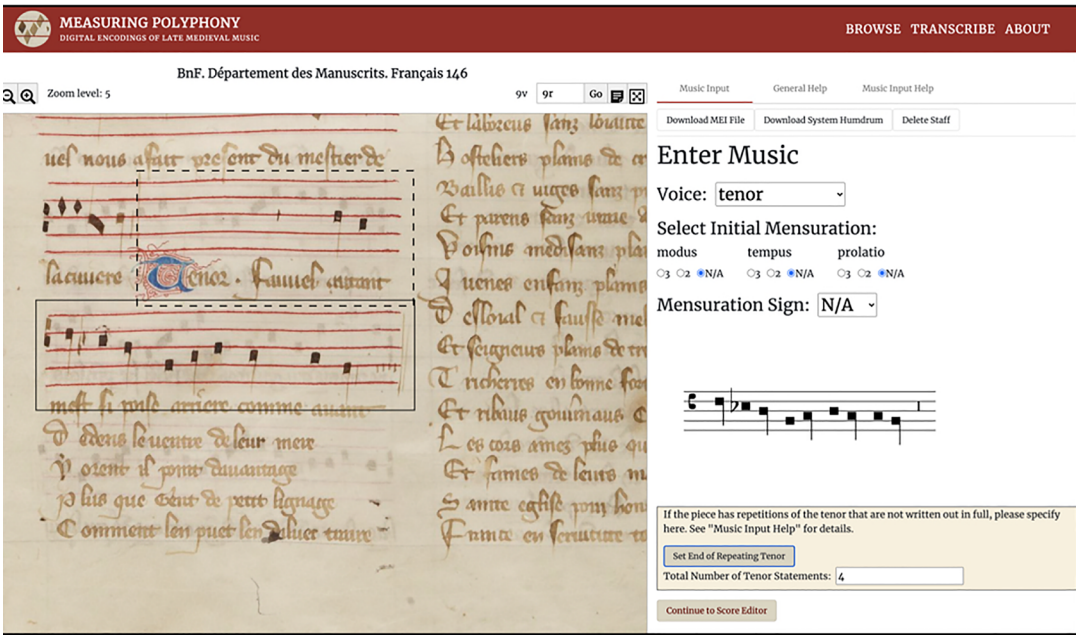


Figure 2.4a: MP Editor Interface to mark *tenor* repetitions.

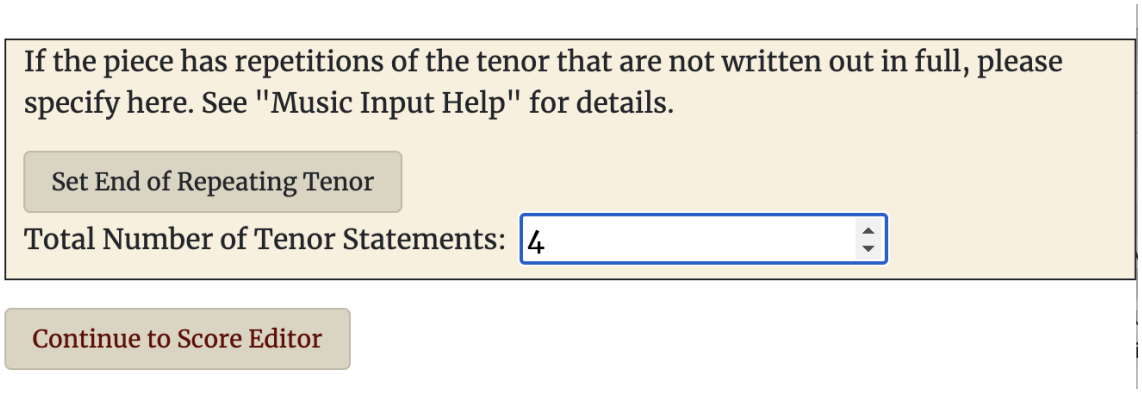


Figure 2.4b: Detail of Tenor Repetition Entry Screen.

```
<part>
  <scoreDef>
    <staffGrp>
      <staffDef n="1" lines="5" notationtype="mensural.black"
        notationsubtype="Ars_antiqua" label="tenor" xml:id="tenor">
        <mensur modusminor="3" tempus="3"/>
      </staffDef>
    </staffGrp>
  </scoreDef>
  <section>
    <staff n="1">
      <layer n="1">
        <pb facs="#m-7211a079-a7e2-4016-b3b9-74c5e4ce03bd"
          xml:id="m-53315e6e-c978-42f0-9133-915ac68d3378"/>
        <sb facs="#mc47c8bc2-ee92-4c2c-93a1-6762131aa821"
          xml:id="m-8f6c12cb-e4da-4382-86bb-57779ddb1ace"/>
        <clef xml:id="m-f89d3444-4758-42e8-af7f-c8970c60b32b" shape="C" line="3"/>
        <note xml:id="m-3a009f48-f43d-43f6-85ab-48383077d93f" dur="longa" oct="3"
          pname="f"/>
        <note xml:id="m-47c737f7-5848-4a20-a045-96ca90957426" dur="brevis" oct="3"
          pname="g"/>
        <note xml:id="m-64042039-24d3-4265-8427-77247745af22" dur="longa" oct="3"
          pname="a"/>
        <note xml:id="m-029c1ae5-9374-46d9-9e6b-52965d345b65" dur="longa" oct="3"
          pname="g"/>
        <note xml:id="m-c472a4cc-0c41-47ec-8e75-34807cf4652b" dur="longa" oct="3"
          pname="f"/>
        <rest xml:id="m-5e364b62-11ae-402b-aa39-cecfc510ab3f" dur="brevis"/>
      </layer>
      <!-- The @n attribute on <dir> is used to represent the number of repetitions
           in a machine-readable format -->
      <dir n="3" layer="1" plist="#m-3a009f48-f43d-43f6-85ab-48383077d93f
        #m-5e364b62-11ae-402b-aa39-cecfc510ab3f"
        follows="#m-5e364b62-11ae-402b-aa39-cecfc510ab3f"/>
    </staff>
  </section>
</part>
```

Listing 2.1: Code from the parts-based file that uses <dir> to mark *tenor* repetitions.

encoding of the actual symbols or text that the scribe used to indicate the *tenor* repetition. In the end, we opted to encode *tenor* repetitions using the <dir> element in the parts-based file, with the number of repetitions of the *tenor* indicated by @n, and the segment marked by including the @xml:ids of the start and end of the segment in the @plist and @follows attributes (Listing 2.1). In the score-based file, the pitches generated following the scribe's instructions are encoded by including a @copyof attribute with the @xml:id value of the pitch it repeats. The interface allows a user to mark the portions of the *tenor* that are repeated, and to indicate the number of statements of this melodic segment (Figures 2.4a and 2.4b). Future discussions with the MEI Mensural Interest Group of the Music Encoding Initiative[6] will look at the options for encoding the symbols and rubrics that indicate the repetitions in the medieval manuscript.

6 https://music-encoding.org/community/interest-groups.html (accessed January 12, 2022).

2.3 First- and Second-time Endings in Song Forms

Another standard abbreviation in manuscripts of medieval polyphony is in the notation of certain song forms: specifically those that have repeated sections, but with different first- and second-time endings. Figure 2.5 shows one manuscript source of Machaut's ballade *Esperance qui* where the first statement of the 'A' section has an open ending, and the second statement has a closed ending. Generally, medieval scribes included no indication in the manuscripts (no symbols or rubrics) that specified which pitches comprised the open and closed endings of the same section: Scribes and performers were aware of this copying/performance convention. In this case, the encoding of the parts-based and score-based files does not have to differ, since modern musicians are also usually aware of the conventions of first- and second-time endings. Currently, encoders using the MP Editor to encode medieval songs simply encode exactly what they see on the page, with the pitches for the second-time ending following directly on from the pitches of the first-time ending (see Figure 2.6). In the future, we plan to add functionality to the interface whereby the encoder could select which notes comprise the first- and second-time endings, so that these endings would be marked-up in the score-based encoding. Again, the manner of encoding will be discussed within the Mensural Interest Group of the Music Encoding Initiative.

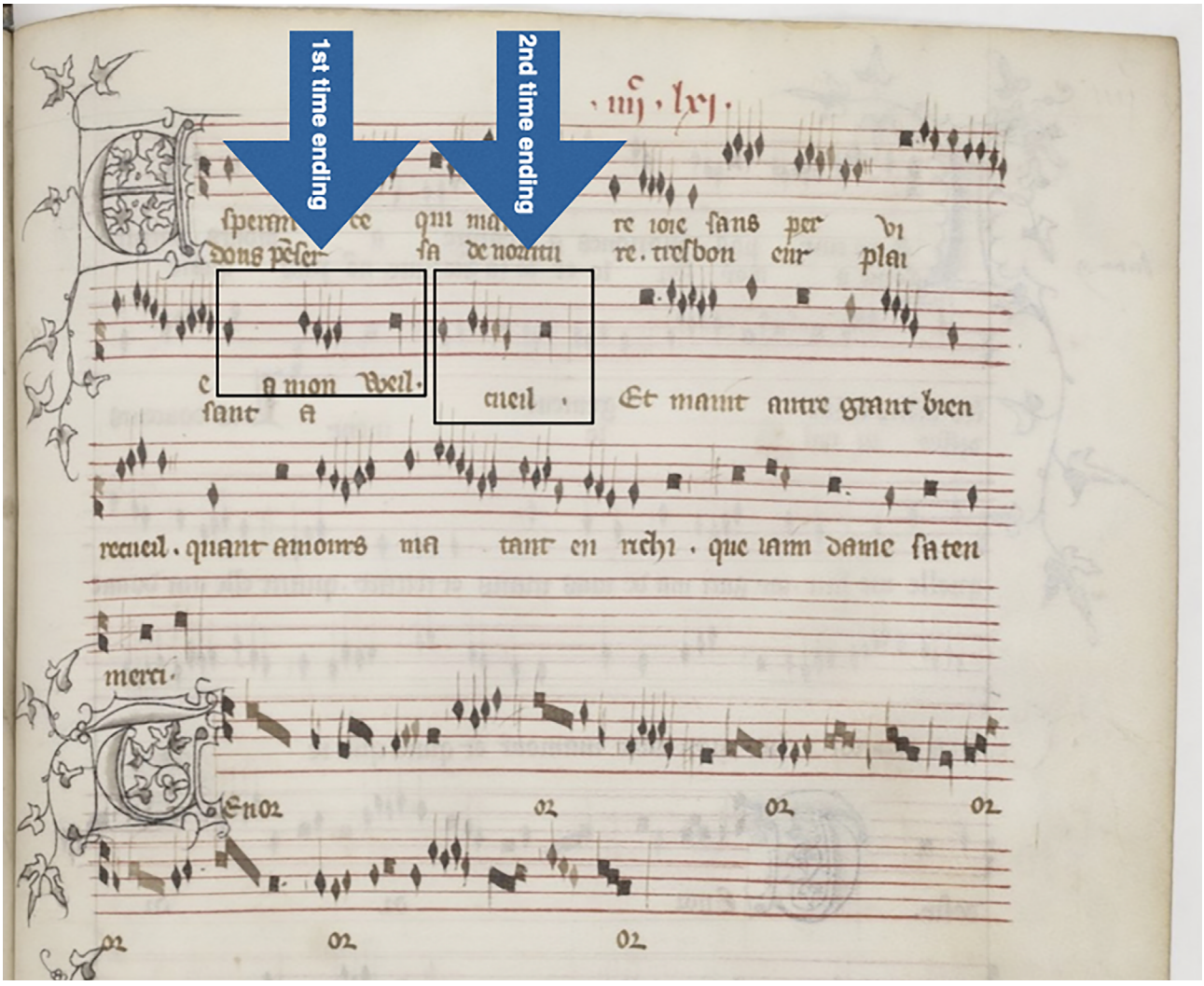


Figure 2.5: Boxes and arrows show the first- and second-time endings in Machaut's ballade *Esperance qui*, copied in Paris, Bibliothèque nationale de France, fr. 1584, fol. 461r (detail), https://gallica.bnf.fr/ark:/12148/btv1b84490444/f943. © gallica.bnf.fr / Bibliothèque nationale de France (license).

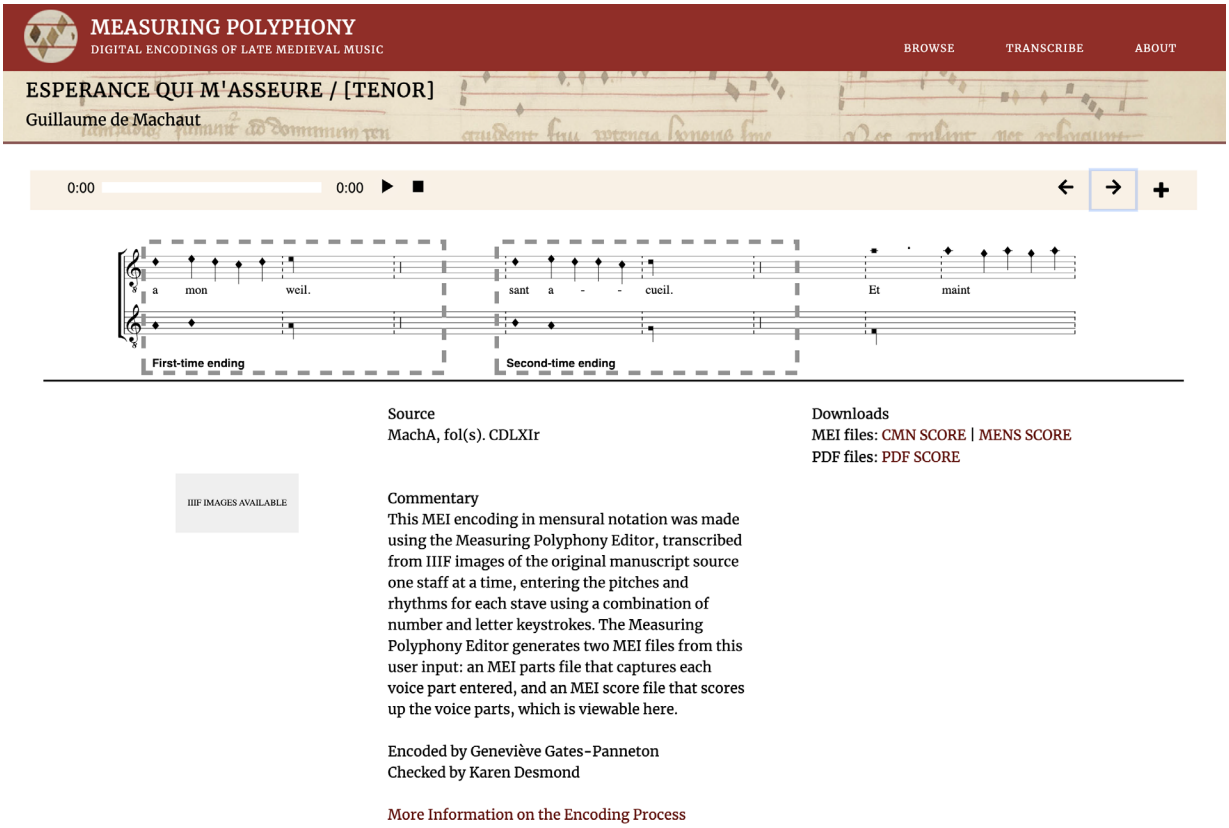


Figure 2.6: The transcription of Machaut's ballade *Esperance qui*, with the first- and second-time endings highlighted with boxes in the image.

Another fundamental way that medieval scribes could save space on the page was to use ligatures to notate pitches that were sung to a single syllable of text, although in this case this copying convention is an inheritance of neume-based plainchant notations.[7] The next section focuses on the problems of encoding and representing pitches notated with ligatures in mensural notation.

3 Bridging CWMN and Mensural Notations: Implementing the Encoding of Ligatures (Pugin)

Mensural notation available in separated parts includes several features that do not fit well with the concept of a score where all voices are vertically aligned and represented together. Nonetheless, the score representation provides the reader with a view of the music that remains essential in many situations. This is particularly true when editing mensural notation music from original sources. The score makes it much more straightforward to find the places in the transcription that need to be corrected or amended than with each voice left transcribed separately. Amongst the features of mensural notation that are problematic in score representation are ligatures. Ligatures can be represented as separated individual notes in a score, but it is often desirable to preserve in the encoding the original aspect of the ligatures in the source.

As part of the development of the MP Editor, some improvements have been made to the Verovio engraving library. Verovio is a music notation engraving library for rendering MEI scores into various digital environments.[8] Its main focus is Common Western Music Notation (CWMN) but it also supports some features of mensural notation. One of the key features of Verovio for mensural notation is its handling of triple and duple divisions. It allows for the voices to be properly aligned without having to encode in each note whether the

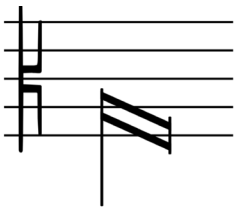
7 In addition, in mensural notation, the form of the ligature had specific meaning for the rhythmic duration of each note contained within it; in modal notation, patterns of ligatures (how many notes contained within each ligature and in what order) indicated specific rhythmic patterns.

8 https://www.verovio.org/index.xhtml (accessed January 12, 2022).

time division is triple or duple since Verovio infers this itself from the encoding of the mensuration sign(s). This proved to be very useful for the MP Editor. On the other hand, ligatures were lacking a robust implementation, and we describe in this section what had to be considered for supporting them in Verovio and the current status of the implementation. We look in particular at what the implementation of ligatures meant in terms of the underlying encoding and in terms of usability of the MP Editor.

3.1 Automatic Layout

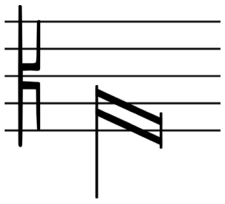
One of the main design principles at the root of the development of the Verovio engraving library is automatic layout. In addition to producing excellent quality engraving results, Verovio aims to do so without extensive engraving instructions indicating how a music notation element needs to look or how it needs to be laid out – i.e., how it needs to be arranged on the page. An important point behind this design principle is that Verovio is meant to be usable in dynamic environments featuring interactivity. The MP Editor is a typical use case where user interactions require dynamic rendering. In such a context, reducing the encoding content to the minimal information needed provides a maximal level of interactivity and flexibility.



```
<ligature form="obliqua">
  <note dur="brevis" pname="a" oct="3" stem.dir="down" stem.pos="left"/>
  <note dur="brevis" pname="f" oct="3"/>
</ligature>
```

Figure 3.1: A ligature of two *breves* with a possible MEI encoding featuring attributes about the ligature form and the stem position and direction.

What does this mean for ligatures in mensural notation? What is the minimal information needed for the rendering of ligatures? Let us consider the ligature in Figure 3.1 with a possible corresponding MEI encoding. In addition to the ligature and the notes elements with their duration and pitch information, the encoding also includes some attributes indicating the placement of the stem on the ligature, for example, that there is a stem on the left side of the first note, and that its direction is downwards. The encoding in Figure 3.1 also includes an attribute on the ligature indicating that the ligature is drawn as an oblique ligature (i.e., it has an angled shape). This is obviously all correct and appropriate. Such an encoding also has the advantage that it will make its rendering quite straightforward. From a notational point of view, however, the information is not reduced to the minimum. The fact that the ligature features a stem down on the first note and that it is oblique is in fact already given by the duration of the notes (two *breves*) and the direction of the interval their pitches yield (downwards). It means that the information needed for this ligature can be reduced. Figure 3.2 shows the MEI encoding with the minimal information Verovio expects for rendering this ligature.

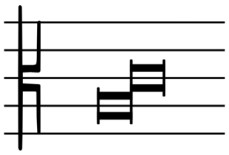


```
<ligature>
  <note dur="brevis" pname="a" oct="3"/>
  <note dur="brevis" pname="f" oct="3"/>
</ligature>
```

Figure 3.2: A ligature of two *breves* with the MEI encoding reduced to the minimal information needed for appropriate rendering.

What are the implications of this in the context of the MP Editor – or any other interactive applications? Let us imagine that the pitch of the ligature is changed so that the interval is now a third upwards as illustrated in Figure 3.3. Since attributes for the stem and about the form of the ligature were not included means that the rendering can be updated directly without having to change or remove that extraneous information. In this particular case, it means rendering the ligature as a *recta* form (proper form, i.e., not oblique) and without a

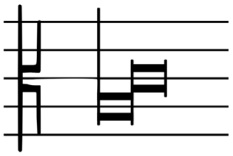
stem. Similarly, if the duration of both notes is changed from *brevis* to *semibrevis*, the automatic layout implementation of Verovio will add the upward stem to the ligature.



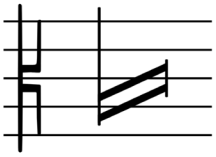
```
<ligature>
  <note dur="brevis" pname="a" oct="3"/>
  <note dur="brevis" pname="c" oct="4"/>
</ligature>
```

Figure 3.3: A ligature of two *breves* yielding an ascending interval.

There are cases, however, where the form of the ligature must be encoded when a particular shape is desired. This is the case for the *cum opposita proprietate* that can be *recta* or *obliqua*. In this case, adding an attribute to indicate the form of the ligature cannot be avoided (Figure 3.4). Similarly, for *obliqua* pairs within a ligature of more than two notes, an additional attribute must be added to indicate this.



```
<ligature>
  <note dur="semibrevis" pname="a" oct="3"/>
  <note dur="semibrevis" pname="c" oct="4"/>
</ligature>
```



```
<ligature form="obliqua">
  <note dur="semibrevis" pname="a" oct="3"/>
  <note dur="semibrevis" pname="c" oct="4"/>
</ligature>
```

Figure 3.4: *Cum opposita proprietate* is one case where the ligature form has to be explicitly encoded for distinguishing *recta* and *obliqua*.

Another layout feature that Verovio implements for ligatures is the stacking of the *longa* at the end of a ligature. Here, a difference is to be made between black and white mensural notation, and Verovio looks at the notation type for distinguishing the two cases. With black mensural notation, the last *longa* is stacked above the previous note when the interval is a third upwards or more (Figure 3.5).

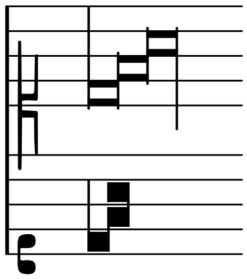


Figure 3.5: Verovio distinguishes black and white mensural notation and stacks up the *longa* at the end of the ligature when appropriate in black mensural notation.

3.2 Ligatures in a Score

When scoring up parts and aligning the voices, ligatures are problematic because they cannot reasonably be stretched to fit the score layout. One possible solution is to preserve them and to align their global duration as illustrated in Figure 3.6. This is the default behavior of Verovio when ligatures are encountered in scored-up parts.

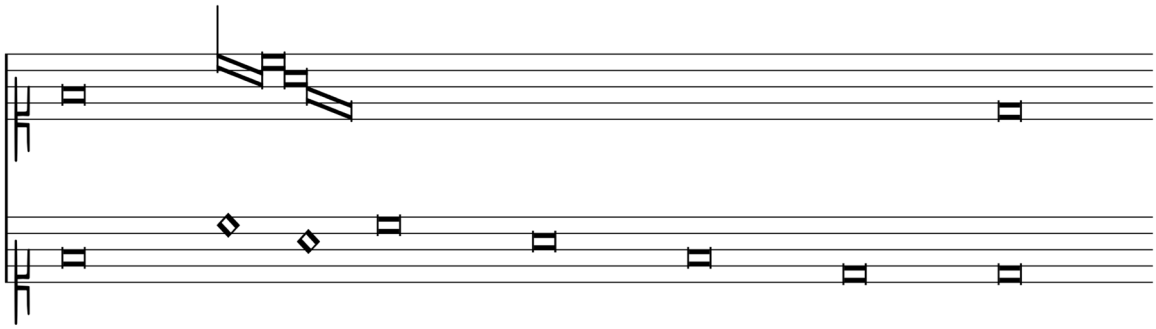


Figure 3.6: When scored up, the context around ligatures can be aligned, but not the ligatures themselves.

However, this is clearly not satisfactory because the reading of the intervals remains difficult. In order to improve it, we introduced a new option in Verovio that allows displaying ligatures with a bracket above the staff with the notes separated and aligned in the score (Figure 3.7). While the rendering with a bracket is not new and simply adopts a standard editorial practice, the originality of the approach here is that the underlying encoding of the ligature remains unchanged.

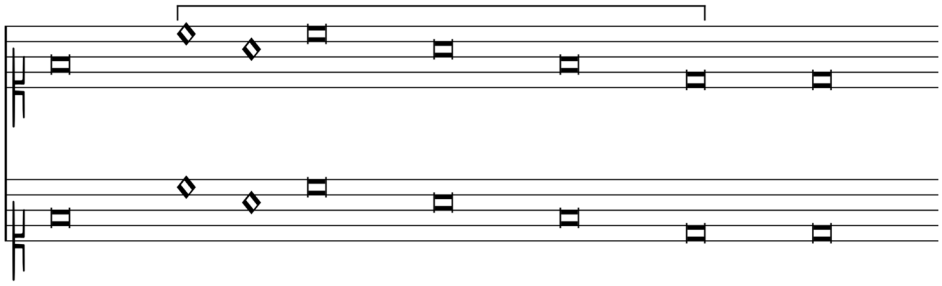


Figure 3.7: Verovio has an option to render ligatures in score as separated and aligned notes together with a bracket above the staff without having to modify the underlying encoding.

3.3 Open Questions

With the improvements made to Verovio for the MP Editor, ligatures can now be properly rendered with a minimal encoding, both in parts and in score. However, some open questions remain. One of them is what to do when the encoding is contradictory to the expected rendering rules. For example, what should happen if a ligature is marked as *obliqua* when the rendering rule would expect it to be *recta*? Is this something that can be improved in the validation of MEI through additional Schematron rules? If that is possible, do we need a way to bypass them whenever necessary, for example, where there is a mistake in a source?

Another open question is how best to deal with the problems arising when 'moving' towards a more CWMN-like score. For example, an editorial next step can be the insertion of some additional bar lines. Some of them can potentially occur within a ligature. As it stands, this can be made possible and properly rendered with Verovio as long as no system break is expected to occur within the ligature. In other words, there is currently no way to split a ligature over two systems. Such limitations, which are not unique to ligatures, can be problematic with dynamic and interactive rendering and will need to be addressed in the future.

4 From OMR to the Measuring Polyphony Editor: Possibilities of Interchange (Thomae, Rizo, and Regimbal)

Any encoding format has two aims: the preservation of the contents described by it and the possibility of interchange between computer applications. In this section, we focus on the latter. We describe the decisions made in order to export contents from the OMR research tool MuRET to the MP Editor. The MP Editor is used to curate MuRET's transcriptions by both correctly scoring up the different parts and correcting scribal errors to make digital editions. The goal of interconnecting these tools is to provide a semi-automatic way to move from digital images of mensural music to edited scores in symbolic notation.

Optical Music Recognition (OMR) applications perform automatic recognition of the music symbols on a page and usually provide an interface to correct the symbols misidentified by the automatic recognition process. The results of this process can be encoded in a format such as MEI. In mensural notation, however, correctly identifying the symbols present on the page does not convey all the rhythmic information of a piece. In triple meter, the duration implied by a particular note shape can differ in particular contexts. In addition to OMR's recognition of the note shapes and pitches, two additional encoding steps are required to make digital editions of works notated in mensural notation: (1) encoding the durations of the note shapes (*perfecta*, *imperfecta*, *altera*) based upon these contextual rules; and (2) correcting scribal errors that affect the alignment of the voices when the encodings in the parts-based files are scored up.

Rather than implementing these two extra steps in MuRET's OMR workflow (which are not generalizable to other notation systems), we use an existing technology that already handles these steps within its design, namely the Score Editor interface of the MP Editor. Here, we first present the MuRET OMR framework, and then focus on documenting the encoding decisions made on both sides (MuRET's and the MP Editor's) regarding the MEI file to be used as an interchange format between the two applications. We close with some final remarks about future work and issues that are out of the scope of this interchange process.

4.1 MuRET: Music Recognition Encoding and Transcription

The online version of MuRET (Music Recognition Encoding and Transcription)[9] is a research-oriented OMR framework [8]. It includes machine learning models to recognize the music symbols on each staff, providing two types of information per symbol: the class of symbol and its position in the staff. The encoding of these two graphical characteristics of a symbol is known as agnostic encoding [1]. The user can add, edit, and remove misidentified symbols within MuRET's interface (see Figure 4.1). These graphical symbols are converted from an agnostic encoding into a semantic encoding by interpreting information such as pitch and key signatures from the position of the symbols in the staff. The encoding of the semantics of mensural symbols is provided by **mens (see Figure 4.2), which is the Humdrum format for mensural notation [9].

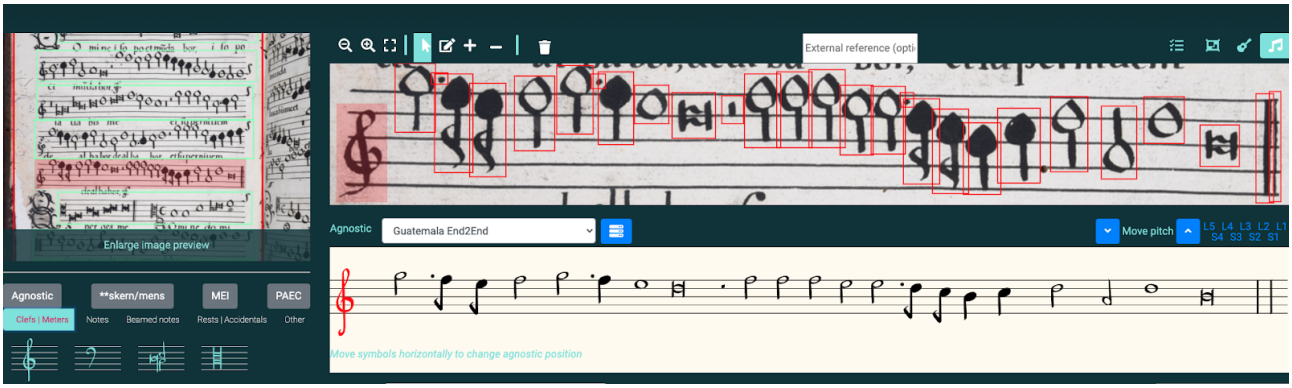


Figure 4.1: Recognition and editing of agnostic symbols in MuRET. The tool incorporates a toolbox for correcting mistakes or even adding them from scratch.

9 https://muret.dlsi.ua.es/muret/ (accessed January 12, 2022).

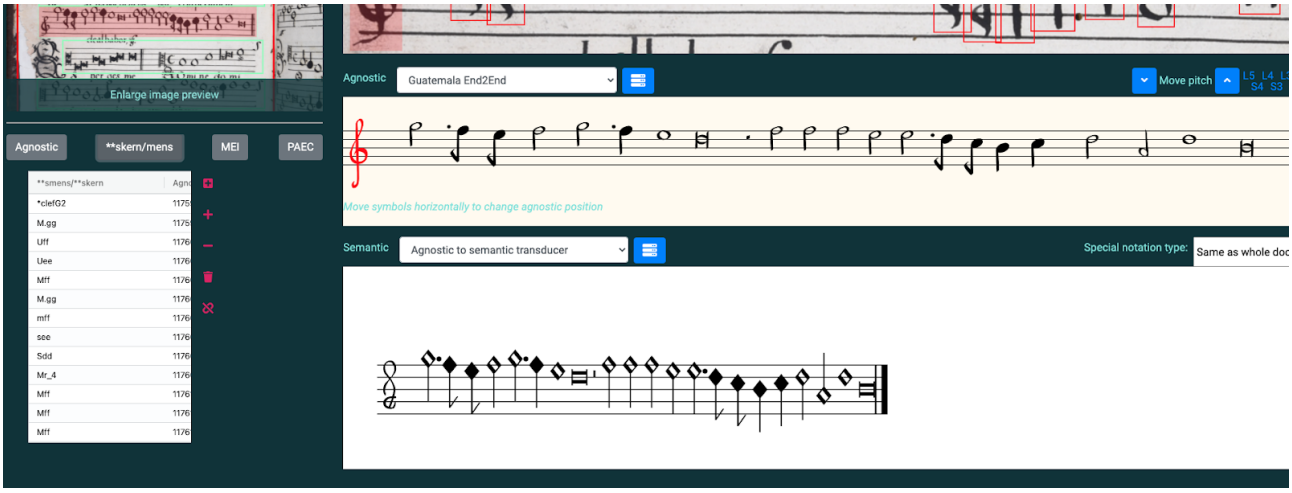


Figure 4.2: Conversion of agnostic symbols (middle staff) into a semantic representation (bottom staff) that is internally encoded using **mens (bottom left).

4.2 Encoding Decisions for the MEI File

We used MEI as the interchange format between MuRET and the MP Editor. In this section, we present the encoding decisions made regarding the MEI file exported by MuRET so that it worked as input for the MP Editor. The decisions are divided into the following four categories:

4.2.1 Use of IIIF Manifest and Server to Share the Transcribed Document Between Applications

Following the philosophy of the International Image Interoperability Framework (IIIF) [11], rather than implementing ad-hoc image servers for each tool and developing a system for interchanging image between each application, the IIIF approach has been used for storing and serving images that can be accessed not only from MuRET and the MP Editor, but also from any other system that could need them in the future.

When importing new musical works to be transcribed with MuRET, images are uploaded into a IIIF-compliant system named Cantaloupe.[10] Currently, MuRET just accesses the images using hardcoded IIIF URIs. When exporting the MEI file obtained from the recognition process, MuRET generates and stores a IIIF manifest whose URI is encoded in the MEI header, inside the `<source>` child of the `<fileDesc>` element identified as IIIF by using the `@targettype="IIIF"` attribute (see Listing 4.1).

```
<mei>
  <meiHead>
    <fileDesc>
      ...
      <sourceDesc>
        <source target="https://<IIIF-server>/<work-name>/manifest.json"
          targettype="IIIF"/>
```

Listing 4.1: Reference within the MEI header to the IIIF manifest.

4.2.2 Facsimile Encoding Decisions

In order to be able to recognize music in a digitized image, the OMR process in MuRET first divides the image into regions of interest such as pages, staves, notes, clefs, etc., that are delimited by bounding boxes. This graphical information is encoded by using the `<facsimile>` element of MEI (see Listing 4.2) and includes the URL of the source image and the bounding boxes of those regions of interest.

Source images are linked using a IIIF URI in the `@target` attribute of the `<graphic>` child of the `<surface>` facsimile element. All bounding boxes of regions of interest that have been identified in the image are specified through `<zone>` tags with their corresponding coordinate attributes (`@ulx`, `@uly`, `@lrx`, `@lry`). Three

10 https://cantaloupe-project.github.io (accessed January 12, 2022).

kinds of zones are used with the `@type` attribute: "page", "region", or "symbol". The former is used to delimit the pages shown in the image (usually just one or two of them). The type "region" is used for all other zones, different from individual symbols, that are now differentiated by using the `@label` attribute with values "staff", "title", "drawing", "author", etc. Finally, zones identified containing symbols such as clefs, mensuration signs, accidentals, dots, or notes are also delimited by bounding boxes with the type "symbol" and a label containing the agnostic string representation of the symbol (e.g., "clef.C:L4").

Some OMR models are not able to recognize the exact symbol bounding box, but an approximate one. In that case, the bounding boxes encode the rectangle defined from the identified horizontal position of the symbol until the next one.

```
<music>
  <facsimile>
    <surface xml:id="image_2728" ulx="0" uly="0" lrx="1335" lry="2000">
      <graphic target="<IIIF-server>/<work-name>/canvas/f1" xml:id="graphic_21083"/>
        <zone xml:id="page_2964" ulx="47" uly="24" lrx="1128" lry="1682"
          type="page" label="Page #1"/>
        <zone xml:id="region_25073" ulx="240" uly="123" lrx="1105" lry="267"
          type="region" label="staff"/>
        <zone xml:id="symbol_116997" ulx="245" uly="173" lrx="282" lry="264"
          type="symbol" label="clef.C:L4"/>
```

Listing 4.2: The MEI facsimile element as generated from MuRET.

All musical contents include references to the graphical information in the `<facsimile>` by using the `@facs` attribute. Instead of using the specified bounding boxes, the MP Editor uses the page beginning (`<pb>`) and system beginning (`<sb>`) references for graphically delimiting the start of each page and system (see Listing 4.3). More details about the use of `<pb>` and `<sb>` are given below.

```
</facsimile>
…
<section>
  <staff>
    <layer>
      <pb xml:id="spb_21356" facs="#page_2964"/>
      <sb xml:id="spb_21357" facs="#region_25073"/>
      <clef line="4" shape="C" facs="#symbol_116997" xml:id="clef_21358"/>
```

Listing 4.3: Graphical content linked from musical MEI elements describing musical content.

4.2.3 Structure of the Mensural MEI Parts File

The MEI file used for interchange between MuRET and the MP Editor is a *parts-based* MEI file. This is an MEI file that encodes the musical content using the `<parts>` element rather than the `<score>` element. This `<parts>` element is intended to encode each of the parts (i.e., voices) individually within a `<part>` element. The parts-based representation reflects exactly what is on the manuscript – most mensural sources are notated in separate parts. This representation is ideal as output for MuRET since, at this point in the workflow, we only have a sequence of music symbols for each part without any note duration information that would allow for lining up the piece as a score – the duration information will be determined by the MP Editor.

The decisions regarding the encoding of the `<part>` elements within `<parts>` may be summarized as follows (note that each `<part>` follows the same structure):

- Each `<part>` has one `<scoreDef>` element containing a single `<staffDef n="1">` (see purple area in Listing 4.4).
- Each `<part>` has one `<section>` element containing a single `<staff n="1">` (see yellow area in Listing 4.4).

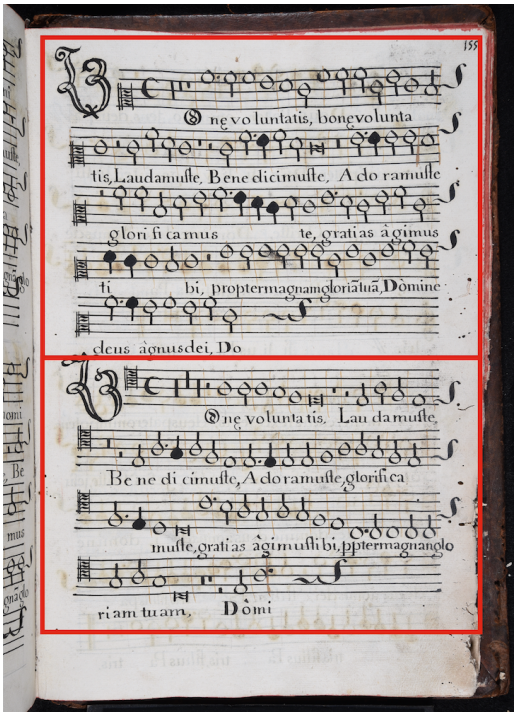
```
<body>
  <mdiv>
    <parts>
      …
      <part/>
      <part>
        <scoreDef>
          <staffGrp>
            <staffDef n="1"></staffDef>
          </staffGrp>
        </scoreDef>
        <section>
          <staff n="1">
            <layer n="1">
            <!-- music content -->
            </layer>
          </staff>
        </section>
      </part>
      <part/>
      <part/>
    </parts>
  </mdiv>
</body>
```

Listing 4.4: Structure of each <part> element.

The decisions made regarding the placement of certain elements within a <part> are the following:

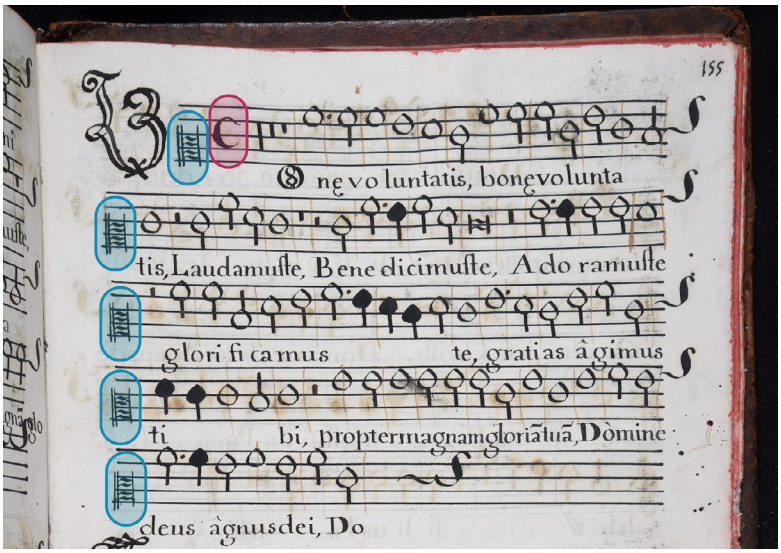
- *Page-beginning elements* (<pb>): The <pb> elements do not mark a physical page beginning; instead, they are placed in each voice (i.e., part) to mark the page beginning *for that voice*. As an example, the encoding below would work for both voices shown in Figure 4.3 (regardless of which voice is placed at the physical beginning of the page). The <pb> elements are placed at the very beginning of each voice (as the first child of <layer>) and at every page turn. The 'absolute' page beginning can be found using encoded facsimile information for each part.
- *Clef (<clef>) and key signature (<keySig>) elements and their relationship with the system beginning element (<sb>):* In the original sources, clefs and key signatures are written at the beginning of each system of a voice. Because of this, we decided to encode the <clef> and <keySig> information following each <sb> element within <layer> (see Listing in Figure 4.4). The <clef> and <keySig> elements point to the physical location in the corresponding system in the facsimile through the @facs attribute (see blue zones in Figure 4.4).
- *Mensuration element (<mensur>):* Contrary to the behavior of clefs and key signatures, mensuration signs are given only at the beginning of the first system of a voice. Because of this, we decided to encode the <mensur> element once within the <staffDef> element. This <mensur> element also points to its location at the beginning of the first system of the piece through its @facs attribute (see magenta zone in Figure 4.4). In the case of a change in mensuration, the new <mensur> element representing the new mensuration sign is encoded within the stream of notes inside <layer>.

To illustrate the encoding of clefs (and key signatures) vs. mensuration signs, see the MEI excerpt in Figure 4.4 used to encode the first voice in Figure 4.3.



```
<part>
  <scoreDef/>
  <section>
    <staff n="1">
      <layer n="1">
        <pb/>
        <sb/>
        <clef/>
        <!-- notes in the 1st system -->
        <sb/>
        <clef/>
        <!-- notes in the 2nd system -->
      </layer>
    </staff>
  </section>
</part>
```

Figure 4.3: Manuscript page with two voices shown in red rectangles (left). Page-beginning element (<pb>) marking the "page beginning" of a voice (right).



```
<part>
  <scoreDef>
    <staffGrp>
      <staffDef n="1">
        <mensur/>
      </staffDef>
    </staffGrp>
  </scoreDef>
  <section>
    <staff n="1">
      <layer n="1">
        <pb/>
        <sb/>
        <clef/>
        <!-- notes in the 1st
        system of the voice -->
        <sb/>
        <clef/>
        <!-- notes in the 2nd
        system of the voice -->
      </layer>
    </staff>
  </section>
</part>
```

Figure 4.4: First voice of Figure 4.3, showing the zones for the mensuration sign (magenta box) and the clefs (blue boxes) (left). These zones correspond to the one <mensur> element and the multiple <clef> elements (right). <mensur> is encoded once in <staffDef> and related to the zone where it appears. On the other hand, <clef> is encoded several times, one after each <sb> (within <layer>), and each clef is related to its zone at the beginning of the system where it appears.

4.2.4 Encoding of Unqualified Durations and Dots

None of the notes in the parts-based MEI file exported by MuRET provide a `@dur.quality` attribute indicating the relative duration of a note. The value of this attribute ("perfecta", "imperfecta", or "altera") is meant to be found and encoded within the <note> elements by the MP Editor's scoring-up functionality. In a similar way, all <dot> elements coming out from MuRET do not include a `@form` attribute clarifying whether the dot is behaving as a dot of division (`@form="div"`) or a dot of augmentation (`@form="aug"`). The nature of the dots, just as the quality of the notes, is computed and encoded by the MP Editor.

4.3 Future Work and Remaining Issues

In the transfer from MuRET to the MP Editor, a few details get lost: the presence of barlines (<barLine>), the stem direction of notes (`@stem.dir`), the location of the rests (`@loc`), and the bounding boxes (the <zone> elements) of individual music symbols. The first three can be easily fixed since barlines, stem directions, and rest positions are supported in **mens, which is the internal representation used in both MuRET and the MP Editor (as pointed out in sections 4.1 and 1.1, respectively). The only change needed is to add functionality in the MP Editor to parse these attributes and/or elements coming from the uploaded MEI file.

On the other hand, encoding the zones of each symbol is not within the scope of the MP Editor. In the Input Editor interface of the MP Editor, the user draws bounding boxes for each system and enters (types in) the symbols – clefs and notes – belonging to that system. As such, entry of symbol-level workflow is unsupported in the current interface. Despite this, the <zone> elements for each symbol in MuRET's MEI file can easily be recovered by running a post-processing script on the output of the MP Editor.

Another minor issue is that a few elements from the uploaded parts-based MEI file change the `@xml:id` values in the MP Editor.[11] This has no effect, however, on the quality of the edited score exported by the MP Editor. Part of future work could devise a way to avoid any loss or change in the information provided in the uploaded parts-based MEI file. One proposal would be to directly access the MP Score Editor and avoid passing through the Input Editor part that internally encodes the uploaded information into Humdrum (see Figure 1.1).

Here we have presented the decisions that allow for MuRET and the MP Editor to communicate with each other in order to generate edited scores with OMR assistance. Each tool is responsible for a different task; MuRET deals with symbol recognition and the MP Editor with automatic voice alignment and editorial corrections. Their integration allows us to move from digital images of mensural music to symbolic edited scores in a semi-automatic way. Future work would entail improving the information interchange between MuRET and the MP Editor to support more varied workflows.

5 Interaction between Measuring Polyphony MEI Scores and Humdrum Analysis Tools (Thomae, Sapp, and Regimbal)

The MP Editor is used to generate edited scores encoded in mensural MEI. The Score Editor interface has an editorial mode for correcting scribal errors. The MP Editor facilitates the editorial process through (1) rendering the piece as a score; and (2) allowing the user to bar the transcription by different mensural note values (semibreves, breves, longs). Computational music analysis tools can also assist in the editorial process by providing information about counterpoint. We present the interaction of Humdrum analysis tools with the MP Score Editor to facilitate the detection of voice-alignment errors. The first section introduces the analysis tool used and the goal behind its inclusion into the MP Editor, while the second section focuses on the implementation details.

5.1 Dissonance Analysis in the MP Editor

The score layout of the voices in the MP Score Editor interface can reveal counterpoint errors that are useful in detecting mistakes in voice alignment. These mistakes might be due to scribal errors, user-entry errors, or

11 Some of the elements that do change their xml ids are <dot>, <accid>, and <ligature>. These elements represent features that are encoded in **mens within the same token as the note they relate to (e.g., Sg. for a note followed by a dot, Lf# for a note with an accidental, and [sg and sa] for the notes at the beginning and ending of a ligature). Other elements that change xml ids are the children of facsimile (<surface>, <graphic>, and <zone>), the page and system beginning elements (<pb> and <sb>), and the lyric-related elements (<verse> and <syl>).

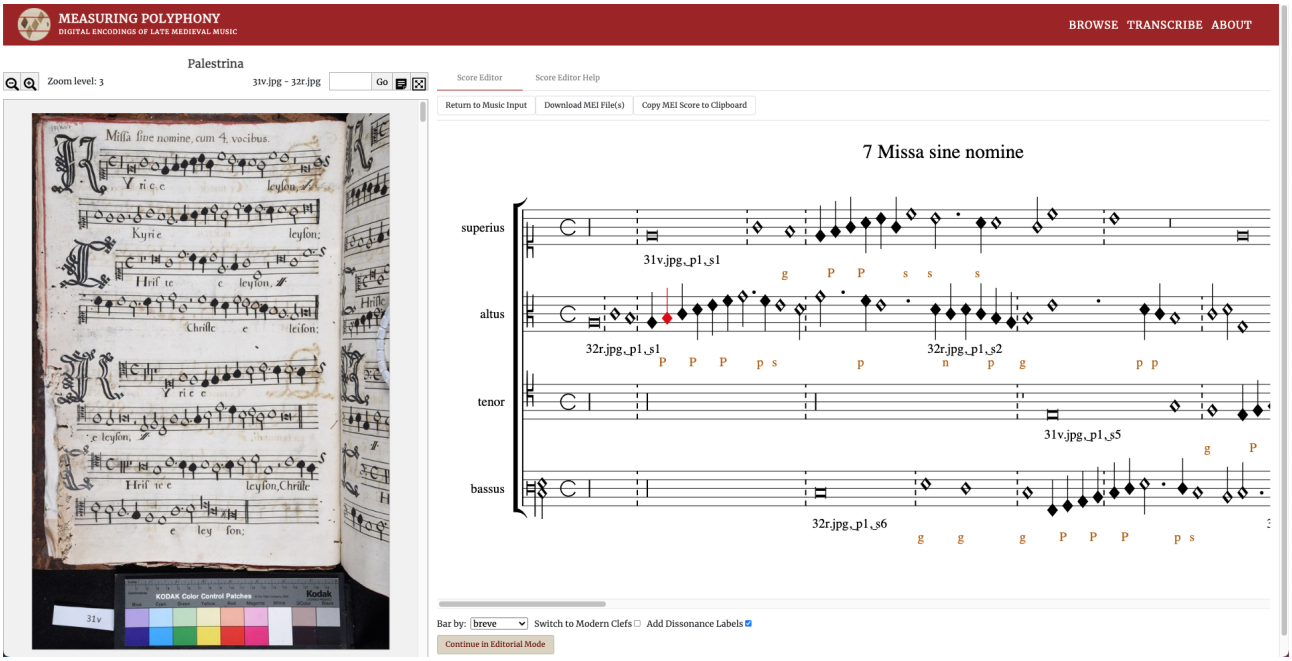


Figure 5.1: MP Score Editor interface with the "Add Dissonance Labels" checkbox activated. The dissonance labels are shown below each voice in a dark orange color. The description of these labels can be found at https://doc.verovio.humdrum.org/filter/dissonant/#dissonant-function-labels (accessed January 12, 2022).

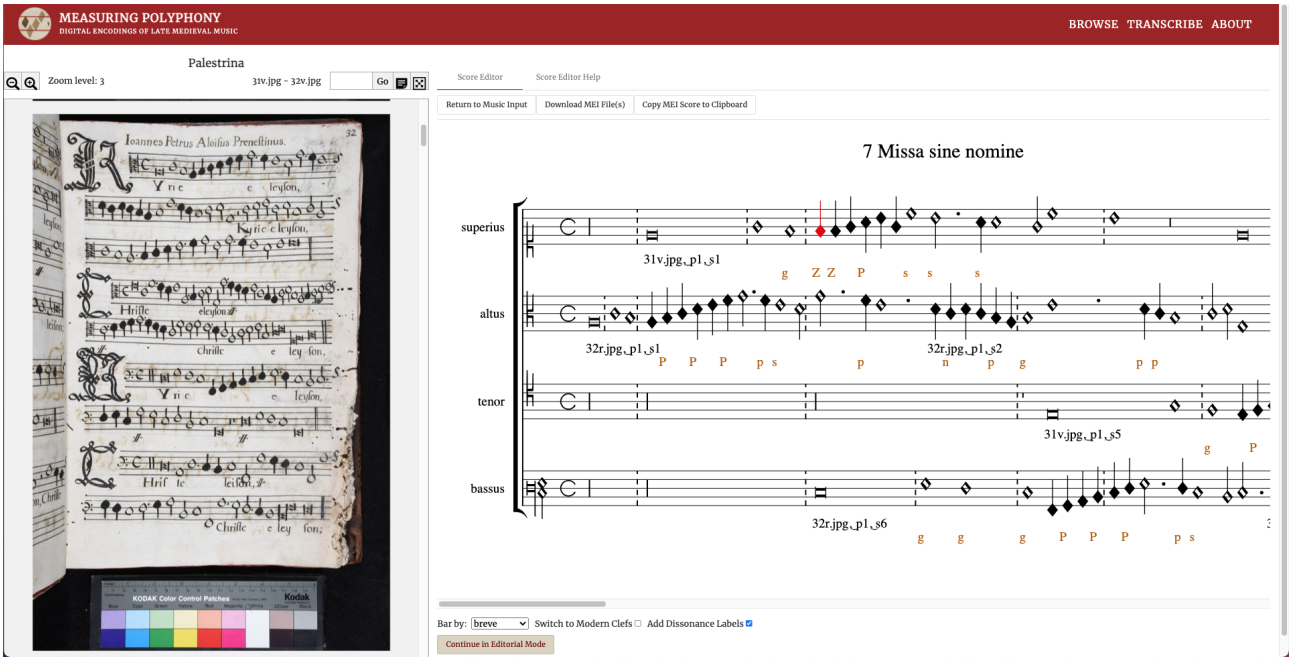


Figure 5.2: The *Z* label in the highlighted note in the *superius* points to a voice-alignment error. The *superius* forms a second with the bass (and the *altus*) in a strong beat, which is an uncommon dissonance for Renaissance style. The voice-alignment error, revealed by this counterpoint error, was due to a user-entry error.

errors in the scoring-up algorithm of the MP Editor. Detecting voice-alignment errors is useful in the process of producing an edited score – where all notes and their interpreted durations are accurate and scribal errors have been corrected.

We have implemented functionality in the Score Editor interface to facilitate the capture of errors in voice alignment by using counterpoint cues, specifically, dissonances. This functionality shows the dissonances in the score (see Figure 5.1), where the definition of 'dissonance' follows the generally understood conventions of Renaissance counterpoint. The dissonance labels used come from the dissonant function labels of the Verovio Humdrum Viewer's (VHV) "dissonant filter" (note that while the fundamentals of contrapuntal writing emerged

in the fourteenth century, many of these labels and contrapuntal concepts will not apply to music before the fifteenth century).[12]

The *Z*/*z* labels indicate an "unclassified dissonance" according to Renaissance style. Since this type of dissonances rarely occurs in Renaissance music, their presence in a piece might indicate an error in voice alignment, as illustrated in Figure 5.2. The appendix provides a pair of graphs, generated from data in the Josquin Research Project[13] and Tasso in Music Project,[14] showing the frequency distribution of the various dissonance labels during two different periods in the Renaissance.

The VHV "Renaissance dissonance labels" filter only works on **kern files, which is Humdrum format for Common Western Music Notation (CWMN). Since the output of the MP Score Editor is mensural MEI, we converted these MEI scores into **kern through a series of processes in order to have access to the "Renaissance dissonance labels" filter. The following section presents the details about this conversion process and the insertion of the dissonance labels into the MEI file for rendering them within the MP Score Editor.

5.2 Implementation Details

The mensural MEI score of the MP Editor is converted into **mens (Humdrum format for mensural notation), and then into **kern (Humdrum for CWMN). The `@xml:id`s of the notes are preserved in this conversion process. Once in **kern, the dissonant filter is used and both the dissonance labels and `@xml:id`s are retrieved and encoded in a JSON file. These JSON-encoded labels can then be applied back to create a new, annotated MEI file. The saved `@xml:id` data are used to find the correct notes in the original MEI file using XPath. A new verse is inserted for these notes in a distinct color to display the dissonance labels in a clear way to the user. The process can be seen in Figure 5.3.

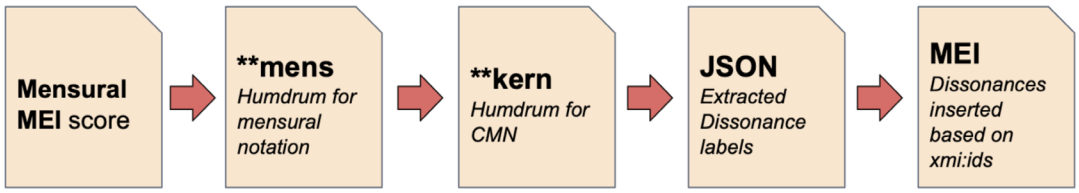


Figure 5.3: Conversion process starting from the MP Editor's Mensural MEI score to retrieve the dissonance labels to render in the MP Editor.

Appendix

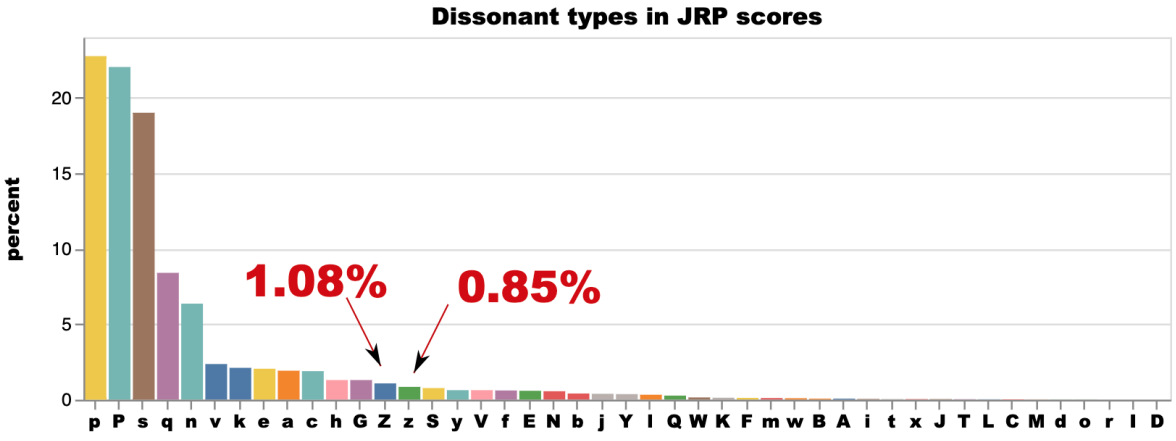


Figure 5.4: Dissonance label frequencies in Josquin Research Project scores (1480–1520). The *Z* label has a frequency of 1.08%, and the *z* label has a frequency of 0.85%.

12 https://doc.verovio.humdrum.org/filter/dissonant/ (accessed January 12, 2022).

13 https://josquin.stanford.edu/ (accessed January 12, 2022).

14 https://www.tassomusic.org/ (accessed January 12, 2022).

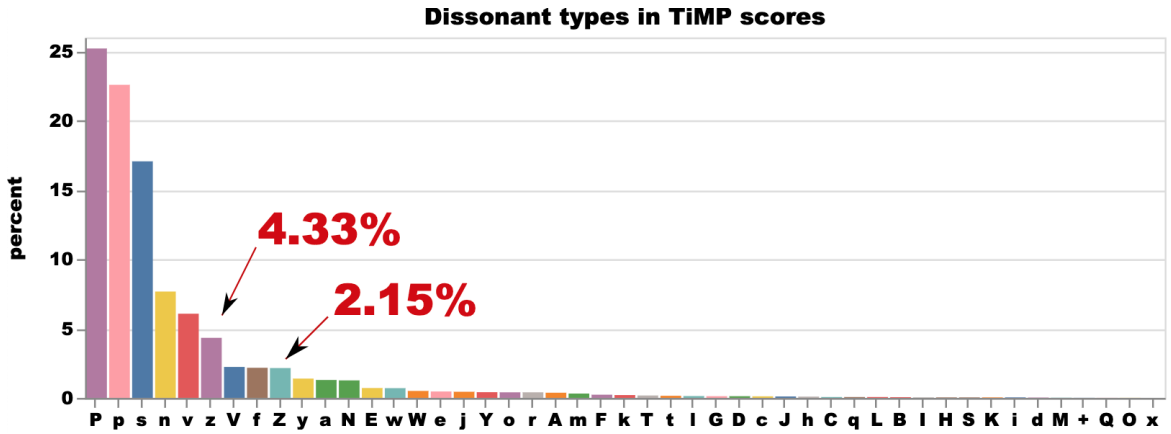


Figure 5.5: Dissonance label frequencies in Tasso in Music Project scores (primarily 1580–1600). The *Z* label has a frequency of 2.15%, and the *z* label has a frequency of 4.33%. The high increase in the use of perfect fourths above the bass (*z*) might indicate that the interval was considered less dissonant in this later period (compare with Figure 5.4).

Acknowledgments

The authors gratefully acknowledge The National Endowment for the Humanities, Fonds de recherche du Québec – Société et culture (FRQSC), Bourse au doctorat en recherche (13D - Musique) 2019-B2Z-261749, Alex Morgan, postdoc for the Josquin Research Project (2017), for his work on the "Renaissance dissonance labels" filter in collaboration with Craig Sapp, the Spanish Ministry HISPAMUS project TIN2017-86576-R, and the MultiScore Project, I+D+i PID2020-118447RA-I00, funded by MCIN/AEI/10.13039/50110001103.

Works Cited

[1] Calvo-Zaragoza, Jorge, and David Rizo. "End-to-End Neural Optical Music Recognition of Monophonic Scores" *Applied Sciences* 8, no. 4 (2018), 606–623, https://doi.org/10.3390/app8040606.

[2] Clark, James, and Steve DeRose. "XML Path Language (XPath)" W3C World Wide Web Consortium Recommendation 16 November 1999, https://www.w3.org/TR/1999/REC-xpath-19991116/.

[3] Desmond, Karen, Andrew Hankinson, Laurent Pugin, Juliette Regimbal, Craig S. Sapp, and Martha E. Thomae. "Measuring Polyphony: An Online Editor for Medieval Music" White Paper, The National Endowment for the Humanities, 2018, https://securegrants.neh.gov/publicquery/main.aspx?f=1&gn=HAA-263800-19.

[4] Earp, Lawrence M. "Notation II" in *The Cambridge History of Medieval Music*, ed. Mark Everist and Thomas Kelly. Cambridge: Cambridge University Press, 2018, 674–717 https://doi.org/10.1017/9780511979866.023.

[5] Grier, James. *Musical Notation in the West*. Cambridge: Cambridge University Press, 2021.

[6] Haines, John, and Stefan Udell. "Motets, Manuscript Culture, Mise-en-page" in *A Critical Companion to Medieval Motets*, ed. Jared Hartt. Woodbridge: The Boydell Press, 2018, 175–192.

[7] Huck, Oliver. "The Layout of the Early Motet" *The Journal of the Alamire Foundation* 7, no. 1 (2015), 11–32, https://doi.org/10.1484/J.JAF.5.103802.

[8] Rizo, David, Jorge Calvo-Zaragoza, and José Manuel Iñesta. "MuRET: a Music Recognition, Encoding, and Transcription Tool" in *Proceedings of the 5th International Conference on Digital Libraries for Musicology* (DLfM 2018), 52–56, https://doi.org/10.1145/3273024.3273029.

[9] Rizo, David, Nieves Pascual-León, and Craig S. Sapp. "White Mensural Manual Encoding: from Humdrum to MEI" *Cuadernos de Investigación Musical* 6 (2018), 373–393, https://doi.org/10.18239/invesmusic.v0i6.1953.

[10] Schmidt, Thomas, and Christian T. Leitmeir, eds. *The Production and Reading of Music Sources: Mise-En-Page in Manuscripts and Printed Books Containing Polyphonic Music, 1480–1530.* Turnhout: Brepols Publishers, 2018.

[11] Snydman, Stuart, Robert Sanderson, and Tom Cramer. "The International Image Interoperability Framework (IIIF): A Community & Technology Approach for Web-based Images" in *Proceedings of the IS&T Archiving Conference* (ARCHIVING 2015), 16–21.