THE MUSIC ENCODING INITIATIVE AS A DOCUMENT-ENCODING FRAMEWORK

Andrew Hankinson[1] **Perry Roland**[2] **Ichiro Fujinaga**[1]
[1]CIRMMT / Schulich School of Music, McGill University
[2]University of Virginia
andrew.hankinson@mail.mcgill.ca, pdr4h@eservices.virginia.edu, ich@music.mcgill.ca

ABSTRACT

Recent changes in the Music Encoding Initiative (MEI) have transformed it into an extensible platform from which new notation encoding schemes can be produced. This paper introduces MEI as a document-encoding framework, and illustrates how it can be extended to encode new types of notation, eliminating the need for creating specialized and potentially incompatible notation encoding standards.

1. INTRODUCTION

The Music Encoding Initiative (MEI)[1] is a community-driven effort to define guidelines for encoding musical documents in a machine-readable structure. The MEI closely mirrors work done by text scholars in the Text Encoding Initiative (TEI)[2] and while the two encoding initiatives are not formally related, they share many common characteristics and development practices.

MEI, like TEI, is an umbrella term to simultaneously describe an organization, a research community, and a markup language [1]. It brings together specialists from various music research communities, including technologists, librarians, historians, and theorists in a common effort to discuss and define best practices for representing a broad range of musical documents and structures. The results of these discussions are then formalized into the MEI schema, a core set of rules for recording physical and intellectual characteristics of music notation documents. This schema is developed and maintained by the MEI Technical Group.

The latest version of the MEI schema is scheduled for release in Fall 2011. The most ambitious feature of the 2011 release is the transformation of the MEI schema from a single, static XML schema language to an extensible and customizable music document-encoding framework. This framework approach gives individuals a platform on which to build custom schemas for encoding new types of music documents by adding features that support the unique aspects of these documents, while leveraging existing rules and guidelines in the MEI schema. This eliminates the duplication of effort that comes with building entire encoding schemes from the ground up.

In this paper we introduce the new tools and techniques available in MEI 2011. We start with a look at the current state of music document encoding techniques. Then, we discuss the theory and practice behind the customization techniques developed by the TEI community and how their application to MEI allows the development of new extensions that leverage the existing music document-encoding platform developed by the MEI community. We also introduce a new initiative for sharing these customizations, the *MEI Incubator*. Following this, we present a sample customization to illustrate how MEI can be extended to more accurately capture new and unique music notation sources. We then introduce two new software libraries written to allow application developers to add support for MEI-encoded notation. Finally, we end with a discussion on how this will transform the landscape for music notation encoding.

2. MUSIC NOTATION ENCODING

There have been many attempts to create structural representations of music notation in machine-readable formats [2]. Some formats, like **kern or MuseData, use custom ASCII-based structures that are then parsed into machine-manipulable representations of music notation. Others, like NIFF or MIDI, use binary file formats. In recent years, XML has been the dominant platform for structural music encoding, employed by initiatives like MusicXML[3] and IEEE1599[4].

The wide variety of encoding formats and approaches to music representation may be attributed to the complexity of music notation itself. Music notation often conveys meaning in multiple dimensions. Variations in placement on the horizontal or vertical axes manifest different dimensions in meaning, along with size, shape, colour, and spacing. To these, however, are added cultural and temporal dimensions that result in different types of music notation expressing different meanings through visually similar notation, depending on where and when that notation system was in use. This complexity prohibits the

[1] http://www.music-encoding.org
[2] http://www.tei-c.org
[3] http://www.recordare.com/musicxml
[4] http://www.mx.dico.unimi.it/index.php

construction of a single, unified set of rules and theories about how music notation operates without encountering contradictions and fundamental incompatibilities between notation systems. Consequently, formulating a single, unified notation encoding scheme for representing the full breadth of music notation in a digital format becomes very difficult.

As a result, representing music notation in a computer-manipulable format generally takes two approaches. The first approach is to identify the greatest amount of commonality among as many different types of music as possible, and target a general encoding scheme for all of them. The consequence is a widely accepted encoding scheme, which serves as a system that is "good enough" to represent common features among most musical documents, but extremely poor at representing the unique features that exist in every musical document. For example, the MIDI system of encoding pitch and timing as a stream of events functions very well if the only musical elements of interest are discrete volume, timing, and pitch values. It is, however, notoriously poor at representing features like phrase markings or distinctions between enharmonic pitch values.

The second general approach is to build an encoding system that takes into account all the subtle variation and nuance that makes a particular form of music notation different from all others. With this approach, highly specialized methods for encoding the unique features of a given notation system may be designed and customized for a given set of users. The disadvantage, however, is that these systems are largely developed independent of each other, and may exhibit entirely incompatible ways of approaching notation encoding. This approach can be seen in many current encoding formats, where the choice to support the features of common music notation (CMN) in MusicXML, for example, creates a fundamental incompatibility with accurately capturing nuance in mensural notation. This is then addressed by developing entirely new encoding formats, such as the Computerized Mensural Music Editing (CMME) format[5], specifically built to handle the unique features of mensural music but ultimately incompatible with other formats without the creation of lossy translators. This creates a highly fragmented music notation ecosystem, where software developers must choose which types of notation they can support in their applications and which ones are specifically out of scope.

Earlier versions of the MEI schema focused on the second approach, initially built to represent CMN with all other systems declared out of scope. This led to a number of criticisms about its ability to accurately capture notation nuance; for example, Bradley and Vetch commented: "Although the scholarly orientation of the MEI markup scheme seemed extremely promising…considerable further work would be needed to extend it so that it could appropriately express these very subtle notational differences" [3].

Later revisions of the MEI schema added support for different types of music notation, but still it was criticized for being unable to capture particular nuances in highly specialized repertoires. A pointed criticism of the representation of neumed notation in MEI was given in [4], which makes entirely valid points about the ability of a generalized notation encoding system to capture highly specific details about a particular notation type.

There are inevitable commonalities between different systems of music notation, yet there are simply no universal commonalities across all systems of music notation. This suggests a possible third approach to the creation of music notation encoding schemes that has yet to be fully explored. This third approach exemplifies what we will call the "framework" approach, where parties interested in supporting new types of notation can leverage existing description methods for common aspects of music notation documents, yet are able to extend this to cover unique aspects of a given repertoire. This allows developers to focus specifically on the features that make that music notation system unique, while still leveraging a large body of existing research and development in common encoding tasks.

We call this the framework approach because it mirrors the use of software development frameworks, like Apple's Cocoa framework[6]. A framework provides a large number of "pre-packaged" methods designed to alleviate the burden of mundane and repetitive tasks, and allows application developers to focus on the features that make their application unique. It significantly reduces duplication of effort, and provides a platform that can easily be bug tested and re-used by many other people.

The MEI 2011 Schema marks the first release where extension and customization can be very easily applied to the core set of elements to produce custom encoding systems that extend support for new types of musical documents. This has been accomplished by adopting the tools and development processes pioneered by the TEI community and will be discussed further in the next section.

3. TEI AND MEI: TOOLS AND CUSTOMIZATION

The TEI was established to develop and maintain a set of standard practices and guidelines for encoding texts for the humanities. The scope of this project is extensive, but even with a comprehensive set of guidelines in place, there is a recognition that the guidelines developed by the core community do not cover all possible current or future use cases or applications for the TEI.

[5] http://www.cmme.org

[6] http://developer.apple.com/technologies/mac/cocoa.html

To address this, the TEI community has developed a process where custom TEI schemas may be generated through a formalized extension process. There is no single TEI schema or TEI standard [5]. The full set of TEI elements is arranged in 21 modules according to their utility in encoding certain features of a text (e.g., names and dates, drama, transcriptions of speech, and others). A customization definition file is then applied to the full set of elements specifying which features from these modules should be present in the output schema, and a custom schema for encoding a particular set of sources is generated by running the source and customization files through a processor.

The most powerful feature of this customization process is that new elements, attributes, or content models may be included in the customization definition, allowing the addition of new elements into the TEI that can address the differences presented by new types of documents. What this customization approach represents is the transition from a single, monolithic encoding schema to an extensible

Module Name	Module content
MEI	MEI infrastructure
Shared	Shared components
Header	Common metadata
CMN	Common music notation
Mensural	Mensural music notation
Neumes	Neume notation
Analysis	Analysis and interpretation
CMNOrnaments	CMN ornamentation
Corpus	Metadata for music corpora
Critapp	Critical apparatus
Edittrans	Scholarly editions and interpretations
Facsimile	Facsimile documents
Figtable	Figures and tables
Harmony	Harmonic analysis
Linkalign	Temporal linking and alignment
Lyrics	Lyrics
MIDI	MIDI-like structures
Namesdates	Names and dates
Performance	Recorded performances
Ptrref	Pointers and references
Tablature	Basic tablature
Text	Narrative textual content
Usersymbols	Graphics, shapes and symbols

Table 1: MEI core modules.

document-encoding framework. Validation schemas for ensuring conformance with TEI guidelines can be dynamically generated from a central source and shared among other community members interested in encoding similar document types.

3.1 MEI as an encoding framework

The MEI core is divided into 23 modules, each used to encapsulate unique characteristics of musical source encoding (**Table 1**). There are a total of 259 elements defined in the 2011 version of the MEI core, up from 238 in the 2010 release. The MEI core, like the TEI core, is expressed in an XML meta-schema language, the "One Document Does-it-all" (ODD) format. The ODD meta-schema language provides developers with the facility for easily capturing encoding rules, grouping similar functionality into re-useable classes, and providing a central place for documentation, following a literate programming style. We use the term "meta-schema," since it does not actually provide XML validation on its own, but provides MEI developers with the ability to express definitions of the MEI elements, the rules of how these elements may or may not be used, and their accompanying documentation. The Roma processor[7] can then be used to create validation schemas expressed in three popular schema languages: RelaxNG (RNG), W3C Schema (XSD), and Document Type Definition (DTD). (Of these three, RelaxNG is the preferred schema validation language for MEI). To generate these custom validation schemas, two ODD-encoded files are needed: the MEI core, containing all possible elements and maintained by the MEI Technical Group; and a customization file containing directives that specify the modules that should be activated in the resulting custom MEI schema. A complete set of HTML documentation may also be produced for a specific customization. This documentation includes usage guidelines for elements and their accompanying attributes, as well as automatically generated information about where a given element may or may not appear in a source tree. This process is illustrated in **Figure 1**.

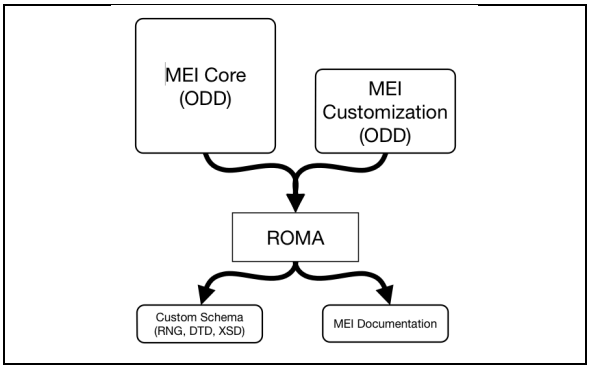

Figure 1: The MEI customization process.

The most powerful feature of this system is that the ODD modification file allows for the definition of new elements and the re-definition or removal of core elements in the resulting schema. This functionality gives schema developers the ability to define extensions to MEI,

[7] http://www.tei-c.org/Guidelines/Customization/use_roma.xml

customizing the core set of elements to accurately capture nuance and unique features of a given repertoire or set of documents. These customizations may be targeted at specifically addressing the needs of these documents, building on and extending the base set of MEI elements.

The customization functionality of MEI challenges the idea of building a common encoding system. The infinite and deep customization functionality available in the framework approach allows the development of incompatible "dialects" of MEI. Does this actually represent an advance in music document encoding over a more fragmented encoding landscape with separate encoding initiatives focused on specific areas? While the creation of incompatible document-encoding systems is a possibility, we believe that there are specific advantages to the MEI and TEI approach, based on three assumptions about the nature of document-encoding languages and their development.

The first assumption is that the developers of custom schemas want to address a perceived need for encoding a given musical document type, and typically do not want to reinvent entire document structures. Without a formal customization and extension process, however, developers of music encoding schemas have needed to construct entirely new encoding platforms from the ground up.

The second assumption is that there are fewer encoding system developers than there are potential users of a given encoding system. A single developer who needs to develop a method of accurately capturing a given document type—German lute tablature, for example—will take the time to learn the customization process, while most encoding projects will be largely satisfied by the capabilities in the MEI core or pre-made and distributed customizations. Once a customization has been completed, that work can then be made available for others to use and extend, reducing further duplication of effort.

Finally, the third assumption is that developing compatible encoding formats is a social and political process, as well as a technical one [6]. The TEI has addressed this by forming Special Interest Groups (SIGs) in which groups of individuals and organizations develop and propose extensions to the TEI core that deal with encoding specific types of documents, like correspondence and manuscripts. The fragmentation of an encoding language into incompatible dialects is not a technical problem, but one that can be addressed through discussion among stakeholders. The advantage that the customization approach brings to the process, however, is that it provides a common platform on which to base development and discussions. The customization tools allow a formalization of these discussions into a well-defined set of rules and guidelines.

These assumptions have yet to be extensively scrutinized and only time and further discussion will tell if they accurately reflect reality. In the next section we will discuss a new MEI community initiative to allow developers to share their MEI extensions among other interested parties in an open development process.

3.2 The MEI Incubator

The MEI Incubator was created to provide community members with a common space for developing and sharing their MEI extension customizations. Incubator projects are proposed by a Special Interest Group (SIG) from the community to address specific needs that members of the SIG feel are not adequately addressed in the MEI core. The Incubator website[8] hosts a common code repository and documentation wiki.

As Incubator projects mature, the SIG may then propose that the work of the SIG be incorporated into the MEI core as a new module, or an update to an existing module. An editorial committee will review the proposed extension for its suitability and ensure that the proposal does not duplicate existing functionality or create incompatibilities with existing MEI core modules.

The complexity of document encoding and the needs of communities to accurately describe sources may ultimately result in modifications that are fundamentally incompatible with the MEI core. While this means that it is unlikely that this extension will make it into the MEI core, the work done by the SIG can still be made available to others, making it possible to leverage a common platform to share existing work in specialized document encoding.

Incubator projects are designed to be a means through which community members can participate in MEI development and propose new means and methods for musical document encoding. In the next section, we will demonstrate this process by examining a current Incubator project and illustrate how ODD modifications may be used to extend the MEI core.

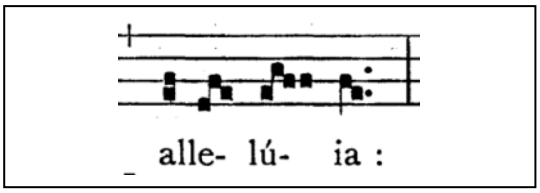

Figure 2: An example of the Solesmes neume notation showing a four-line staff, neumes, and divisions (vertical lines).

3.3 Sample Extension: The Solesmes Module

The monks at Solesmes, France, were responsible for creating a large number of liturgical service books for the Catholic Church in the late 19th and early 20th centuries. These books included missals, graduals and, perhaps most famously, the *Liber Usualis* [7], a book containing most of the chants for the daily offices and masses of the Catholic Church. These books were notated using a revival of 12th-century Notre Dame notation, featuring square note groups (neumes) on a four-line staff (**Figure 2**).

[8] http://code.google.com/p/mei-incubator

There are a number of features of this particular type of notation that make it different from other types of earlier notation. Although MEI has included functionality for encoding neume notation since 2007, it was ultimately found to be insufficient for accurately capturing Solesmes-style neume notation for a project dedicated to automatically transcribing the contents of these books. Certain features, like *divisions* (similar, but not equivalent to breath marks, graphically represented by a vertical line across the staff), *episema* (note stresses) and Solesmes-specific neume names and forms were not present in the existing MEI core.

A new Incubator project was proposed to address the need for an updated method of handling this type of neume notation. The ODD modification file created for this project defines four new elements for MEI, as well as their accompanying attributes. Due to space considerations we cannot reproduce the entire modification file, but we will illustrate the process by focusing on the method used to define the <division> element. We follow the convention of using angle brackets (< >) to identify XML elements, and the @ symbol to identify XML attributes.

```
<elementSpec ident="division"
    module="MEI.solesmes" mode="add">
  <desc>Encodes the presence of a division
        on a staff.</desc>
  <classes>
    <memberOf key="att.common"/>
    <memberOf key="att.facsimile"/>
    <memberOf key="att.solesmes.division" />
  </classes>
</elementSpec>
```

Figure 3: Declaration of the <division> element in ODD.

This <elementSpec> definition (**Figure 3**) creates a new element, <division>, with the name specified in the @ident attribute. The @module attribute specifies the MEI module to which this element belongs, and the @mode attribute specifies the mode the Roma processor should use for this element. The @mode attribute may be one of "add," for adding a new element, "delete," for removing an existing element from the resulting schema, or "replace," for re-defining an existing element (the "delete" and "replace" attribute use are not shown in **Figure 3**).

The <desc> tags provide the documentation string for this element. The Roma processor will use this information to create the HTML documentation for the resulting schema customization. The <classes> element specifies the classes this element belongs to. In this case, the <division> element will automatically inherit the XML attributes specified in the att.common, att.facsimile, and att.solesemes.division classes. Of these three classes, two are defined in the MEI core while the third is declared elsewhere in the Solesmes ODD file.

The <classSpec> declaration (**Figure 4**) creates a new class of attributes, att.solesmes.division. This class is used to define a new group of attributes that may be used on any element that is a member of this class; in this case, only the <division> element is a member of this class, but more general classes of attributes may be defined that apply to multiple XML elements (like the att.common class). The new @form attribute is declared by the <attDef> element. Additional attributes may be declared by creating more <attDef> children of the <attList> element. The @usage attribute on <attDef> declares this attribute to be optional, meaning that it is acceptable if a <division> element does not possess a @form attribute. Required attributes may be specified by setting this to "req."

```
<classSpec ident="att.solesmes.division"
   type="atts" mode="add">
  <desc>Divisions are breath and
        phrasing indicators.</desc>
  <attList>
    <attDef ident="form" usage="opt">
      <desc>Types of divisions.</desc>
      <valList type="closed">
        <valItem ident="comma" />
        <valItem ident="major" />
        <valItem ident="minor" />
        <valItem ident="small" />
        <valItem ident="final" />
      </valList>
    </attDef>
  </attList>
</classSpec>
```

Figure 4: Declaration of the att.solesmes.division class to describe a common attribute group.

The <valList> element defines the possible values that the @form attribute may have; in this case the only valid values for the @form attribute are given by the <valItem> elements. Since the value list here is a closed set, any values supplied in the @form attribute that is not one of those specified will not pass validation.

```
<division form="comma" />
Valid, @form can take comma as a value.
<division />
Valid, @form is optional.
<division form="bell" />
Invalid, @form must be one of the specified
values.
<division name="long" />
Invalid, @name is not allowed on this element.
```

Figure 5: Valid and invalid use of the <division> element defined in the Solesmes module.

These definitions will result in a schema that allows a <division> element in an MEI file, something that is not considered valid in unmodified MEI. **Figure 5** llustrates valid and non-valid examples of this in practice.

The full Solesmes module contains definitions for four new elements, <division>, <episema>, <neume>, and <nc> (neume component) and eight new attributes to accompany these elements. When this customization is processed with the Roma processor against the 2011 MEI core, a schema is produced that can be used to validate MEI instances.

4. MEI SOFTWARE LIBRARIES

For software developers looking to integrate MEI into their applications, we have developed two new software libraries to support reading and writing MEI files. Libmei is written in C++, and PyMEI is written in Python. Using object-oriented programming principles, these software libraries were designed to reflect the same modular structure as MEI, and are extensible by others to add support for new customizations. PyMEI 1.0 was developed as a rapid prototype for testing and designing a common API, which was then written in C++ as libmei. PyMEI 2.0, scheduled for release in Fall 2011, will adopt libmei as the base platform, unifying the two projects and serving as a reference implementation for the creation of MEI software libraries in other languages.

Architecturally, every element in the MEI core is mirrored in the software libraries by a corresponding class—the `<note>` element has a `Note` class, and so on. Every element class inherits from a base `MeiElement` class. This base class contains methods and attributes common to all MEI elements, like getting and setting names, values, child objects, and element attributes. Subclasses that inherit from this base class gain all of these functions. In the subclasses, however, are musical methods and attributes that are specific to the semantic function of that particular MEI element. For example, a `Note` class has get and set methods for pitch-related attributes, while a `Measure` class has methods for working with measure numbers.

To extend this software, developers can easily add new classes to reflect new elements that they have added to an MEI customization. For example, a developer who wishes to support the `<division>` element specified in the Solesmes module would only need to create a `Division` class that inherits from the base `MeiElement` class, and then implement any methods that he or she wants to support for this class. For example, a developer may wish to add explicit `getForm` and `setForm` methods to set the `@form` attribute on the `<division>` element. The libmei and PyMEI projects are available as open source projects on GitHub[9,10], licensed under the MIT license.

5. CONCLUSION

With the 2011 release of the MEI Schema and the adoption of tools developed by the TEI project, MEI has moved beyond a static music document schema to an extensible document-encoding framework, providing developers with a formalized method of customizing and extending MEI to meet specific needs. An extensive set of elements and guidelines for creating valid MEI documents forms the core of MEI, but the complexity of music makes it impossible to anticipate every context in which users may want to use it.

To help support and direct these efforts, we have created a new MEI community initiative, the MEI Incubator. This initiative will provide community members with a common space to "grow" their customizations and share them with other members of the community, reducing duplication of effort. As Incubator projects mature, they may be proposed as extensions to the MEI core, subject to editorial review, and finally adopted into the specification itself.

To support MEI in software applications, we are also releasing software libraries that assist developers with providing MEI import and export functionality. Currently we are targeting two common programming languages, C++ and Python, but we are also investigating support in other languages as well.

MEI goes beyond simple notation encoding. It is a powerful platform for creating, sharing, storing, and analysing music documents. We are investigating methods of integrating MEI into optical music recognition platforms, as well as searching, analysing, and displaying MEI-encoded document facsimiles in a digital environment.

6. ACKNOWLEDGEMENTS

The authors would like to thank Erin Mayhood for her support and encouragement, the MEI Technical Group for their valuable musical and technical insights, and our colleagues at the Distributed Digital Music Archives and Libraries Lab. This work was supported by the Social Sciences and Humanities Research Council of Canada, the National Endowment for the Humanities, and the Deutsche Forschungsgemeinschaft (DFG).

7. REFERENCES

[1] Jannidis, F. 2009. TEI in a crystal ball. *Lit. & Ling. Comp.* 24 (3): 253–65.

[2] Selfridge-Field, E. 1997. *Beyond MIDI: The handbook of musical codes*. Cambridge, MA: MIT Press.

[3] Bradley, J., and P. Vetch. 2007. Supporting annotation as a scholarly tool: Experiences from the Online Chopin Variorum Edition. *Lit. & Ling. Comp.* 22 (2): 225–41.

[4] Barton, L. 2008. KISS considered harmful in digitization of medieval chant manuscripts. In *Proc. Int. Conf. Automated Solutions for Cross Media Content and Multi-channel Distribution* at Florence, Italy. 195–203.

[5] Ide, N., and C. Sperberg-McQueen. 1995. The TEI: History, goals, and future. *Comp. and the Humanities* 29: 5–15.

[6] Bauman, S., and J. Flanders. 2004. Odd customizations. In *Proc. Extreme Markup Lang.*. Montréal, Quebec.

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

[9] http://github.com/ahankinson/pymei

[10] http://github.com/ddmal/libmei