A MusicXML Test Suite and a Discussion of Issues in MusicXML 2.0

Reinhold Kainhofer, http://reinhold.kainhofer.com, reinhold@kainhofer.com
Vienna University of Technology, Austria
and
GNU LilyPond, http://www.lilypond.org
and
Edition Kainhofer, http://www.edition-kainhofer.com, Austria

Abstract

MusicXML [Recordare LLC, 2010] has become one of the standard interchange formats for music data. While a "specification" in the form of some DTD files with comments for each element and equivalently in the form of XML Schemas is available, no representative archive of MusicXML unit test files has been available for testing purposes. Here, we present such an extensive suite of MusicXML unit tests [Kainhofer, 2009]. Although originally intended for regression-testing the musicxml2ly converter, it has turned into a general MusicXML test suite consisting of more than 120 MusicXML test files, each checking one particular aspect of the MusicXML specification.

During the creation of the test suite, several shortcomings in the MusicXML specification were detected and are discussed in the second part of this article. We also discuss the obstacles encountered when trying to convert MusicXML data files to the LilyPond [Nienhuys and et al., 2010] format.

Keywords

MusicXML, Test suite, Software development, Music notation

1 About the MusicXML test suite

MusicXML has become the de-facto exchange format for visual music data, supported by dozens of software applications. It has even been proposed as a tool for musicological analysis [Vigliante, 2007; Ganseman et al., 2008], online-music editing [Cunningham et al., 2006] or evaluating OMR systems [Szwoch, 2008] among others.

Recently, the first version of the Open Score Format specification [Yamaha Corporation, 2009] was published together with a PVG (Piano-Voice-Guitar) profile, which employs MusicXML as its data format. Thus most problems with MusicXML will automatically carry over to this new specification, too.

Despite a full syntactic definition of the MusicXML format [Recordare LLC, 2010], no official suite of representative MusicXML test files has been available for developers implementing MusicXML support in their applications. The only help were the comments and explanations given in the specification to create test cases manually. The predominant advice is to use the Dolet plugin for Finale, which is a proprietary Windows and MacOS application that is not easily available for many Open Source developers employing Linux. Also, this approach invariably will lead to MusicXML being interpreted as behaving like the Dolet plugin instead of being an application-independent specification. Furthermore, the lack of a test suite means that a lot of work is duplicated creating MusicXML test cases.

This lack of a complete MusicXML test suite for testing purposes was our main incentive for creating such a semi-official test suite [Kainhofer, 2009]. Due to the complexity of musical notation, a complete set of test cases, covering every possible combination of notation and all possible combinations of XML attributes and elements, is apparently out of reach. However, we attempted to create representative samples to catch as many common combinations as possible. Our main goal was to create small unit test cases, covering not only the most common features of MusicXML, but also some less used musical notation elements, like complex time signatures, instrument specific markup, microtones, etc.

The test suite together with sample renderings are available for download at its homepage: http://kainhofer.com/musicxml/

2 Structure of the test suite

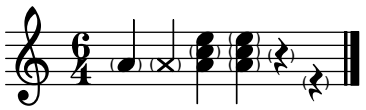
We identified twelve different feature categories, each dealing with separate aspects of the MusicXML specification (like basic musical notation, staff attributes, note-related elements, page layout, etc.). Each of these categories was further split into more specific aspects, for which we created several test cases each.

01-09 ... Basics
01 Pitches
02 Rests
03 Rhythm
10-19 ... Staff attributes
11 Time signatures
12 Clefs
13 Key signatures
20-29 ... Note-related elements
21 Chorded notes
22 Note settings, heads, etc.
23 Triplets, Tuplets
24 Grace notes
30-39 ... Notations, articul., spanners
31 Dynamics and other single symbols
32 Notations and Articulations
33 Spanners
40-44 ... Parts
41 Multiple parts (staves)
42 Multiple voices per staff
43 One part on multiple staves
45-49 ... Measure issues and repeats
45 Repeats
46 Barlines, Measures
50-54 ... Page-related issues
51 Header information
52 Page layout
55-59 ... Exact positioning of items
60-69 ... Vocal music
61 Lyrics
70-75 ... Instrument-specific notation
71 Guitar notation
72 Transposing instruments
73 Percussion
74 Figured bass
75 Other instrumental notation
80-89 ... MIDI and sound generation
90-99 ... Other aspects
90 Compressed MusicXML files
99 Compatibility with broken MusicXML

Table 1: Structure of the files, categorized by file name

The test suite currently consists of more than 120 test cases, where each file represents one particular aspect of the MusicXML format and is named accordingly. The file name starts with two digits, encoding the area of the test (see Table 1 for the exact meaning of the first two digits), followed by a letter to enumerate the test cases within each category. Finally, a short verbal description[1] of the test case is given in the file name. The file extension follows the standard of `.xml` for normal MusicXML files and `.mxl` for (ZIP-) compressed MusicXML archives as defined in `container.dtd`.

Every test case is supposed to test one particular feature or feature combination of MusicXML. If a feature has multiple possible attribute values or different uses within a score, the corresponding test file contains several subtests, separated as much as possible by using different notes or even different measures or staves for each of the values or combinations. For example, parenthesized notes or rests use the `parentheses` attribute of a `notehead` XML element. However, for an application it might make a difference if that note is a note on its own, a note with a non-standard note head or part of a chord. Similarly, parenthesized rests can have a default position in the staff or an explicit position given in the MusicXML file (using the `pitch` child element of the `note` describing the rest). The test case for parenthesizing covers all these cases:



This choice of combining closely related combinations or aspects of the same feature into one test case provides a nice balance between clearly separating test cases for different features to avoid influences of bugs in one feature on another feature, while still keeping the number of test files relatively low, which is relevant if running a test suite cannot be automated.

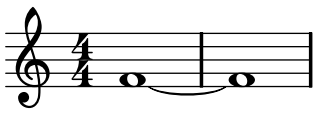
3 An example of a unit test

The unit test files are kept as simple as possible, so that they can best fulfill their purpose of checking only one particular aspect. For example, the unit test file `33b-Spanners-Tie.xml` to check the processing of simple ties – a trivial feature, which still needs to be tested in coverage and regression tests – reads:

```
<?xml version="1.0" encoding="ISO-8859-1"
    standalone="no"?>
<!DOCTYPE score-partwise PUBLIC
    "-//Recordare//DTD MusicXML 0.6b
    Partwise//EN"
"http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise>
  <identification>
    <miscellaneous>
      <miscellaneous-field
          name="description">Two simple tied
           whole notes</miscellaneous-field>
    </miscellaneous>
  </identification>
```

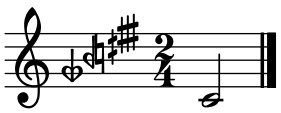
[1] A more detailed description is given in a description element inside the XML file.

```
  <part-list>
    <score-part id="P1"/>
  </part-list>
  <part id="P1">
    <measure number="1">
      <attributes>
        <divisions>1</divisions>
        <key><fifths>0</fifths></key>
        <time>
          <beats>4</beats>
          <beat-type>4</beat-type>
        </time>
        <staves>1</staves>
        <clef number="1">
          <sign>G</sign>
          <line>2</line>
        </clef>
      </attributes>
      <note>
        <pitch>
          <step>F</step>
          <octave>4</octave>
        </pitch>
        <duration>4</duration>
        <tie type="start"/>
        <voice>1</voice>
        <type>whole</type>
        <notations><tied
            type="start"/></notations>
      </note>
    </measure>
    <measure number="2">
      <note>
        <pitch>
          <step>F</step>
          <octave>4</octave>
        </pitch>
        <duration>4</duration>
        <tie type="stop"/>
        <voice>1</voice>
        <type>whole</type>
        <notations><tied
            type="stop"/></notations>
      </note>
    </measure>
  </part>
</score-partwise>
```



Other test files check for more exotic features, like for example the test case for non-traditional key signatures with microtone alterations (excerpt of `13d-KeySignatures-Microtones.xml`):

```
<attributes>
  <divisions>1</divisions>
  <key>
    <key-step>4</key-step>
    <key-alter>-1.5</key-alter>
    <key-step>6</key-step>
    <key-alter>-0.5</key-alter>
    <key-step>0</key-step>
    <key-alter>0</key-alter>
    <key-step>1</key-step>
    <key-alter>0.5</key-alter>
    <key-step>3</key-step>
    <key-alter>1.5</key-alter>
  </key>
[...]
</attributes>
```



However, in all cases the structure of the test file is kept as simple as possible to avoid cross-interactions of bugs in different areas of an application.

4 Sample renderings

Originally, the files of the test suite were generated as test cases for the implementation of musicxml2ly, a utility to convert MusicXML files to the LilyPond [Nienhuys and et al., 2010] format. Using this utility, sample renderings of all the test cases can be created automatically and are made available on the homepage of the test suite.

However, these renderings cannot be regarded as official reference renderings, either, since they represent only one particular interpretation (the one of the musicxml2ly converter), while the MusicXML specification leaves several aspects unclear, as detailed below. So several aspects of the semantic interpretation of MusicXML are left to the importing application. Furthermore, the musicxml2ly converter does not yet fully support every aspect of MusicXML. Rather, these sample renderings should be understood as an indication how one particular application understands the files.

5 Shortcomings of the MusicXML format

While generating the MusicXML test suite, we encountered several problems or inconsistencies with the MusicXML specification as given in the DTDs or XML Schemas. Those issues involve semantic ambiguities, suboptimal XML design choices and missing features.

5.1 Semantic ambiguities of MusicXML

First, while the MusicXML specification gives a precise syntactic specification of the format, the complexities of music notation inevitably lead to additional semantic restrictions, that can not properly be expressed in a DTD or an XML Schema. Additionally, MusicXML is also designed to cover the features of several commercially available music typesetting applications, where each application has implemented some

aspects differently. Unless the MusicXML specification clearly describes how certain combinations of attributes and/or elements are to be understood, there cannot be a unique interpretation of the exact musical or layout content of some MusicXML snippet.

Thus, the first type of issues we identified are semantic ambiguities, which might or rather should be clarified in the specification itself. Several of these were answered or explained by Michael Good in email threads on the MusicXML mailing list, but the official specification remains ambiguous to new developers.

5.1.1 Only a syntactical definition

The biggest problem with MusicXML is that it is a *purely syntactical definition*, while the musical content requires additional semantic restrictions. For example, spanners like tuplets, ties, crescendi, analysis brackets etc. are simply marked with a start, possibly some continuation and an end element. Several spanners can be arbitrarily overlapping, however, it is not possible to properly specify that each of these spanners must be closed (at a position where it makes sense musically). Furthermore, it does not make sense from a musical standpoint to have e.g. a crescendo and a decrescendo overlapping in the same voice. However, a part can have several different voices, each with different hairpins, so this is not a restriction for a part, only for a voice in the conventional sense. Adding such restrictions at the voice-level is simply not possible in a pure syntax specification like the DTD or XSD.

5.1.2 Voice-Basedness

Many music notation and sequencer applications are based on the concept of *voices*, which was also introduced in MusicXML through the `voice` element of `note`. Notes with the same voice element value are assumed to be in the same voice, but the voice element is not required (in the OSF PVG profile, a voice element is finally required). It is not clear, whether a missing voice element implicitly means voice one or whether notes without a voice element should be placed in a voice of their own. In any case, an importing application needs to check whether the imported assignment of notes to voices is possible in the application at all, and thus the voice element can only be used as a strong indication, but not as a definitive assignment to voices. In particular, many applications don't allow overlapping notes in the same voice. Such notes would need to be assigned to different voices upon import.

5.1.3 Attributes

Another unclear part of the MusicXML specification regards the *display of the settings* of the `attributes` element. Some applications export the time and key signature for every measure, even if it hasn't changed. The specification is quiet on whether the presence of an `attributes` element is supposed to imply the presence of a graphical indication of these settings (i.e. explicitly displaying the key or time signature) or only to assert specific values and leave the decision to implementations. In the latter case (which is apparently favored by Michael Good), the MusicXML file does not specify the layout uniquely and different applications will produce different output.

5.1.4 Chords

Chords are another unclear area in the specification: A sequence of notes with the `chord` element can only appear after a note that does not have the `chord` element set and serves as the base note for the chord. This restriction cannot be handled in a DTD, but was introduced in the PVG profile of OSF. Even worse, the grammar allows e.g. `forward` or `backward` elements before a note with `chord`, so that the note does no longer appear at the same time as the chord's base note. This additional restriction that no forwards or backward appear between the notes of a chord needs to be clarified. Also, theoretically the notes of a chord can belong to different voices in the file, while this is not supported by most applications.

5.1.5 Lyrics

Lyrics in MusicXML can have both a stanza number and a name attribute to distinguish different lyrics lines. But the specification is quiet whether the number, the name alone or the combination of number and name should be used to determine, which syllables belong together. Even worse, it seems that for the page display, the vertical position of the lyrics is the main factor to associate lyrics syllables into words. This is a violation of the otherwise rather strict separation of content and display.

5.1.6 Figured Bass

Figured bass numbers in MusicXML are always assigned to the "first regular note that follows" per specification, but it does not say if this is meant in XML order or in time-order. In particular, there might be a `forward` or `backward`

element immediately after the figured bass. In this case, the XML-order and time-wise order is clearly different. Also, the `slash` value of the `suffix` child element does not distinguish forward slashes and back ticks (usually through a "6" to indicate a diminished chord). The DTD only says "The orientation and display of the slash usually depends on the figure number." While forward and backward slashes might mean the same from a musical point of view, the display of the figure will thus vary from application to application. In most other cases, in contrast, MusicXML tries to be as precise possible as far as the display is concerned.

5.1.7 Harp Pedal Diagrams

Finally, the `harp-pedals` element for *harp pedal diagrams* lists the pedal states for the D, C, B, E, F, G, and A harp pedals, but only recommends to give the pedals in the usual order. For different orders, it is not clear whether the harp pedal should be displayed in the usual order or in the order given. Also, there is no way to indicate the usual vertical delimiter for other orders.

5.2 Sub-optimal XML design

A further type of issues with the MusicXML format concerns the general design of the XML format. As the MusicXML format is supposed to be backward compatible, these design choices cannot be undone. However, for completeness, we will still discuss how they could have been done better.

5.2.1 Strict Order of XML Child Nodes

The MusicXML DTDs specify that the children of a `note` element (and of several similar elements) have to be in a *fixed order*. In particular, the `duration`, the optional `voice` and the `type` elements have to be in this exact order, although one would intuitively place the duration (the length in time units) and the type of the note (the visual representation of the duration) together. Also, from a theoretical point of view, there is no need to force a fixed ordering, as all child elements simply specify properties of that note. The only reason for this fixed ordering is – according to Michael Good[2] – a technical one, namely that the DTD does not provide a proper way to allow arbitrary ordering of the XML child elements while at the same time adding restrictions on the number of times an element appears. With an XML Schema this would be possible to model, but the XSD for MusicXML still enforces a fixed ordering to provide backwards compatibility.

5.2.2 Names of XML Elements Containing Pitch Information

Another suboptimal design choice regards all elements that contain some kind of pitch information or alteration. Their names are chosen to include the name of the enclosing XML element, even though this can already be deduced from the enclosing context. For example, the alteration of a normal note uses the `alter` element inside a `pitch`, while the alteration of a chord root uses the `root-alter` element inside a `root` and the alteration of a chord note uses `degree-alter` inside `degree`. As all of them simply indicate an alteration of the enclosing element, it would be cleaner to use the same element name `alter` for all these uses and instead use the enclosing context, too. This would also make implementations cleaner, as they can base all pitch and alteration information on one common base class, using the same names for the children. This issue appears not only with the `alter` and `(key|tuning|root|degree)-alter` elements, but also with the `step` and `(key|tuning|root)-step` as well as with the `octave` and `(key|tuning)-octave` elements.

5.2.3 Metronome Marks and Non-Standard Key Signatures

Contrast this over-correctness in naming and ignoring the context in the XML tree to the `metronome` element. Tempo changes of the form "old value = new value" using the `metronome` element are defined in the DTD as

```
(beat-unit, beat-unit-dot*,
 beat-unit, beat-unit-dot*)
```

where the two beat units indicate the value before and after the tempo change. As a result of the optional dots, one cannot simply access the old and the new tempo unit separately by their child index of the `metronome` element, but has to iterate through the children sequentially, checking for existing dots. For most other purposes MusicXML tries to give each item with even the slightest different function its own name. Similarly, custom key signatures are defined as

```
((...|((key-step, key-alter)*)),
                          key-octave*)
```

[2] Mail message to the MusicXML mailing list on March 11, 2008

where steps and alterations alternate. This design has the additional problem that the (optional) octave definitions for each of the key elements are given after the step/alter pairs in left-to-right order. As pointed out by Tulgan[3], this would be better represented by enclosing each of the key element information (step, alter and optional octave) in a separate `key-element` child element.

5.2.4 Shortcomings of DTD and XSD Formats

In the original DTD for MusicXML, *enumerated element values* cannot be properly modeled, but have to use the #PCDATA type and explain the possible values in the comments. Thus, all enumerations are badly defined in the DTD, as only some possible values are mentioned, which are unaccessible to any syntax checker. In the XSD specification for MusicXML, the possible values are mostly properly specified using an enumeration, which however makes extensions of MusicXML to non-western music notation very hard (for example, displaying microtone accidentals used in Turkish music). Additionally, the possible values are not further explained (for example `sharp-sharp` vs. `double-sharp` for accidentals).

A similar problem are *attributes*, where the DTD allows a text value, which is then understood as an integer or a real number. These attributes are mainly fixed to integer values in the OSF PVG profile.

5.2.5 Staff-assigned Elements

Markup text in MusicXML is mostly *assigned to a whole staff*. In reality, however, a text might be inherently assigned to one particular note or voice. For example, when two instruments are combined as two voices on one staff, a markup "dolce" or "pizz." might apply only to one of the two instruments. Similar cases are instrument cue names given in a piano reduction, as can be seen for example in the Telemann MusicXML example provided by Recordare. Unfortunately, the `voice` child element of `direction` is optional and thus most MusicXML exporters ignore it, causing problems for voice-based applications.

5.3 Missing features in MusicXML 2.0

While MusicXML 2.0 already added several important features over MusicXML 1.1, like exact positioning and offsets, there are several settings for professional music typesetting, which cannot be encoded in MusicXML. For these issues, we give some suggestions for a possible inclusion in MusicXML 2.1 or 3.0.

5.3.1 Document-wide Header and Footer Lines

First, while MusicXML allows one to completely describe the page layout of the music sheet, there is no way to specify a *document-wide header or footer line*. The `credit` element allows to place arbitrary text on any page, but it refers to only one page (page 1 by default). Thus, page headers and footers need to be defined for each page separately. We propose to add values "all", "even" and "odd" to the data type of the page attribute for `credit` (currently xsd:positiveInteger):

```
<credit page="even">
  <credit-words default-x="955"
      default-y="20">Even
      footer</credit-words>
</credit>
```

5.3.2 No Information about Purpose of Credit Elements

Another problem with the `credits` elements is that they are simply text labels, but do not store its function (i.e. whether such an element gives the composer, poet, title, etc.). For a pure layout-based application this might suffice, but any application, that tries to extract metadata from the MusicXML file or that has a different handling for score titles and contributor names, needs to extract that information. We propose to add an enumerated `type` attribute to the `credit` element with possible values title, subtitle, composer, poet or lyricist, arranger, publisher, page number, header, footer, instrument, copyright, etc.

```
<credit type="composer">
  <credit-words
    default-x="1124" default-y="1387"
    justify="right">Composer</credit-words>
</credit>
```

5.3.3 System Delimiters

In orchestral scores the systems are typically separated with a *system delimiter* consisting of two adjacent thick slashes. This is currently also not possible to express in MusicXML. One possible solution would be to add it as a `system-separator` element inside the `defaults -> system-layout` block of a score.

[3]Mail message to the MusicXML mailing list on October 7, 2006.

```
<defaults>
 <system-layout>
   <system-separator>double-slash
       </system-separator>
 </system-layout>
</defaults>
```

5.3.4 Cadenzas

Finally, there is no proper way to *encode a cadenza* in MusicXML. While a measure can have an arbitrary number of beats in MusicXML, irrespective of the time signature, the information about where a cadenza starts cannot be represented. This will cause problems with applications and utilities that check MusicXML files for correctness, as they have no way to distinguish incorrectly encoded files from files with a cadenza.

6 Issues in MusicXML translation to the LilyPond

In [Good, 2002] Michael Good discusses issues that appear in the translation of MusicXML files from and to the MuseData, NIFF, MIDI and Finale file formats. In this section, we will discuss issues that appear in the conversion of the MusicXML format to the LilyPond file format.

6.1 Musical Content vs. Graphical Representation

As LilyPond [Nienhuys and et al., 2010] is a WYSIWYM (what you see is what you mean) application, its data files describe the musical contents of a score, rather than its graphical description. Thus a converter from MusicXML to LilyPond needs to extract the exact musical contents from a file. In MusicXML, several elements – most notably dynamic signs like p, f, crescendi etc. – are mainly tied to a position on the staff and in some cases its onset and end can only be deduced by the horizontal offset of the dynamic sign in the MusicXML file. In LilyPond, however, almost all notation elements are attached to a note or rest, so these elements need to be quantized and correctly assigned to a note or rest in LilyPond. This is often not easily possible, due to explicit offsets on the page, effectively assigning the element to a different position than the position in the MusicXML file.

6.2 Staff-Assigned Items

Another problem in the conversion to LilyPond is caused by the same fact that in MusicXML many notation elements like dynamic markings or text markup is assigned to a staff, while in LilyPond they must be assigned to a particular note. If a staff contains two instruments (i.e. two voices, one for each instrument), one has to determine to which note to assign an encountered dynamic marking or text markup. In most case assigning it to the nearest note will produce the desired output, but there are many cases where it is not possible to determine whether a marking belongs to both instruments or just to one of them. As an example, a dynamic sign like "p" or "ff" might either apply to both instruments at the same time, or (e.g. if one of them has a short solo) only to one of them. Similarly, each instrument cue name provided as a help for the conductor in a piano reduction apply only to one particular voice in the (multi-voice) piano part, while other text markups will apply to all voices simultaneously.

If one is only interested in generating the exact layout as provided in the MusicXML file, a misassignment will still lead to correct visual output, although the extracted music information is not entirely correct. However, if one also wants to create separate instrumental parts for the two instruments in the first case mentioned above, then it is crucial to correctly assign each staff-assigned element to either one or both instrumental voices. On the other hand, assigning an element to both voices will then lead to duplicated items in the LilyPond output.

While it is true that MusicXML defined the `direction` element to optionally contain a `voice` element for that exact purpose, in reality most GUI applications for music notation will not cater for this functionality and thus produce MusicXML files without proper voice-assignment.

6.3 Voice-Based vs. Measure-Based

A further problem is that LilyPond is voice-based, where the measure length is determined by the current time signature, while MusicXML is measure-based, where a measure can contain an arbitrary number of beats, irrespective of the time signature. In LilyPond, voices are independently split into measures during processing rather than in the input. Only later are those measures synchronized. As a consequence, if one voice has more beats in a measure than another voice, LilyPond will not be able to properly synchronize them, so some skip elements need to be inserted to line the voices up. The

voice-basedness of LilyPond also causes problems with MusicXML's optional voice element, which allows a voice to have overlapping notes, which is not allowed in LilyPond, so that a MusicXML voice will possibly need to be split into multiple voices.

6.4 Page Layout

Concerning the *page layout*, LilyPond needs metadata about the score title and the contributors and will create its own labels on the title page and the header and footer bars. As discussed above, the `credit` element does not contain that information, so that the title page from the MusicXML file cannot be reproduced. Even worse, page headers and other markup is placed at absolute positions in the MusicXML score, which is not possible in LilyPond.

6.5 Workarounds in MusicXML Files

Finally, even some sample files provided by Recordare on the MusicXML homepage mix the graphical display with the musical information. For example, in the `Chant.xml` sample file a divisio minima (a short tick through the topmost staff line) is not encoded as a barline with the proper `tick` bar-style attribute, but as a `words` direction element with text "|", which is then shifted manually so that it appears at the correct position. Such hacks cannot work with any application that tries to extract the musical contents and not the exact page layout.

7 Conclusion

The MusicXML test suite presented in this article finally provides software developers in the area of music notation with an extensive set of representative test cases to check conformance to the MusicXML specification and to perform regression and coverage tests.

Even though MusicXML has established itself as an industry standard for exchanging music notation, it is still encumbered with several minor issues. Our discussion in the second part of the paper attempts to provide implementors of MusicXML import and export features with some hints about possible pitfalls and ambiguities in the format.

Nonetheless, MusicXML is a very useful file format for the extremely hard and complex task of music notation exchange. As the OSF specification has already shown, one can expect that future versions of MusicXML will clarify, solve or at least soften most of the issues we discuss here.

References

Stuart Cunningham, Nicole Gebert, Rich Picking, and Vic Grout. 2006. Web-based music notation editing. In *Proc. IADIS Int. Conf. on WWW/Internet 2006*. Murcia, Spain, October 5-8, 2006.

Joachim Ganseman, Paul Scheunders, and Wim D'haes. 2008. Using XQuery on MusicXML databases for musicological analysis. In *Proc. ISMIR 2008*, pages 427–432. 9th Int. Conf. on Music Information Retrieval. Philadelpha, September 14-18, 2008.

Michael Good. 2002. MusicXML in practice: Issues in translation and analysis. In *Proc. First Int. Conf. MAX 2002: Musical Application Using XML*, pages 47–54. Milan, September 19-20, 2002.

Michael Good. 2006. Lessons from the adoption of MusicXML as an interchange standard. In *Proc. XML 2006*.

Reinhold Kainhofer. 2009. Unofficial MusicXML test suite. `http://kainhofer.com/musicxml/`. Representative set of MusicXML test cases.

Han-Wen Nienhuys and Jan Nieuwenhuizen et al. 2010. GNU LilyPond. `http://www.lilypond.org/`. The music typesetter of the GNU project.

Recordare LLC. 2010. MusicXML 2.0. Document Type Definition (DTD): `http://musicxml.org/dtds`, W3C XML Schema Definition (XSD): `http://musicxml.org/xsd`.

Mariusz Szwoch. 2008. Using MusicXML to evaluate accuracy of OMR systems. In *Diagrammatic Representation and Inference: Proc. Diagrams 2008*, volume 5223 of *Lecture Notes in Computer Science*, pages 419–422. Springer Verlag, Berlin. Herrsching, Germany, September 19-21, 2008.

Raffaele Vigliante. 2007. MusicXML: An XML based approach to automatic musicological analysis. In *Proc. Digital Humanities 2007*. Urbana-Champaign, IL, June 4-8, 2007. Urbana-Champaign, IL, June 4-8, 2007.

Yamaha Corporation. 2009. Open Score Format (OSF, ver. 1.0), packaging specification. `http://openscoreformat.sf.net/`.