8 MusicXML for Notation and Analysis

Michael Good

Recordare
PO Box 3459
Los Altos, CA 94024
good@recordare.com
www.musicxml.org/xml.html

Abstract

MusicXML is intended to represent common western musical notation from the seventeenth century onwards, including both classical and popular music. MusicXML is intended to support interchange between musical notation, performance, analysis, and retrieval applications. It is designed to be sufficient, not optimal, for these applications.

COMPUTING IN MUSICOLOGY 12 (1999-2000), 113-124.

MusicXML is an XML-based music-interchange language.[1] It is intended to represent common western musical notation from the seventeenth century onwards, including both classical and popular music. The language is designed to be extensible to future coverage of early music and less standard notational needs of twentieth and twenty-first century scores. (Non-western musical notations would probably be best represented through a separate XML language.)

[1] An abstract of this paper was presented as a poster at the International Symposium on Music Information Retrieval MusicIR (October 2000).

MusicXML is intended to support interchange between musical notation, performance, analysis, and retrieval applications. It is therefore designed to be sufficient, not optimal, for these applications. MusicXML is not intended to supersede other formats that are optimized for specific musical applications, but to support sharing of musical data between applications. The development goal is to support interchange with any musical program for western notation with a published computer data format.

The current MusicXML converter software runs on *Windows*. As of October 2000, it reads from:

- *MuseData* format
- *Finale*'s Enigma Transportable File (ETF) format (Coda 1998)
- NIFF format

The current MusicXML software writes to:

- *MuseData* format
- Standard MIDI Files in Format 1 (MIDI 1997)
- The *Sibelius* 1.3 and *Finale* 2001 notation applications

MusicXML software currently provides complete coverage for both reading and writing *MuseData* files, and partial coverage of the other formats and applications. The NIFF, ETF, and MIDI converters use XML versions of these formats as intermediate data structures.

MusicXML adapts the *MuseData* and *Humdrum* formats to XML, adding features needed to cover more of music usage from the mid-nineteenth century to the present time.[2] These were chosen as starting points because they are two of the most powerful languages currently available for musical analysis and interchange. One of *Humdrum*'s important features is its explicitly two-dimensional representation of music by part and by time. A hierarchical representation like XML cannot directly support this

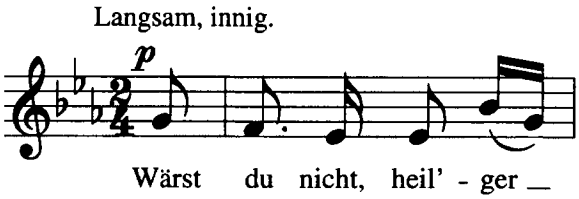
[2] I wish to thank Eleanor Selfridge-Field, Walter B. Hewlett, Barry Vercoe, and David Huron for their advice, encouragement, and prior work in musical score representation.

type of lattice structure, but automatic conversion between these two orderings is an adequate alternative. MusicXML uses Extensible Style Sheet Transformations (XSLT) to convert between two hierarchical representations: a *part-wise* score where measures are nested within parts, and a *time-wise* score where parts are nested within measures.

8.1 A Sample MusicXML Encoding

To give a flavor of MusicXML, here is an encoding of the beginning of the voice part of Robert Schumann's Op. 35 setting of Kerner's "Frage," illustrated in Figure 1.

Figure 1. Robert Schumann, start of the setting of Kerner's "Frage" and its representation in MusicXML.




```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC
    "-//Recordare//DTD MusicXML Partwise//EN"
    "http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise>
  <part-list>
    <score-part id="P1">
      <part-name>Voice</part-name>
    </score-part>
  </part-list>
  <part id="P1">
    <measure number="0" implicit="yes">
      <attributes>
        <divisions>4</divisions>
        <key>
          <fifths>-3</fifths>
          <mode>major</mode>
        </key>
        <time>
          <beats>2</beats>
          <beat-type>4</beat-type>
        </time>
        <clef>
          <sign>G</sign>
          <line>2</line>
        </clef>
```

```
      <directive>Langsam, innig.</directive>
    </attributes>
    <note>
      <pitch>
        <step>G</step>
        <octave>4</octave>
      </pitch>
      <duration>2</duration>
      <type>eighth</type>
      <stem>up</stem>
      <notations>
          <dynamics>
              <p/>
          </dynamics>
      </notations>
      <lyric>
          <syllabic>single</syllabic>
          <text>W&auml;rst</text>
      </lyric>
    </note>
  </measure>
  <measure number="1">
    <note>
      <pitch>
        <step>F</step>
        <octave>4</octave>
      </pitch>
      <duration>3</duration>
      <type>eighth</type>
      <dot/>
      <stem>up</stem>
      <lyric>
          <syllabic>single</syllabic>
          <text>du</text>
      </lyric>
    </note>
    <note>
      <pitch>
        <step>E</step>
        <alter>-1</alter>
        <octave>4</octave>
      </pitch>
      <duration>1</duration>
      <type>16th</type>
      <stem>up</stem>
      <lyric>
          <syllabic>single</syllabic>
          <text>nicht,</text>
      </lyric>
    </note>
    <note>
      <pitch>
        <step>E</step>
        <alter>-1</alter>
        <octave>4</octave>
      </pitch>
```

```
        <duration>2</duration>
        <type>eighth</type>
        <stem>up</stem>
        <lyric>
            <syllabic>begin</syllabic>
            <text>heil</text>
        </lyric>
      </note>
      <note>
        <pitch>
          <step>B</step>
          <alter>-1</alter>
          <octave>4</octave>
        </pitch>
        <duration>1</duration>
        <type>16th</type>
        <stem>up</stem>
        <beam number="1">begin</beam>
        <beam number="2">begin</beam>
        <notations>
          <slur type="start" number="1"/>
        </notations>
        <lyric>
            <syllabic>end</syllabic>
            <text>ger</text>
            <extend/>
        </lyric>
      </note>
      <note>
        <pitch>
          <step>G</step>
          <octave>4</octave>
        </pitch>
        <duration>1</duration>
        <type>16th</type>
        <stem>up</stem>
        <beam number="1">end</beam>
        <beam number="2">end</beam>
        <notations>
          <slur type="stop" number="1"/>
        </notations>
        <lyric>
            <extend/>
        </lyric>
      </note>
    </measure>
  </part>
</score-partwise>
```

MusicXML score files do not represent presentation concepts such as pages and systems. The details of formatting will change based on different paper and display sizes. In the XML environment, formatting is handled separately from structure and semantics. The same applies for

detailed interpretive performance information. Separate XML languages could be developed to represent individual printings and performances.

Each MusicXML score file represents a single movement. Multi-movement works and collections are represented in a MusicXML opus file, based on a separate DTD for linking and musicological data.

MusicXML documents are larger than previous text formats such as *MuseData* and *Humdrum*. However, XML documents compress well, and zip compression typically reduces the size of MusicXML files by a factor of 30. MusicXML files that are seven times larger than *MuseData* files when uncompressed are only twice as large when compressed.

8.2 MusicXML Score DTD Examples

The MusicXML score DTD is still being refined as it is tested with more music, music formats, and musical applications. The complete DTD can be accessed via *www.musicxml.org/xml.html*. Some examples from the current version illustrate both the level of detail in MusicXML and some of the standardization issues that arise when defining an XML interchange language.

In Figure 2 we see how a note element is defined in the current MusicXML score DTD. XML internal entities, such as full-note and voice-track in the following example, are equivalent to macros in other languages. The note definition is based on the layout of note records within *MuseData*.

Figure 2. Definition of a note element in a MusicXML DTD.

```
<!-- Internal entities to simplify note definitions -->
<!ENTITY % full-note "(chord?, (pitch | unpitched | rest))">
<!ENTITY % voice-track "(footnote?, level?, track?)">
<!-- Definition of the note element -->
<!ELEMENT note ((((cue | grace), %full-note;) |
               (%full-note;, duration, tie?, tie?)),
               instrument?, %voice-track;, type?, dot*,
               accidental?, time-modification?, stem?
               notehead?, staff?, beam*, notations*, lyric*)>
```

Two elements within the note definition are `pitch` and `tie`. Each element of a pitch definition is defined to contain parsed character data (PCDATA). The `tie`, however, is an empty element with a required type attribute that indicates whether this is the beginning or end of the tie (Figure 3).

Figure 3. Elements within the note definition.

```
<!ELEMENT pitch (step, alter?, octave)>
<!ELEMENT step (#PCDATA)>
<!ELEMENT alter (#PCDATA)>
<!ELEMENT octave (#PCDATA)>

<!-- Tie is an empty element with one attribute. -->
<!ENTITY % start-stop "(start | stop)">
<!ELEMENT tie EMPTY>
<!ATTLIST tie type %start-stop; #REQUIRED>
```

These definitions illustrate an interesting dichotomy in XML DTDs: `element` text is weakly typed, but `attribute` text can be strongly typed. Yet for most purposes, it is overall better design practice to put semantic data into elements rather than attributes (Harold 1999). Elements are generally easier to manipulate than attributes from within an XML program, and elements can have more complex structure than attributes can.

The weak typing of element text helps make XML DTDs more extensible for new applications, but puts a heavier burden on documentation and software to handle interoperability. In our current MusicXML software, pitch names and note types are interpreted using American terminology (C, not *do* or *ut*; an `eighth` note, not a *quaver* or *crochet*). But this is due to the software. Comments in the MusicXML DTD note this current restriction, but nothing in the MusicXML DTD can enforce it automatically. This particular restriction may be removed in future iterations of MusicXML, but in general, dialect issues can arise within a DTD based on the actual content of the XML elements. One of the benefits of XML schemas is to make stronger typing available throughout an XML language definition, not just at the attribute level. This does not, however, eliminate the design tradeoff between extensibility benefits and dialect drawbacks.

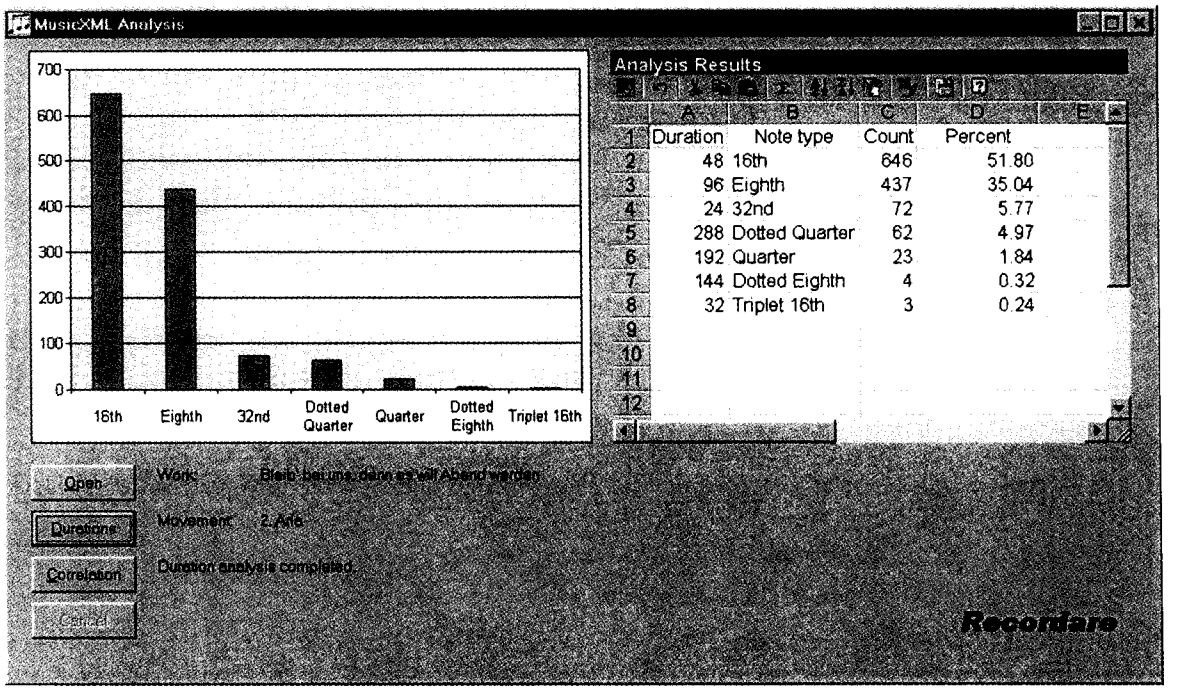
8.3 MusicXML Analysis Examples

One limitation to computer-based musical analysis has been the tight coupling of representations to development tools. *Humdrum* tools require familiarity with Unix usage, while *MuseData* tools run in TenX, a non-standard DOS environment. In contrast, XML programming tools are available for all major-industry programming languages and platforms. This lets the user rather than the representation language choose the programming environment, making for simpler development of musical applications.

Two main programming models are currently available for handling XML data: the Document Object Model (DOM) and Simple API for XML (SAX) (Martin et al. 2000). The W3C's DOM interprets an entire XML document as a tree of nodes. SAX (for serial access) provides an alternative event-based model, where an entire XML document need not be read into memory at once, but can instead be parsed on an as-needed basis. Tools for both models are available for many programming languages (e.g. Java, C++, *Visual Basic*) from many vendors. These examples use a DOM-based model coded using Microsoft *Visual Basic*. Both of the analysis program examples are adapted from the problem list on the *Humdrum* Web site.

Say we want to investigate whether Bach's pieces really have 90% of their notes in one of two durations (e.g. quarters and eighths, or eighths and sixteenths). We can do this by plotting a distribution of note durations on a bar chart, displayed together with a simple spreadsheet. Figure 4 shows the duration distribution for the second movement of Bach's Cantata No. 6 (BWV 6). The two most prevalent note durations make up nearly 87% of the notes. This is not quite the 90% posed in the question, but still a more uneven distribution than often seen. For retrieval purposes, an extended program could then look for the works in a given corpus with the most uneven distribution of note durations.

Figure 4. Duration distribution for Bach's Cantata No. 6 (BWV 6), second movement.



Note durations are represented as fractions in many musical codes, including *MuseData* and NIFF. MusicXML follows *MuseData*'s example in encoding the denominator (which changes rarely) in a separate element from the numerator, which is coded individually for each note. Thus to build the distribution chart, we need to search not only for `<note>` elements, but also for the `<divisions>` elements that change the duration denominator, represented as fractions of a quarter note. An earlier piece of code searches through the document for all the `<divisions>` used, and computes a value called `nDivisions` that is an even multiple of all these different divisions.

Once this is done, we search the file for both `<note>` and `<divisions>` elements using *XPath*, a W3C recommendation for addressing parts of an XML document (Clark 1999). We then update the counters in an array that contains all possible duration values within the file. Cue and grace notes are excluded (Figure 5). Following this code, we assign the data to our charting tools of choice (in this case, Microsoft's *Office Web Components*).

Figure 5. Xpath addressing of notes and divisions.

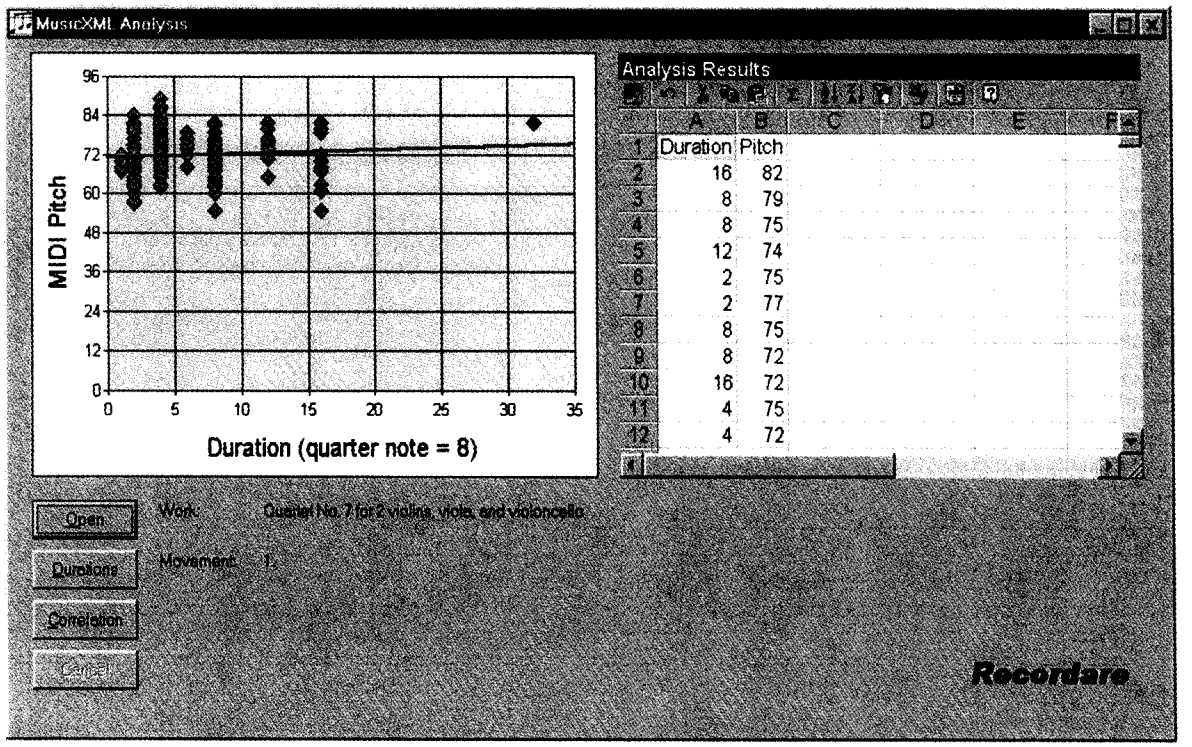
```
Set oNodes = oRoot.selectNodes("//note | //divisions") ' XPath
Do
  Set oNote = oNodes.NextNode
  If oNote Is Nothing Then Exit Do
  If oNote.tagName = "divisions" Then
    nMultiplier = nDivisions \ CLng(oNote.Text)
  Else
    Set oTmp = oNote.selectSingleNode("duration")
    If Not oTmp Is Nothing Then ' Exclude grace and cue notes
      nDuration = CLng(oTmp.Text) * nMultiplier
      nCounts(nDuration) = nCounts(nDuration) + 1
    End If
  End If
Loop
```

As another example, say we wanted to investigate whether there is a correlation between pitch and duration in a given score. The code logic is nearly the same. Instead of computing the counts in an array, we instead add pitch/duration pairs for each note to the spreadsheet. Rests are excluded, along with cue and grace notes (Figure 6).

Figure 6. Xpath or a coordinate search of pitch and duration.

```
' Code structure is same as for distribution analysis.
' The Else clause for handling notes changes:
Else
    Set oDuration = oNote.SelectSingleNode("duration")
    Set oPitch = oNote.SelectSingleNode("pitch")
    If (Not oDuration Is Nothing) And _
       (Not oPitch Is Nothing) Then   No rests either
      nDuration = CLng(oDuration.Text) * nMultiplier
      nPitch = MidiNote(oPitch)      ' Separate function
      oWorksheet.Cells(i,1) = nDuration
      oWorksheet.Cells(i,2) = nPitch
    End If
End If
```

Figure 7. Pitch/duration scatter-plot for Mozart's Quartet No. 7 (K. 169), first movement.



Afterwards, we map the scatter-plot axes to the two columns in the spreadsheet to display the graph. Figure 7 shows a scatter-plot of pitch vs. duration for the first movement of Mozart's String Quartet No. 7 (K. 169). As with most musical scores we have looked at so far, there is no correlation between the two.

8.4 Conclusions

Music information retrieval faces a tower-of-Babel problem. There is no musical format in widespread use today that overcomes MIDI's limitations as an interchange format between performance, notation, analysis, and retrieval applications. A problem for past interchange efforts has been the absence of commonly used formats for complex structured data in general. XML provides the technical foundation for an interchange format that is more powerful and expressive than the current MIDI

format. Developing converters between existing formats and a single MusicXML document type definition has the potential to greatly simplify the task of music information retrieval. MusicXML attempts to provide a common document type definition that is well designed from musical, human, and computer perspectives.

References and URLs

Bray, Tim, Jean Paoli, and Charles M. Sperberg-McQueen (eds.) (2000). *Extensible Markup Language (XML)* 1.0 (Second Edition). World Wide Web Consortium (W3C) Recommendation, October 6, 2000. *www.w3.org/TR/2000/REC-xml-20001006*

Clark, James (ed.) (1999). *XSL Transformations (XSLT)* Version 1.0. W3C Recommendation, November 16, 1999. *www.w3.org/TR/1999/REC-xslt-19991116*

Harold, Elliotte Rusty (1999). *XML Bible.* (Foster City, CA: IDG Books Worldwide).

Hewlett, Walter B. (1997). "*MuseData*: Multipurpose Representation" in *Beyond MIDI: The Handbook of Musical Codes*, ed. Eleanor Selfridge-Field (Cambridge, MA: The MIT Press), 402–447. *www.ccarh.org/publications/books/beyondmidi/online/musedata/*

Martin, Didier et al. (2000). *Professional XML.* (Birmingham, UK: Wrox Press).

Huron, David (1997). "*Humdrum* and *Kern*: Selective Feature Encoding" in *Beyond MIDI: The Handbook of Musical Codes*, ed. Eleanor Selfridge-Field (Cambridge, MA: The MIT Press), 375–401. *dactyl.som.ohio-state.edu/Humdrum/*

MusicIR (2000). *International Symposium on Music Information Retrieval* (Plymouth, MA; Oct. 23-25, 2000). U. Mass. Center for Intelligent Information Retrieval in conjunction with the Digital Libraries Phase II and the National Science Foundation. Abstracts at: *ciir.cs.umass.edu/music2000*

MusicXML (2000) *www.musicxml.org/xml.html*