6 Extensible Markup Language (XML) for Music Applications: An Introduction

Gerd Castan, Michael Good, and Perry Roland[1]

Abstract

[1] Individual addresses are found in the succeeding articles.

Extensible Markup Language (XML), a subset of the Standard Generalized Markup Language (SGML; ISO standard 8879 of1986), is designed to make it easy to interchange structured documents over the Internet. Musical scores are considered here as structured documents.

This brief introduction describes XML's general character and strengths, as well as its suitability for representing notated music as a structured document.

Many initiatives to use XML for the interchange of musical data are now underway. Specific implementations (for notation, analysis, and cataloguing of musical sources) are described in three succeeding articles, one by each of the authors.

COMPUTING IN MUSICOLOGY 12 (1999-2000), 95–102.

6.1 What is XML?

XML is a system-neutral Internet format for representing structured data. It is a subset of Standard Generalized Markup Language (SGML), which was developed in the Seventies for the interchange of text files between printers. The primary motivation behind XML is to simplify and improve delivery of information via the Internet. The first version of XML was designed by the World Wide Web Consortium (W3C) in 1998.

XML has several secondary objectives. It defines how Internet Uniform Resource Locators [URLs] can be used to identify component parts of XML data streams. It removes inconsistencies and ambiguities of earlier markup languages. Unlike SGML and HyperText Markup Language (HTML), it forces users to adhere to strict data-models and provides syntactical error-checking.

In contrast to HTML, XML emphasizes elements of content, as opposed to typographical style, layout, or form. Like SGML and HTML, XML involves extensive tagging, but the focus of the tagging is to identify logical types of information (in music such types might include beams, bars, and rests). Although default tags are available in XML, they are often impractical for document exchange on the Internet.

Of particular benefit to music applications is the fact that XML allows support for multiple descriptions of the same kinds of data on a continuum of "completeness." The task of formatting documents is relegated to other software adapted to read XML data descriptions ("schemas"), formally called Document Type Definitions (DTDs).

6.2 XML and Previous Interchange Initiatives

A great deal of effort has been expended on encoding musical data for notation, analysis, and query. More than 20 music codes are described in detail in *Beyond MIDI: The Handbook of Musical Codes* (Selfridge-Field, 1997); several dozen others exist. More than 1,000 complete works of classical music have been symbolically encoded for non-commercial use. For the full potential of music representation to be realized and music-information-retrieval procedures to be implemented, a common interchange standard is essential.

Existing methods of interchanging musical data range from formal schemes designed over many years but implemented only slightly to *de facto* procedures adapted to short-term needs. At the formal end of the range, Standard Music Description Language (SMDL) used principles of SGML to build a tagging system for elements of musical notation, musical performance, and "the logical core of the work," in which many elements might duplicate those of the other musical phenomena but a few might be unique. Although the formal description was adopted by the ISO in 1996, implementations of SMDL have been few.

The NIFF file format (NIFF 1996; Grande 1997) was designed to allow the interchange of music-notation data between music-notation editing and publishing programs and music-scanning programs. Its intended range of applications ran from full music-publishing systems through simpler music-display systems to symbolic codes. Its original model was the *SCORE* parametric representation (Smith 1997). Although NIFF specifications were designed with industry support, neither of the original sponsors (Passport, Coda) implemented the format in commercial products.

At the informal end, MIDI is effectively the one interchange format in widespread use today. While sufficient for many performance applications, MIDI does not represent many concepts needed by musical applications concerned with printing and data retrieval. Its inadequacies reflect its origins as a hardware interface. It is inexact about enharmonic notation and too literal about performed note values, such that rhythmic values may deviate from the norms of visual grammar in images.

A frequent proposal has been to extend an existing code for use in the interchange of musical data between applications. In essence, MIDI has been that code. Its shortcomings for applications outside sound illustrate the pitfalls that could arise with any code developed for one purpose and "borrowed" for another. Musical information is enormously feature-rich. Codes work for a specific application when they represent all the variables relevant to that application. When the goal is changed, the variables relevant to the new application must magically be available.

Many existing music representations are inappropriate for interchange due to their narrow focus on one objective. They may be useful only as input codes (Selfridge-Field 1997: 571). Some would argue, conversely, that other representations, including SMDL, have attempted to represent music too broadly. SMDL has been unable to attract a large user group in part because it is difficult for potential users and tool-developers to see how SMDL might apply to their particular application. A standard that is defined generally enough to represent all music can only be made to work for a particular subset with great effort.

While XML seems to provide musicians with a representation scheme that holds the potential to eclipse both MIDI and formal interchange codes, such as NIFF and SMDL, as an interchange format, early experiments with XML have identified two areas in which some difficulties may lurk:

- the distinction between "style" (SGML, SMDL) and "content" (XML) is not absolutely clear in every situation.
- the construction of hierarchies of musical information within XML schemas may make certain assumptions about processing order that are arbitrary and will not work with all programs.

These procedural issues are both known from the world of music representation and data interchange prior to the advent of XML (see Hall 1997).

6.3 Design Principles of XML

XML is in principle transparent. It is written in generic ASCII code. Its ability to handle global commerce (requiring the use of many alphabets and special symbol sets) is strengthened by its provisions for handling Unicode, a standard international-character encoding scheme.

A lot of attention was devoted in the early stages of XML's evolution to tidying up inconsistencies and closing loopholes in previous markup languages. Attention has also been given to reconciling XML with modern hardware and operating systems; SGML was written for the mainframe era and still bears some evidences of these origins.

6.3.1 Syntax: Elements and Attributes

The syntax of XML is predefined and relies on two basic categories of information—elements and attributes—used in earlier markup languages. An *element* in XML is a basic category of information. Elements in a description of music might include notes, chords, and so forth. An element must have a name. It may contain attributes, dependent sub-elements (still called "elements"), and textual content.

An element might read

```
<note name= "c" octave= "1"/>
```

The simplicity of this statement illustrates the principle of immediate comprehensibility: we would have some idea of what the statement above meant even if we had no formal knowledge of XML. As in the syntax of HTML and other markup models, elements are enclosed in less-than (<) and greater-than (>) signs. In the above construction "`note name`" is called the *tag* or *element name.*

Next in order come the attributes of the element. Each attribute has a *name* (here `octave`), followed by the equalsign (=) and a value (here `"1"`).

If an element has no "children", it ends with a forward slash (/) before the closing sign (>). Note that the indicator meaning "no children" is part of the syntax.[2] Thus, no knowledge of the data-model is necessary to know if a concrete element has children or not.

[2] We use the word *child* to describe secondary elements.

XML syntax has a hierarchical structure that fits well with basic concepts in musical notation. For example,

```
<chord>
   <note name= "c" octave= "1" />
   <note name= "e" octave= "1" />
   <note name= "g" octave= "1" />
</chord>
```

In this case we have a chord with three notes as *children*. As in all parallel constructions, the statement without a slash (`<chord>`) is called a *start tag.* The statement with a slash (`</chord>`) is an *end tag.*

In hierarchies, start tags and end tags must always be nested. For example, the sequence

```
<measure>
   <chord>
      <note .../>
</measure>
```

would not be allowed, because a `</chord>` tag is missing before the `</measure>` tag.

6.3.2 Semantics

Beyond the syntax, we can define that in a valid music notation document any chord element must have one or more note elements as children. We can also define the attributes that may or must occur in a note element and we can define the values that an attribute may accept.

When we do that, we are talking about an application of XML. We now see that XML is really a meta-format that defines the syntax. It has implications for the way in which we define the semantics. The syntax is strictly separated from the semantics that we define ourselves.

6.3.3 Definition of the Semantics

A Document Type Definition (DTD) in the intial version of XML dealt with the expression of the semantics. It served as a contract between the XML writing and the reading software. Anything that can be expressed with a DTD can be expressed with what is now called an XML *schema.* Eventually all DTDs may be replaced by schemas, but applications based on DTDs, which are upwardly compatible with schemas, are in no danger of being made obsolete by schemas.

6.4 SML for Conventional Music Notation

6.4.1 Expressing Hierarchies

Hierarchies are as fundamental to XML syntax as they are to musical notation. These are examples of notational concepts to which the organizational concept of hierarchies seems to have an obvious application:

- systems within a page
- staves within a system
- measures within a staff
- chords within a measure
- notes within a chord

There are sometimes reasons not to express all of these relations as hierarchies.

6.4.2 Grouping by Common Values

Hierarchies can be viewed from top to bottom or bottom to top. For example, we could replace the chord-contains-notes hierarchy with a sequence of notes with a common chord attribute:

```
<note chord= "chord1" ... />
<note chord= "chord1" ... />
<note chord= "chord2" ... />
<note chord= "chord2" ... />
```

Here it is indicated that the first two notes belong together because they share a common value ("chord1"). This way of grouping is seen in several binary and XML-based formats. While it is usual for such groups to share common values, there is no single place where they can be stored.

It is generally preferable to store data in one single, well-defined place. Storing data with a common function in different places is costly and error-prone.

6.4.3 Grouping by Common References

To almost every rule in music notation there is an exception (e.g. beams can cross bar lines, melodic lines can cross staves, line-breaks can occur in the middle of measures, and so forth). Such occurrences make the use of hierarchies inadvisable. XML provides accommodation for hierarchical exceptions through links.

How do we represent objects (such as beams) which contain other objects and various parameters determined by the other objects when we cannot employ hierarchies analogous to the "measure-contains-notes" one? If we grouped the notes with the *common value* procedure, we would have no single place to store such information as the gradient of the beam. One solution offered by XML is to make the beam an element with a unique identifier (id), e.g.

```
<beam id= "beam1" .../>
```

XML provides a mechanism that assures that this identifier occurs only once in a document. Then we use notes as follows:

```
<note beamRef= "beam1" .../>
```

6.5 XML as a Comprehensive Data Model

XML's ability to represent both hierarchies and references spanning hierarchies is attractive. XML is also attractive because standard tools (parsers and editors with a parser inside) can be used to validate models; no new code may need necessarily to be created. The document-type or schema description lies outside the program. As the following articles demonstrate, multiple DTDs and schemas for representing music are already in use.

Among its other strengths, XML is easy to learn because it has the same "look and feel" as HTML, the current formatting language of World Wide Web documents. While other markup languages and formats claim to be easy to learn, XML is designed to facilitate rapid development of related tools and to be accessible in any modern programming language. Free XML syntax-checking tools are available to check the validity of a document without knowing what it means.

References

DuCharme, Bob (1999). *XML: The Annotated Specification.* Upper Saddle River, NJ: Prentice Hall PTR.

Graham, Ian S., and Liam Quinn (1999). *XML Specification Guide.* New York, NY: John Wiley & Sons.

Grande, Cindy (1997). "The *Notation Interchange File Format*: A *Windows*-Compliant Approach" in *Beyond MIDI: The Handbook of Musical Codes*, ed. E. Selfridge-Field (Cambridge, MA: MIT Press), 491–512.

Hall, Tom (1997). "DARMS: The A-R Dialect" in *Beyond MIDI: The Handbook of Musical Codes*, ed. E. Selfridge-Field (Cambridge, MA: MIT Press), 193–200.

NIFF 6a (1996): *neume.sourceforge.net/niff/* [also in HTML format; 1998].

Selfridge-Field, Eleanor, ed. (1997). *Beyond MIDI: The Handbook of Musical Codes.* Cambridge, MA: MIT Press.

SGML (1986). International Standards Organization (ISO) document 8879.

Smith, Leland (1997). "*SCORE*" in *Beyond MIDI: The Handbook of Musical Codes*, ed. E. Selfridge-Field (Cambridge, MA: MIT Press), 252–280.

Unicode: *www.unicode.org*