STRUML: AN XML BASED FILE FORMAT FOR THE MARK-UP OF SIMPLE SHEET MUSIC

by
Kevin Doupe

A Master's paper submitted to the faculty
of the School of Information and Library Science
of the University of North Carolina at Chapel Hill
in partial fulfillment of the requirements
for the degree of Masters of Science in
Information Science

Chapel Hill, North Carolina

April 2001

Approved by:

Advisor

Kevin Doupe. StruML: An XML based file format for the mark-up of simple sheet music. A Master’s paper for the M.S. in I.S. degree. April, 2001. 28 pages.
Advisor: Gregory B. Newby

This project concerns the creation of an electronic music book based upon the eXtensible Mark-up Language (XML 1.0). It begins with a brief overview of the literature on the subject of music mark-up, and XML's relation to previous efforts in this area. There is a sizeable history of research in the electronic representation of music. This literature review is not comprehensive on all aspects of the subject. Rather it is concentrated upon the textual representation of electronically stored music, and how XML became the preferred file format. The project then demonstrates the steps taken by the author to create an XML file format that could be validated against a DTD or XML Schema, and an acceptable XSLT Stylesheet to display the song. The project will then close with a summary of possible directions for future development/research.

Headings:

Music – Computer network resources

Music -- Data processing

Musical notation – Computer programs

Electronic data processing – Music

XML – (Document markup language)

Table of Contents

Introduction – Review of Literature

The electronic manipulation of music has long been a goal of computer scientists as well as musicians. These communities did not limit their goal to the representation of music upon a printed page. They also wanted to describe live and recorded music. They held a belief that it was possible to capture the thought processes behind the encoding and decoding of traditional music notation within a computer program. This necessitated capturing the concepts of time and duration of a music note and not just a note’s finger position upon an instrument. They believed that the "musical score…constitutes a complete system of graphic signs and, properly represented for computer input, may be analyzed as a logical image of the unfolding musical events which make up the composition" (Forte, p.3). Until the early 1990's programmer's efforts in this area were held back by the limitations of computer software and hardware. They awaited the development of software programs that could represent music in a human readable format. The memory and processor speed necessary for the successful rendering of computer read music were not widely available.

Initial attempts at mapping music notation into a computer readable format centered upon converting music notation into alphanumeric codes. These efforts had a jump start from the literature exploring German Lute tablatures from the middle ages. "Lute tablature is composed of two main pieces of information: an alphanumeric character that indicates a finger location on the lute …[and] the minimal rhythmic value or duration of a vertical" (Charnasse, p.155). Lute tablature thus dealt with the note played and made an attempt at describing the time element of music. But a major problem with the lute tablature was the degree of variation from one transcriber to another. "If we want to design a system that is general enough to handle … variety it is essential to work on a standard representation of a tablature that uses a continuous value system for each note or rhythmic value instead of tablature characters" (Charnasse, p.155). The lute tablature was not quite a 50% solution. It had converted finger positions to alphanumeric codes, made an attempt at dealing with the time duration of a note, but could not coherently deal with the variety of interpretations by the transcribers. Programmers would have to look elsewhere for a useful computer representation.

In 1967 an MIT music researcher, Alan Forte, published a program that he declared the first of its kind. He devised a program, which read, "scores encoded in an input language isomorphic to music notation" (Forte, p.iii). He described the basis of his program as "syntactic (in the sense that parsing operations are performed on formal structures in the output string), many extensions and refinements can be made without excessive difficulty" (Forte, p.iii). He stated that his basis for beginning this project was the belief that within traditional musical notation, were rules that could be "interpreted as algorithms, stated in programming languages…[that could] yield a complete and precise structuring of the data represented by the score" (Forte, p.3). Forte wanted to capture in computer code the minute judgments and determinations that go into turning performed music into written music, and written music into performed music. Forte moved from music written in traditional notation, to an alphanumeric representation of each note or chord. This alphanumeric representation was then converted into computer code. With the development of XML, and the increase in memory and processor speed of average computers, Forte's intermediate step, the alphanumeric description of a music note or chord, could become the element tag of a language.

Implicit in all of these early projects was the quest for a system that would be, if not standard, then at least reproducible from one writer to another. The necessary program would also have to be extensible to a variety of applications, and that would be readable by humans as well as computers. In short, all of these developments were waiting for the development of a language with the characteristics of SGML or XML.

The late 1980s and early 1990s brought forth the first software that was powerful enough to process music notation directly into a computer readable form without "the cumbersome substrate of alphanumeric mappings" (Huron, p.12). SGML and later XML were adapted for this purpose.

There were several concerns with the development of a music programming language that did not rely upon alphanumeric coding. Dividing music into uniquely identifiable parts was difficult. Providing plain language tags to accompany traditional notation symbols was not always possible. "There are notational icons which do not have unique linguistic names" (Huron, p.17). One solution was to make new words. But that would risk even further diversity in naming and tagging conventions than already existed. The traditional icons had different names in different languages and sometimes even within the same language. American and Oxford English differ over several musical terms (eighth note and quaver for example). If researchers agreed upon a strict English, Latin or Italian based names as an international standard but they would lose a functional requirement that the tags be easy to reverse engineer by programmers of various mother languages. "It is better that the mnemonic relationship should be easier to recall in the decoding than in the encoding process itself" (Huron, p.24). Keyboard symbols were seen as a partial solution (& as treble clef, # as sharp, b as flat, o as whole note, | as bar line). But they were incomplete. Not all musical symbols can be represented in an intuitive manner using only keystrokes.

Another issue became apparent. Music notation had developed over several hundred years. It was a highly compact and efficient way to describe multiple simultaneous events. Loops, refrains, verses, melody, bass, rhythm, all were carried on a traditional piece of sheet music. This notation could not be rewritten from scratch in a matter of weeks or even years for computer usage. To a trained human eye, musical notation was readable in a nonverbal way. The solution might not be to code music for computers to read, but to build computers and develop software that could read music as it is already written.

The first attempts at digitizing music scores on the web dealt with GIF or JPEG formatted images, loaded onto a web page. They had low resolution, took up a lot of memory, and were not readable by machine. The images were flat. Information could not be removed from them other than by reading them as one would a printed piece of sheet music. With the appearance of Java and XML in the mid 1990s programmers started producing XML and JAVA based approaches. Many of these initial approaches are no longer available on the web. The Connection Factory in Holland developed and released in March 1998 an XML based program MusicML for the markup and description of music. The same group has also developed a Java applet for viewing MusicML scores within web pages. Unfortunately, their products are difficult to find and no longer supported by the new owners www.x-hive.com.

There were several other Java and XML related developments that are no longer found:

JComposer August 1997, from Alu O'Neal Java Laboratories http://www.jars.com/classes/jresout.cgi?resource=2135 "JComposer is a complete WYSIWYG Java application to create, edit, web-publish and print music notation. Evolved from the 1.0.2 based Java Music Viewer." Only this reference remains. The DTD and examples are no longer on the web.

XML became the focus of computer based music coding. A complete list of music initiatives involving XML can be found on: http://xml.coverpages.org/xmlMusic.html. Products/developments covered include:

- SMDL - Standard Music Description Language. (1991) Highly theoretical, for music theorists, conforms to the SGML standard.
- NIFF - Notation Interchange File Format (1995) For the display of SMDL documents.
- MNML - Music Notation Markup Language (1996)
- FlowML - A Format for Virtual Orchestras (April 2000)
- MusiXML (April 2000)

- MuTaTeD - Music Tagging Type Definition (April 2000)
- Music XML (November 2000)
- 4ML - Music and Lyrics Mark-Up Language (March 2001)
- MusicML http://www.tcf.nl/3.0/musicml/index.html
- JscoreML http://nide.snow.utoronto.ca/music February 2001
- HyTime - Hypermedia/Time-Based Structuring Language

The ultimate goal of all of these diverse projects is to develop a way of describing and marking up music. XML is the language of choice. XML's biggest advantages are multifold. It is open source. The DTDs and schemas, which define an XML document, can be published on the Web. The Extensible Style Sheet Transform language (XSLT), which has emerged as the most powerful (but complicated) method of displaying XML documents, can transform a single XML file into a limitless number of displays and formats.

The proliferation of competing flavors is XML’s greatest disadvantage. XSLT can counter this weakness by transforming one flavor of a music oriented XML into another. The various flavors, as long as they are conformant with the XML 1.0 specification, can be mutually recognizable. The XSLT’s second purpose is to render XML documents for display by transforming the XML into HTML.

The combination of XML with XSLT fulfills a long-standing goal of theorists. Huron stated, "the unspoken objective of computer based musical activities is the creation of a broad representational network… for the invention of new types of representations which may be integrated into the overall schemes…to provide a favorable environment for the discovery of new goals" (Huron, p.15). XML and

XSLT meet this objective. A single DTD or schema can be applied towards a number of songs. Each song, can in turn, be represented in a limitless number of ways: sound file, Braille, sheet music, with subsets of each. XSLT sheets can be written to display only lyrics, lyrics and melody, lyrics, melody, and chords, backing instruments etc… The potential is unbounded. The original goal of defining a flexible, yet standard representation of music, displayed in a limitless number of variations, has been met.

XML also provides a structured way of marking up data that greatly improves the capability of searching files. "Music research is almost a textbook application area for database technology. There is a part of the world in which the music researcher is interested which comprises musical objects, such as musical compositions, performances, instruments, sounds and so on. The researcher will wish to represent and manipulate information about that world. Relevant facts may be represented as data and different music research tasks may necessitate the viewing and manipulation of the same data, but from different perspectives" (Eaglestone, p.42). We could mark-up the song files instrument-by-instrument, track-by-track. It would be possible to retrieve only the desired aspects of the song. Those aspects could then be transformed into either an audio file or a visual format.

XML, in conjunction with XSLT, and XML Schema will allow all of the theorist's requirements. A single agreed upon, standardized, XML Schema or DTD could be used to validate a limitless number of XML files. Each of those files can be displayed in a limitless number of ways through the use of XSLT Stylesheets. Mapping tools exist that will allow users to write their XML tags from any human language and validate them to a standard central schema. The programs will be readable by humans and computer. They will be reversible. They will allow the manipulation, presentation, of music in a way that is at once standardized, and user defined.

Methodology

Project title: StruML

Project Goal:

To develop an electronic music book based upon an open source XML application.

Step 1: Discovery

The first step was the performance of an Internet search in order to find the state of the art in regard to XML applications for marking up music. This period took approximately 15 hours spread over the course of 5 days. We discovered several high-end groups developing music mark up for orchestral music. These groups were located in Europe, primarily in Italy, Holland, and Scotland. The high-end applications are geared towards the mark-up and notation of orchestral scores. Given the number of orchestras and symphonies in Europe, and the difficulty of transporting duplicate sheet music for each member of an orchestra, an electronic version could be highly beneficial. An electronic coding and display of music would ease the troubles associated with moving orchestral music from city to city. Many of these scores are of extreme importance as artifacts in their own right. Electronic transfer would reduce the chance of a catastrophic loss of a score with its hand written notes and mark-ups from the original composer or previous conductors.

Step 2: Decide upon an area of concentration

I limited my efforts to the low-end, mark-up and display of simple chord and lyric versions of songs. Other features can be added as the major browsers improve their support of XML, as my skill level increases, and as the W3C approves methods and languages supporting the display of XML. Planned additions include the display of tablature notation for guitars, melodies for voice, and notation for piano, bass, etc... The software would remain open source, based upon XML, with a public schema. Our data would not be proprietary, but the device and display will.

Step 3: Verify XML's suitability for project

Metadata. We chose XML because of its ability to mark-up not just the music and lyrics, but all of the copyright data, legal data, attributions, etc... that would ordinarily go on the title page of a songbook, or on the liner notes of a CD or album.

We want our database of songs to be searchable not just its title or performer, but by what album or CD it appeared on. We also wanted to display legal restrictions and copyright data as well. Other attributes included were genre and year. Ultimately we want a user to be able to ask for all of the "southern rock" songs written between 1974 and 1977. Ultimately we want to use the same basic ideas and schemas to complete audio transformations of XML documents.

Difficulties

The continuous state of flux that XML has been our chief adversary. Some of the corresponding issues include:

- DTD vs. Schema
- Attributes vs. child elements

Display:

- Apply CSS directly to XML
- Transform XML into HTML by way of XSLT, then apply a style sheet to the HTML output.
- The role of Java or JavaScript
- The development of XPath, XLink and XPointer
- Browser support by Netscape and Explorer for XML

Most of the XML books that we came across at the start of the project in August 2000 had been entirely overcome developments since their publication. Even the books published in 1999 or the beginning of 2000 were behind the state of the art by the time we began the project. This situation had corrected itself by late Winter 2001.

The largest initial difficulty was deciding between W3C Schema or DTD. XML needs some kind of declaration defining its elements, attributes, and their relationships with one another. Initially, the DTD was the preferred way. But there were two problems with DTDs.

- They were not written in valid XML and therefore could not be parsed.
- They did not support Name Spaces, which would make it harder to reuse similar named elements, and to distinguish between individual song instances.

Namespaces emerged in 1998 and 1999 as a way to make each and every XML document unique. They were not supported by the limited syntax of DTDs. The preferred solution was the development of the XML Schema Language. Schemas have several advantages over DTDs:

- They are written in valid, well-formed XML.
- They can be read by an XML parser.
- They support namespaces.

Along with the shift away from DTDs towards schemas, was the shift away from an element-attribute relationship of data representation towards child element - parent element relationship representation. What used to be thought of as an attribute describing an element was now seen as a child element of its parent element. This cleaned up the hierarchy of the DTDs and schemas, but added ambiguity. By the end of the semester the consensus gathered from various websites and web discussion groups was that DTDs and attributes were relics of the past and needed to be phased out quickly. Schemas made up of elements and name spaces would replace element – attribute oriented DTDs.

Facilitators

Until Fall 2000 I found very little documentation detailing what the XML Schema Language would look like or how it was formatted as opposed to a DTD. My difficulties decreased with the publication in November 2000 of Elizabeth Castro's XML for the World Wide Web, from Peach Pit Press. This was the first book that I'd come across that coherently and succinctly communicated the differences between DTDs and XML Schema Language. The book described the process that takes place as an XML file is:

- Read by a parser,
- Validated against both the W3C XML 1.0 specification and its own DTD or Schema,

- Transformed by an XSLT program into HTML or another flavor of XML,
- Displayed by applying CSS to the resulting HTML.
- Apply CSS directly to the XML file for a more limited display

The main limitation in all of these approaches is the limited support for XML in Internet Explorer 5.x and the near total lack of support for XML in both Netscape Navigator 6.0 and 4.7.

Castro's book also dealt with the possibility of XML enabling languages and concepts such as XPointer and Xlink. These are the XML methods for hyper linking and moving around within an XML document. They are also possible ways to display images. Unfortunately, neither Netscape nor Internet Explorer supports these methods. We cannot use them to display images or links in XML. It is still necessary to convert the files to HTML via XSLT for display. It is also possible to embed HTML into an XML file and use CSS to display the image.

At time of this writing, I am still uncertain of which method to concentrate on.

Options:

- XML >> CSS >> Display
- XML >> XSLT >> HTML >> CSS >> Display
- XML >> RDBMS >> Display

Future Developments

Work on the display issues will continue. Our current method is to have one schema control many separate XML documents each of which contain a single song. Ideally, a single XSLT document will transform those many XML documents. The resulting HTML will be displayed after being acted upon by a single CSS style sheet.

We are not certain if this is possible. Each song, due to its individual nature, might require unique transformation and display documents. We are currently trying to mimic the traditional display of sheet music. This is more difficult to control with XML than I initially understood. Sheet music currently presents a tremendous amount of information in a single black and white page. Songs are not generally displayed in the linear manner that computers do best. A song with different numbers of chords in each bar, or different number of lines in the verse and chorus, presents difficulties. If you look at a piece of sheet music in a songbook, or even a hymnal, several issues will become apparent. Some songs will have four verses, two lines each, each verse followed by a two-line chorus. Other songs might have four verses, two lines each, but the second line is identical in each verse. This is easy to display in print but not to dynamically produce. Writing a single XSLT or CSS document that will display the infinite varieties of even simple chord and lyric songbooks might not be possible. Each song might require it’s own XSLT or at least it’s own CSS document for proper display. But the basic idea of a single schema controlling and defining many XML instances will remain.

XML provides the strength and flexibility to define and organize a sub-language around a domain of interest such as music. XML’s enabling languages, W3C Schema, and XSLT, allow the possibility that disparate languages based around a common subject matter may be rendered mutually intelligible.

Appendix A: Sample StruML Files

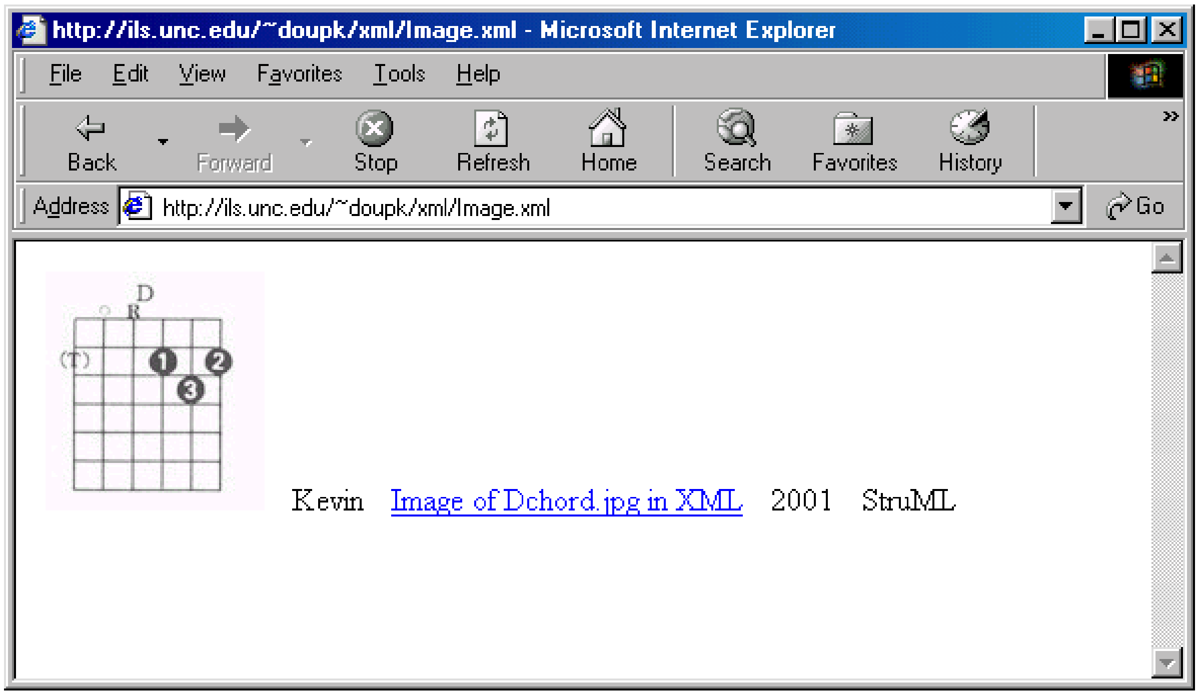
Figure A1: XML file to display an image:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="Image.css"?>
<StruML xmlns:html="http://ils.unc.edu/~doupk/xml/struml.xsd">
   <song>
      <chord>
         <html:img src="D_chord.jpg"/>
      </chord>
      <author>Kevin</author>
      <title xml:link="simple" show="replace"
       href="http://ils.unc.edu/~doupk/xml">Image of Dchord.jpg in
       XML</title>
      <pubyear>2001</pubyear>
      <publisher>StruML</publisher>
   </song>
</StruML>
```

Figure A2: Corresponding CSS Style Sheet:

```
StruML {display:table; }
song {display:table-row; }
song *{display:table-cell; padding:5px;}
title {color:blue; text-decoration:underline;}
```

Figure A3: Resulting Screen shot:


http://ils.unc.edu/~doupk/xml/Image.xml - Microsoft Internet Explorer
File
Edit
View
Favorites
Tools
Help
Back
Forward
Stop
Refresh
Home
Search
Favorites
History
Address
http://ils.unc.edu/~doupk/xml/Image.xml
Go
D
R
(T)

Kevin Image of Dchord.jpg in XML 2001 StruML

Figure A4: Sample XML Encoded Song (empty tags) based on StruML Schema

```
<?xml version="1.0" encoding="UTF-8" ?>
  <struml xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-
     instance"
     xsi:noNamespaceSchemaLocation="http://ils.unc.edu~doupk/xml/stru
     ml.xsd">
     <legal>
        <title />
        <written_by />
        <performed_by />
        <copyright />
     </legal>
     <song>
        <genre />
        <instructions />
        <intro>
           <chord />
           <chord />
           <chord />
           <chord />
        </intro>
        <verse>
           <chord />
           <chord />
           <lyrics />
           <chord />
           <chord />
           <lyrics />
        </verse>
        <chorus>
           <chord />
           <chord />
           <lyrics />
        </chorus>
        <outro>
           <chord />
           <chord />
           <chord />
           <chord />
        </outro>
     </song>
  </struml>
```

Figure A5: Sample StruML Schema

```
<?xml version="1.0" encoding="UTF-8" ?>
  - <!--
  Generated by XML Authority. Conforms to w3c
  http://www.w3.org/2000/10/XMLSchema
-->
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
<xsd:element name="struml">
   <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="legal" />
        <xsd:element ref="song" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="legal">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="title" />
        <xsd:element ref="written_by" maxOccurs="unbounded" />
        <xsd:element ref="performed_by" maxOccurs="unbounded" />
        <xsd:element ref="copyright" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="song">
    <xsd:complexType>
      <xsd:sequence maxOccurs="unbounded">
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element ref="chord" />
        </xsd:sequence>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element ref="lyrics" />
        </xsd:sequence>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element ref="genre" />
        </xsd:sequence>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element ref="instructions" />
        </xsd:sequence>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element ref="intro" />
        </xsd:sequence>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element ref="verse" />
        </xsd:sequence>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element ref="bridge" />
        </xsd:sequence>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element ref="chorus" />
        </xsd:sequence>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element ref="coda" />
        </xsd:sequence>
```

```
     <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="outro" />
      </xsd:sequence>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="title" type="xsd:string" />
<xsd:element name="written_by" type="xsd:string" />
<xsd:element name="performed_by" type="xsd:string" />
<xsd:element name="copyright" type="xsd:string" />
<xsd:element name="chord" type="xsd:string" />
<xsd:element name="lyrics" type="xsd:string" />
<xsd:element name="genre" type="xsd:string" />
<xsd:element name="instructions">
  <xsd:complexType mixed="true">
     <xsd:sequence maxOccurs="unbounded">
      <xsd:element ref="chord" minOccurs="0"
        maxOccurs="unbounded" />
      <xsd:element ref="lyrics" minOccurs="0"
        maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="intro">
  <xsd:complexType>
     <xsd:sequence maxOccurs="unbounded">
      <xsd:element ref="chord" minOccurs="0"
        maxOccurs="unbounded" />
      <xsd:element ref="lyrics" minOccurs="0"
        maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
 <xsd:element name="verse">
   <xsd:complexType>
     <xsd:sequence maxOccurs="unbounded">
      <xsd:element ref="chord" minOccurs="0"
        maxOccurs="unbounded" />
      <xsd:element ref="lyrics" minOccurs="0"
        maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
 <xsd:element name="bridge">
   <xsd:complexType>
     <xsd:sequence minOccurs="0" maxOccurs="unbounded">
      <xsd:element ref="chord" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
 <xsd:element name="chorus">
   <xsd:complexType>
     <xsd:sequence maxOccurs="unbounded">
      <xsd:element ref="chord" minOccurs="0"
        maxOccurs="unbounded" />
      <xsd:element ref="lyrics" minOccurs="0"
        maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
   <xsd:element name="coda">
     <xsd:complexType>
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="chord" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="outro">
     <xsd:complexType>
       <xsd:sequence maxOccurs="unbounded">
        <xsd:element ref="chord" minOccurs="0"
          maxOccurs="unbounded" />
        <xsd:element ref="lyrics" minOccurs="0"
          maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Figure A6: Sample StruML File (empty tags) based on StruML DTD

```
<?xml version="1.0" encoding="UTF-8" ?>
  <!DOCTYPE struml (View Source for full doctype...)>
   <struml>
     <legal>
      <title e-dtype="string" />
      <written_by e-dtype="string" />
      <performed_by e-dtype="string" />
      <copyright e-dtype="string" />
    </legal>
     <song>
      <genre e-dtype="string" />
      <instructions />
       <intro>
        <chord e-dtype="string" />
        <chord e-dtype="string" />
        <chord e-dtype="string" />
        <chord e-dtype="string" />
      </intro>
      <verse>
        <chord e-dtype="string" />
        <chord e-dtype="string" />
        <lyrics e-dtype="string" />
        <chord e-dtype="string" />
        <chord e-dtype="string" />
        <lyrics e-dtype="string" />
        <chord e-dtype="string" />
        <chord e-dtype="string" />
        <lyrics e-dtype="string" />
        <chord e-dtype="string" />
        <chord e-dtype="string" />
        <lyrics e-dtype="string" />
      </verse>
     <chorus>
        <chord e-dtype="string" />
        <chord e-dtype="string" />
        <lyrics e-dtype="string" />
        <chord e-dtype="string" />
        <chord e-dtype="string" />
        <lyrics e-dtype="string" />
      </chorus>
       <verse>
        <chord e-dtype="string" />
        <chord e-dtype="string" />
        <lyrics e-dtype="string" />
        <chord e-dtype="string" />
        <chord e-dtype="string" />
        <lyrics e-dtype="string" />
        <chord e-dtype="string" />
        <chord e-dtype="string" />
        <lyrics e-dtype="string" />
        <chord e-dtype="string" />
        <chord e-dtype="string" />
        <lyrics e-dtype="string" />
    </verse>
    <outro>
      <chord e-dtype="string" />
      <chord e-dtype="string" />
      <chord e-dtype="string" />
      <chord e-dtype="string" />
    </outro>
  </song>
</struml>
```

Figure A7: StruML DTD

```
<?xml version='1.0' encoding='UTF-8' ?>
<!--Generated by XML Authority-->

<!ELEMENT struml (legal , song)>

<!ELEMENT legal (title , written_by+ , performed_by+ , copyright+)>

<!ELEMENT song ((genre)* , (instructions)* , (intro)* , (verse)* ,
 (bridge)* , (chorus)* , (coda)* , (outro)*)+>

<!ELEMENT title (#PCDATA)>

<!ATTLIST title  e-dtype NMTOKEN  #FIXED 'string' >
<!ELEMENT written_by (#PCDATA)>

<!ATTLIST written_by  e-dtype NMTOKEN  #FIXED 'string' >
<!ELEMENT performed_by (#PCDATA)>

<!ATTLIST performed_by  e-dtype NMTOKEN  #FIXED 'string' >
<!ELEMENT copyright (#PCDATA)>

<!ATTLIST copyright  e-dtype NMTOKEN  #FIXED 'string' >
<!ELEMENT chord (#PCDATA)>

<!ATTLIST chord  e-dtype NMTOKEN  #FIXED 'string' >
<!ELEMENT lyrics (#PCDATA)>

<!ATTLIST lyrics  e-dtype NMTOKEN  #FIXED 'string' >
<!ELEMENT genre (#PCDATA)>

<!ATTLIST genre  e-dtype NMTOKEN  #FIXED 'string' >
<!ELEMENT instructions (chord | lyrics)*>

<!ELEMENT intro (chord | lyrics)*>

<!ELEMENT verse (chord | lyrics)*>

<!ELEMENT bridge (chord)*>

<!ELEMENT chorus (chord | lyrics)*>

<!ELEMENT coda (chord)*>

<!ELEMENT outro (chord | lyrics)*>
```

Figure A8: CSS Style Sheet

(basic version – just to show functionality)

```
?xml version="1.0"?>
<HTML xsl:version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<head>
<title>StruML Style.CSS</title>
<style>

TUNE {
  display : block;
  background-color : silver;
  font-size : 12pt;
  width : 100%;
  padding : 1px;
  border : 3px
  border-color : black;}

Legal {
  display : block; font-family : Arial Black;
  font-size : 18pt;
  border-style : none;}

 Song {
  display : table;
  color : #silver;
  border : 1px;
  padding : 1px;
  border-style : solid;}

 Instructions {
  display : table-cell;
  font-family : Times;
  color : black;}

 Chord { display : table-cell;
  width="25%";
  color : red;
  border-style : none;}

 Image {
  display : table-cell;
  IMG {height: 30px; margin: 0; padding: 0;}}

 Lyrics {
  display : table-cell;
  width="100%";
  font-family : Arial;
  color: blue;
  border-style : none;}

</style>  </head>
<body></body>
</html>
```

References

Castro, Elizabeth (20001). XML for the world wide web: Visual quick start guide.. Berkeley, Peachpit Press

Charnasse, Helene and Stepien, Bernard (1992). Automatic transcription of German lute tablatures – an artificial intelligence application. In Alan Marsden and Anthony Pople (Eds.), Computer representations and models in music (pp.143-170). San Diego, Academic Press Inc.

Eaglestone, Barry M. (1992). Extending the relational database model for computer music research. In Alan Marsden and Anthony Pople (Eds.), Computer representations and models in music (pp.41-66). San Diego, Academic Press Inc.

Forte, Allen (1967). Syntax-based analytic reading of musical scores. Cambridge, Massachusetts Institute of Technology (Project MAC – an MIT research project sponsored by the Advanced Research Projects Agency, Department of Defense, under the Office of Naval Research Contract Nonr-4102(01). Also appeared in the Winter 1966 issue of the Journal of Music Theory.

Huron, David (1992). Design principles in computer-based music representation. In Alan Marsden and Anthony Pople (Eds.), Computer representations and models in music (pp.5-39). San Diego, Academic Press Inc.

Marsden, Alan and Pople, Anthony (1992). Introduction: Music, computers, and abstraction. In Alan Marsden and Anthony Pople (Eds.), Computer representations and models in music (pp.1-4). San Diego, Academic Press Inc.

GNU Lilypond. Nieuwenhuizen, LilyPond homepage Available: http://lilypond.org/development/. Last update 13 APR 2001. Last referenced 15 April 2001.

MusicML. van Rotterdam, Jeroen. Untitled. Available: http://www.tcf.nl/3.0/musicml/. Last referenced 15 April 2001.

ChordML. Frederico, Gustavo. ChordML. Available: http://www.cifranet.org/xml/ChordML.html. Last referenced 15 April 2001.

SMDL – Standard Music Description Language. Mounce, Steve. SMDL Information. Available: http://www.techno.com/smdl.htm and http://www.student.brad.ac.uk/srmounce/smdl.html. Last update February 2001. Last reference 15 April 2001.

MusiXML, Castan, Gerd. Common music notation and computers: Motivation. Available: http://www.s-line.de/homepages/gerd_castan/compmus/MusiXML_e.html Last update 4 April 2001. Last referenced 15April 2001.

NIFF - Notation Interchange File Format. Mounce, Steven. NIFF Information. Available: http://www.student.brad.ac.uk/srmounce/niff.html. Last update November 1997. Last reference 15 April 2001.

MNML - Music Notation Markup Language. Peter Chiam Yih Wei, Kartik Narayan, Leong Kok Yong and Dr. Tan Tin Wee Internet Research and Development Unit, Computer Centre, National University of Singapore. MNML Syntax V2.0. Available: http://www.oasis-open.org/cover/mnmlv200.html Last referenced 15 April 2001.

FlowML – Schiettecatte, Bert. FlowML: today's and tomorrow's audio production. Available: http://tinf2.vub.ac.be/~bschiett/flowml/Copyright/frameset_Copyright.html. Last referenced 15 April 2001.

MuTaTeD. Dr. Stephen Arnold, Carola Boehm , Dr. Cordy Hall, MUTATED! - Music Tagging Type Definition, Department of Music, University of Glasgow. Available: http://www.pads.ahds.ac.uk/mutated. Last referenced 15 April 2001.

JscoreML. Weinkauf, David. JscoreML. Avalilable: http://nide.snow.utoronto.ca/music. Last update 20 February 2001, Last referenced 15 April 2001.

HyTime - Hypermedia/Time-Based Structuring Language. Rutledge, Loyd. HyTime. ISO 10744:1997 -- Hypermedia/Time-based Structuring Language (HyTime), 2nd Edition Available: http://xml.coverpages.org/hytime.html. Last update 29 January 1999. Last Accessed 15 April 2001.