Proceedings

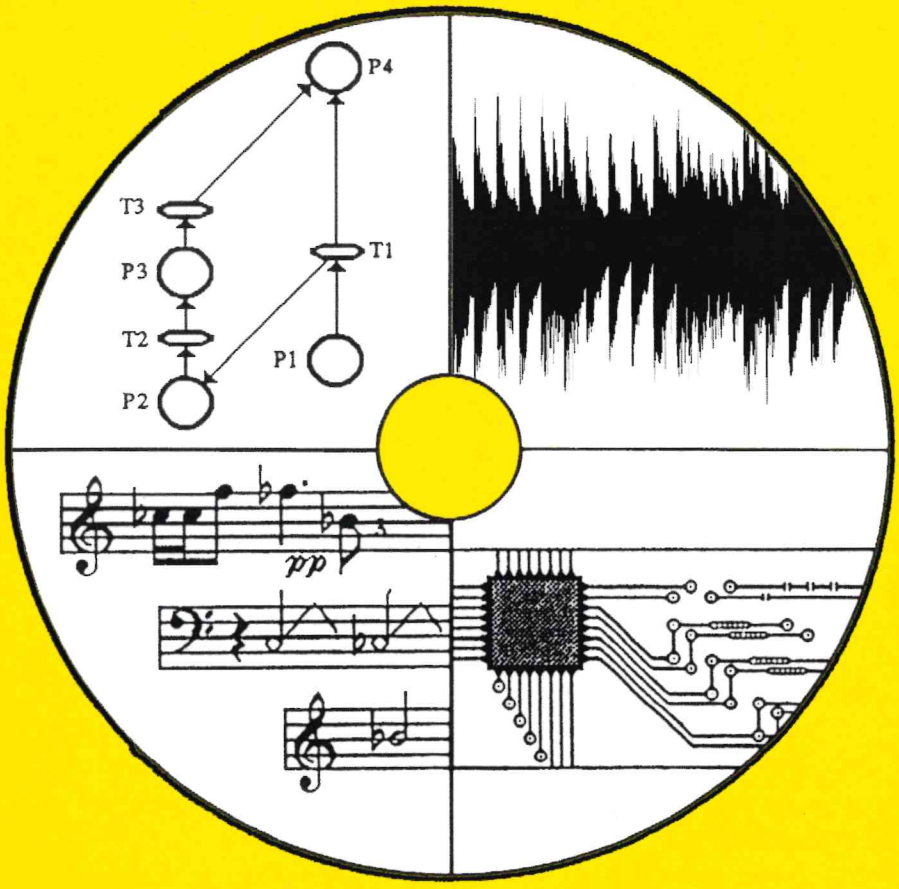
First International Conference

MAX 2002

Musical Application using XML

September 19-20, 2002

Editors: Goffredo Haus & Maurizio Longari



IEEE SA Working Group on
Music Application of XML

IEEE Computer Society Technical Committee on
Computer Generated Music

LIM-DSI
Laboratorio di Informatica Musicale
Computer Science Department
State University of Milan

TECHNICAL COMMITTEE

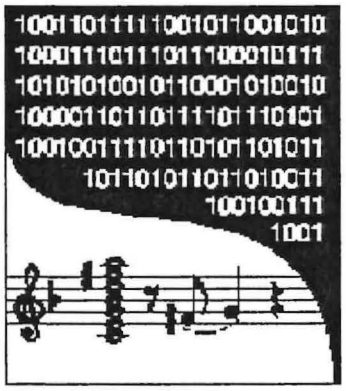
Denis L. Baggi	(CIMSI Manno, Switzerland)
Goffredo Haus	(State University of Milan)
Maurizio Longari	(State University of Milan)

ORGANIZING COMMITTEE

Denis Baggi, Goffredo Haus, Maurizio Longari, Massimiliano Pancini, Maria Luisa Pelegrin Pajuelo

LIM-DSI, State University of Milan

MAX 2002 is sponsored by:



IEEE Computer Society
Technical Committee on
Computer Generated Music

IEEE Standard Association

IEEE Computer Society

PREFACE

Much effort has been spent in the last forty years to improve both subsymbolic and symbolic music coding and processing. I think that nowadays more effort has to be spent to improve music communication.

The IEEE Standards Association Working Group for the "Definition of a Commonly Acceptable Musical Application Using the XML Language" can be considered a first collective step towards this final goal. The recommended practice which we will try to define in the next two years is dedicated to improving capabilities of individuals and companies in the areas of:

- music processing and delivering, at multiple levels of representation;
- modelling for music analysis and synthesis;
- defining and implementing tools for automatic music understanding;
- defining and implementing tools for composing and for performing music;
- defining and implementing tools for music browsing within the web.

I am very glad I can salute and thank all those who are contributing to this working group, and also those who may not currently believe in this cooperative effort, but who, I hope, may come to believe in it in the near future.

Finally, a special thank to Denis Baggi and Maurizio Longari for their fundamental help in preparing the event.

Goffredo Haus
Director, LIM

The Activity of the Technical Committee on Computer Generated Music Within the IEEE CS

The **Institute of Electrical and Electronic Engineers,** know as **IEEE,** is the largest professional association dedicated to electrical engineering, worldwide, with its 400,000 members (see *www.ieee.org*). The **Computer Society,** or **CS,** is a section of the IEEE which deals specifically with computing science and technology, and has about 100,000 members worldwide (see *www.computer.org*).

There are several bodies within both the IEEE and the CS. The most relevant to the activities described here are: the **S**tandard **A**ctivity **B**oard or **SAB,** which is responsible for the definition of standards, from computer languages to busses and plugs. And the **T**echnical **A**ctivity **B**oard or **TAB,** which is the set of **T**echnical **C**ommittees, similar to interest groups, dedicated to various aspects of Computer Science, from hardware to languages, from music to software engineering. The number of TC's varies, as old ones cease their activities and new ones start, and it is normally about three dozens.

Among the TC's there exists the ***Technical Committee on Computer Generated Music***. It started from the activity which lead to the publication of the July 1991 issue of IEEE COMPUTER, the flag magazine of the IEEE CS, dedicated entirely to Computer and Music. That was the first time the CS Press had published a CD-Audio that went with the published material, and that the quality of a paper depended not only on the written contents, but also on the sound and music material that went with it.

The TC was established in a preliminary form as a Task Force in 1992 and became a full TC in 1994. The book *Readings in Computer Generated Music* was published by the CS Press in 1992.

It activity continued with the printing of the CD-ROM "An Intelligent Music Workstation" in 1994, with examples and programs, and the printing of another CD-ROM in 1995 "Standards in Computer Generated Music", with text, programs and descriptions of existing standards, in HTML, that can be navigated with any browser.

In September 2001, the TC on CGM proposed to SAB the new standard **"Definition of a Commonly Acceptable Musical Application Using the XML Language".** The idea is to use the flexibility and readability of XML to standardize music, symbolic music, sounds, scores and make them universally available for browsers, media such as DVDs and CD-ROMs. This should obviate the inherent deficiencies of existing de-facto standards, which are either incomplete (MIDI), binary and proprietary (AIFF, WAV, MP3). More of that is described in the articles below. In addition, while obviously the IEEE is not involved in any financial support, since all activities - except staff - are based on voluntary service, it has been possible to submit the work on this standard to the international research fund **Intelligent Manufacturing Systems** and have the project abstract endorsed unanimously, as explained below.

To conclude, the IEEE CS TC on Computer Generated Music has been in existence for about 10 years. It represented a new challenge for the IEEE, to accept an interest group dedicated to what most computer scientists consider a fringe area. As a result, the IEEE CS has added Computer and Music to the list of topics of interest in the form one fills when joining and renewing membership. It is hoped that, with such activities, the interest in computing applications to music and musicology will be kept high, both in the interest of the IEEE CS as in that of all workers in the field.

Denis Baggi
Chairman, IEEE CS Technical Committee on Computer Generated Music
Institute CIMSI of the Professional University of Southern Switzerland, SUPSI

Synergy Between IEEE And International Research Funds:
A Case Of Having Your Standard Financed

The IEEE **S**tandard **A**ctivity **B**oard **(SAB)** has a standard procedure for the submission of proposal for *standards,* which starts from a **P**roject **A**uthorization **R**equest **(PAR)**, as clearly explained at **www.ieee.org.**

Last September, the *Technical Committee on Computer Generated Music* proposed to SAB the new standard **"Definition of a Commonly Acceptable Musical Application Using the XML Language"**, which defines codification of music, sounds and the like for transmission via WEB, reproduction on DVDs, CD-ROMs, namely, where present de-facto standard (MIDI, Wav, Aiff) are incomplete. The proposal has been accepted, labeled as **PAR1599** and work is proceeding with two IEEE conferences, **MAX2002** in Milan in November 2002 and a **Workshop** in 2003. The TC is grateful for the support by the IEEE and SAB for the work involved, but of course there no financing was expected.

Intelligent Manufacturing Systems (IMS) is an international research fund sponsored by *regions*, which are: Australia, Canada, European Union, Japan, Korea, Switzerland and the United States of America (see **www.ims.org**). A project has to involve at least two regions, and the abstract must be endorsed by all regions unanimously. Project **A Musical Application Standard Using the XML Language For Intelligent Manufacturing of Music for CDs, DVDs, WEB** has had its abstract endorsed in August 2002. It is about the realization of PAR1599 mentioned above, and its description can be found at the IMS Web site under *Projects*, *Endorsed Abstracts*.

Still much remains to be done for the actual start of the IMS project. The *International Project Coordinator* has to prepare a thorough definition and coordination of all project steps. Each *Regional Project Coordinator* will have to prepare a financial plan for its national funding agency (in the US, the NSF; in the EU, the European Framework in Brussels; in Switzerland, the Federal Office for Education and Science). Nevertheless, this attempt may pave the road for people in academia, independent researchers, scientist who operate outside of large companies that have a budget for such activities, to contribute a standard thanks to the **IEEE** and have it financed by some other research body such as **IMS**.

Denis Baggi
Chairman, IEEE CS Technical Committee on Computer Generated Music
Institute CIMSI of the Professional University of Southern Switzerland, SUPSI

Table of Contents

Full Papers Session

MPEG Audio Coding and XML: samples, models, descriptors 3
Giorgio Zoia, Ruo-hua Zhou, Marco Mattavelli

Automatic Marking of Musical Dictations By Applying the
Edit Distance Algorithm On a Symbolic Music Representation 11
Guy Tremblay, France Champagne

A Proposal for a Regular Grammar To Parse Jazz Chords Notation.....18
Denis. L. Baggi

Framework for a music markup language 22
Jacques Steyn

Modeling Music Using XML – Some Basic Consideration -- 30
Markus Lepper

Towards a Symbolic/Time-Based Music language based on XML 38
Goffredo Haus, Maurizio Longari

MusicXML in Practice: Issues in Translation and Analysis 47
Michael Good

The Music Encoding Initiative (MEI) 55
Perry Roland

Demo Session

Actos: A Peer-to-Peer Application for the Retrieval of Encoded Music.....63
Gustavo C. S. Frederico

A case study: Prepared-Piano Notation in XML.....64
M. Longari

Background Papers

Learning our Lessons from SMDL.....69
Donald Sloan

Notation Interchange File Format NIFF 6b.....75

Music Information Description by Mark-up Languages within DB-Web Applications...83
Goffredo Haus, Maurizio Longari

Learning our Lessons from SMDL.....91
Goffredo Haus, Maurizio Longari

Full Papers Session

MPEG Audio Coding and XML: samples, models, descriptors

Giorgio Zoia, Ruo-hua Zhou, Marco Mattavelli
Signal Processing Institute
EPFL – ITS/LTS3
1015 Lausanne, Switzerland
+41 21 6936981
{Giorgio.Zoia, Ruohua.Zhou, Marco.Mattavelli}@epfl.ch

ABSTRACT
The different MPEG standards are well-known for their media compression capabilities. Starting with MPEG-4, the goal of the standardization activity has also included media representation, indexation and retrieval. MPEG-4 joins advanced compression technologies with high-level media representations such as synthesized content and scene descriptions. MPEG-7 defines standard media descriptors and the way in which these descriptors can be compressed if desired, handled and extended in user defined description schemes. In this paper first the main audio-related features of the last MPEG standards, which go well beyond simple compression, are briefly reviewed. Then some examples of applications are developed showing how XML can be profitably used in conjunction with MPEG-4 and MPEG-7 schemes in the field of musical applications.

Keywords
MPEG, Sound modeling, Sound description, indexing, retrieval.

1 INTRODUCTION

The fundamental migration of sounds and music from analog to digital representations started about two decades ago. Essentially, the main and more remarkable benefit behind this radical change was the possibility to move from sound recordings as "memory" of an event and base for its distribution to sound production as starting point for sound and music editing, manipulation and control. Even more, a second step in the evolution of this digital era is the capability to join together different levels of the media representation (the sound itself, models of the sound, descriptions of the sound content, and so on) into a single digital item that can provide to the user a wider range of experiences.

Expressed in another way, it can be stated that the main benefit of digital audio is not really an improved quality in sound storage and reproduction (which is true indeed, but mostly due to the comparison of new digital devices with older standard analog equipments), but rather the new possibilities in the manipulation of sound, in its representation and classification, in the degree of potential interaction with the sonic media itself.

The ISO committee SC29WG11, better known as MPEG, has, from the very beginning of its successful and widespread standards, been sensible to the digital item representation in relationship with the state-of-the-art in technology. If the first two international standards, known as MPEG-1 and MPEG-2, were aiming essentially at information coding to reduce storage and bandwidth requirement in a perceptually non degraded way for natural audio and natural video, the following phases had as main focus the multimedia representation including synthetic audio and video objects (MPEG-4) and content description and indexation (MPEG-7).

This paper intends to put in evidence how far the MPEG standards have gone so far by introducing the latter and not so well-known features and capabilities. Furthermore, some meaningful examples will be presented of how MPEG-4 and MPEG-7 technologies are close to an XML-based description, in particular MPEG-7, which is indeed based on XML description schemes. The paper is organized as follows. The second section of the paper is dedicated to a brief introduction to MPEG-4 Audio, to MPEG-4 scene description mark-up language (BIFS) and to the basic concepts of MPEG-7 Audio. The third section shows how existing, MPEG-based tools can be integrated through XML descriptions. To conclude, in the last section some straightforward extensions are proposed and two meaningful examples of applications related to the world of music are presented. They can be easily specified by the XML descriptions previously presented and implemented basing on standard MPEG-4/MPEG-7 platforms.

2 ADVANCED MPEG AUDIO (AND MUSICAL) TOOLS

As previously remarked, if MPEG-1 and MPEG-2 were essentially focusing on perceptual natural media coding, starting with MPEG-4 and continuing with MPEG-7 the attention was extended to more advanced contexts like

0-7695-1864-8/02 $17.00 * 2002 IEEE

hybrid natural and synthetic multimedia representation formats, content analysis and descriptions. The following subsections will introduce the main concepts behind the last audio-related MPEG features.

MPEG-4 Hybrid Audio Coding

The MPEG-4 standard[1] represents an effort to focus on all those aspects of sound coding and delivery that deal with multimedia content up to user interaction and virtual reality applications. MPEG-4 actually provides hybrid, object-oriented profiles that combine more conventional, yet high-quality schemes like perceptual coding (e.g. for dry tracks or, in general, conventional recordings), and structured voice coding (e.g. based on linear prediction codecs) within a highly structured scene description in form of a post-processing subtree. At the same time, in order to deal with new or custom technologies, this subtree is extensible, because general structured synthesis and processing blocks can be added by means of a dedicated programming language.

In MPEG-4 there are two fundamental layers that are used for content description. At the first layer, which can be called here for clarity the *content layer*, audio-video "objects" are delivered to a terminal as elementary streams and they are converted back to samples and pixels by several instances of the corresponding decoders. The sources are then made accessible at the upper layer, the *systems layer* (or, more intuitively, the scene description layer), which takes care of the manipulation of the parameter fields of the scene, composed by a structured description and by some interaction and update mechanisms described below in more details.

In the audio part of the content layer, three different approaches are made available to feed the audio scene with voices and sounds: low structured ("natural") coding techniques, classic model-based voice coding techniques (including a text-to-speech interface) where the decoding algorithm is fully specified, and finally the programmable structured audio coding, where synthetic or newly defined algorithms can be described through SAOL (Structured Audio Orchestra Language). It must be clear that MPEG-4 SA (Structured Audio) does not represent a simple compression or synthesis method, but a toolset for describing compression and/or synthesis methods. All the necessary information about the audio coding tools available in MPEG-4 has been extensively described in literature (see for instance [2, 4]).

SA [8] derives from software sound synthesis (SWSS) languages; the SAOL programming language is similar to many of these SWSS tools, but the SA toolset as a whole has been conceived for multimedia and downloadable applications; the syntax of the language itself is very close to that of C, with stronger object-oriented connotations and several stream-oriented extensions.

Note further that a structured audio coding scheme can be seen as a general coding approach [7]; in these terms, decoding is a synonym for program execution. While providing a maximum of flexibility and extensibility, SAOL can be considered as a general purpose DSP language; consequently, it may be not very efficient when other established and specific technologies are available, like in the case of perceptual or voice coders.

MPEG-4 Audio Scene Descriptions

At the system layer, BIFS (Binary Format for Scenes) is a mark-up language for scene descriptions; it is a language very similar in its hierarchical structure to VRML [9], but with an innovative audio-specific processing subtree, often referred to as "AudioBIFS". The functionality of AudioBIFS ranges from simple mixer or delay "nodes" up to advanced spatialization schemes based on geometrical and perceptual information, so that a complete virtual audio environment can be precisely described (for more details see [4], Chapter 6).

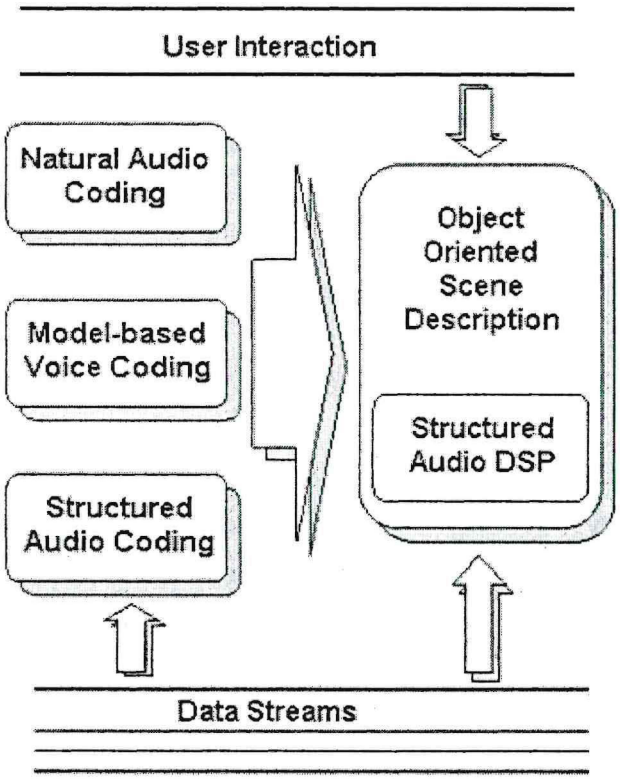


Figure 1 *MPEG-4 SNHC Coding Scheme for Audio*

At the scene description layer, both highly structured, well-parameterized spatialization techniques (for either perceptual and/or geometrical descriptions) and programmable DSP algorithms for custom functionality are

[1] For a more comprehensive overview see [1, 5].

supported. User defined processing algorithms, again described by means of dedicated SAOL code, can be included in the scene tree graph through the AudioFX node.

In addition to binary coding, it is currently under discussion in MPEG a textual format for scene descriptions that will be based on XML schemas. This will permit to describe the same AudioBIFS scenes with all the MPEG-4 features by XML files. Only draft documents are available while writing, consequently more details and references will be eventually included in the final version of this paper.

Figure 1 sketches the overall architecture of the Audio coding scheme of MPEG-4 from a natural/structured perspective. Audio objects, described at different levels of bitrate/optimization by the three different approaches, flow through a post-processing flexible subtree of nodes until the rendering stage interface (the actual rendering requirements are often open in MPEG-4). Update mechanisms guarantee the possibility for a dynamic scene evolution, since update commands can be received in a dedicated stream and interpreted by the scene decoder. At the same time the author may decide to expose some fields of the scene to user interaction, so that parameters like e.g. mixer matrices, delay lengths, reverb times and so on can be interactively manipulated by the user.

So as to guarantee an appropriate quality of service and the feasibility of the complete system, appropriate Profiles and Levels and related Conformance testing are established, where a subset of tools is defined, fundamental parameters are limited and quality assessment tests are specified.

MPEG-7 Audio Description Schemes

While MPEG-4 focuses on the representation and delivery of multimedia content, in many different forms and allowing several degrees of user interaction with the content, MPEG-7 targets a different problem in the media world. Formally named "Multimedia Content Description Interface", MPEG-7 provides a set of standardized tools to describe multimedia content. Both human users and automatic systems that process audiovisual information are within the scope of MPEG-7 [3, 6].

The main elements of the MPEG-7 standard are:

- Description Tools: Descriptors, that define the syntax and the semantics of each feature (metadata element); and Description Schemes (DS), that specify the structure and semantics of the relationships between their components, that may be both Descriptors and Description Schemes themselves;
- A Description Definition Language (DDL); MPEG-7 employs XML as the language for the textual representation of content descriptions, since XML Schema has been the base for the DDL, that is used for the syntactic definition of MPEG-7 Description Tools and to allow extensibility of Description Tools themselves (either new MPEG-7 ones or application specific). Considering the popularity of XML, its usage facilitates interoperability with other metadata standards;
- System tools, to support both textual and binary coded representation for efficient storage and transmission, transmission mechanisms (for both textual and binary formats), multiplexing of descriptions, synchronization of descriptions with content, management and protection of intellectual property in MPEG-7 descriptions, etc.

In particular, MPEG-7 Audio is based on five technologies: the Audio Description Framework, musical instrument timbre description tools, sound recognition tools, spoken content description tools, and melody description tools. The Audio Framework contains low-level tools designed to provide a basis for the construction of high level audio applications. By providing a common platform for the structure of descriptions and the basic semantics for commonly regarded audio features, MPEG-7 Audio establishes a platform for interoperability across all applications that might be built on the framework.

There are essentially two ways of describing low-level audio features. It is possible to sample values at regular intervals or to use Segments to demark regions of similarity and dissimilarity within the sound. Both of these possibilities are embodied in two low-level descriptor types (one for scalar values and one for vector values types), which create a consistent interface. Any descriptor inheriting from these types can be instantiated, describing a segment with a single summary value or a series of sampled values, as the application requires. The low-level audio descriptors are of general importance in describing audio. There are seventeen temporal and spectral descriptors that may be used in a variety of applications.

In addition to low-level basic descriptors, five sets of high level audio description tools, which try to represent some application areas, are integrated in the standard: sound recognition, musical instrument timbre, spoken content, melodic contour, and melody. Some examples of these description tools are given in the following of the paper, in order to show the potential use of the MPEG-7 approach in a musical context.

3 DESCRIBING MPEG TOOLS THROUGH XML

This section is dedicated to provide some examples of how already existing MPEG tools, from both MPEG-4 and MPEG-7 standards, can interact together using XML as merging language. Such possibility is extremely interesting because all existing MPEG decoders and terminals are suitable implementation ready to use for audio scenes described in XML. AudioBIFS nodes and SA are treated

first, while examples with MPEG-7 description tool follows in the second part of the section.

AudioBIFS nodes and XML

BIFS is a mark-up language, used for scene descriptions. It is straightforward to consider an XML schema as a description of AudioBIFS nodes, so that a scene can then be described by an XML instance. As already noted, a similar effort is currently ongoing in MPEG and it will be eventually reported in the future; here a few examples are made, which have been prepared by the authors to clarify at best the concepts under discussion. Without entering in too many details, for instance an AudioBIFSNode type, including all the MPEG-4 version 1 AudioBIFS nodes, is defined as follows:

```
<complexType name="AudioBIFSNodeType">
  <choice>
    <element name="AudioSource" type="AudioSourceType"
minOcuurs="0" maxOccurs="1"/>
    <element name="AudioDelay" type="AudioDelayType"
minOcuurs="0" maxOccurs="1"/>
    <element name="AudioMix" type="AudioMixType"
minOcuurs="0" maxOccurs="1"/>
    <element name="AudioSwitch" type="AudioSwitchType"
minOcuurs="0" maxOccurs="1"/>
    <element name="AudioFx" type="AudioFxType"
minOcuurs="0" maxOccurs="1"/>
    <element name="AudioClip" type="AudioClipType"
minOcuurs="0" maxOccurs="1"/>
    <element name="AudioBuffer" type="AudioBufferType"
minOcuurs="0" maxOccurs="1"/>
  </choice>
</complexType>
```

In the same way, all the types introduced above can be defined specifying their fields. On top of all that, the Sound node is used to attach sound to the scene (in a way very similar to VRML). Sound is defined in the schema as follows:

```
<element name="Sound" type="SoundType">
<complexType name="SoundType">
    <sequence>
    <element name="direction" type="SFVec3fType"
minOccurs="0" maxOccurs="1" default="0 0 1"/ >
    <element name="intensity" type="zeroToOneFloatType"
minOccurs="0" maxOccurs="1" default="1.0"/ >
    <element name="location" type="SFVec3fType"
minOccurs="0" maxOccurs="1" default="0 0 0"/ >
    <element name="maxBack" type="nonNegativeFloatType"
minOccurs="0" maxOccurs="1" default="10.0"/ >
    <element name="maxFront" type="nonNegativeFloatType"
minOccurs="0" maxOccurs="1" default="10.0"/ >
    <element name="minBack" type="nonNegativeFloatType"
minOccurs="0" maxOccurs="1" default="1.0"/ >
    <element name="minFront" type="nonNegativeFloatType"
minOccurs="0" maxOccurs="1" default="1.0"/ >
    <element name="priority" type="zeroToOneFloatType"
minOccurs="0" maxOccurs="1" default="0.0"/ >
    <element name="source" type="AudioBIFSNodeType"
minOccurs="0" maxOccurs="1" />
    <element name="spatialize" type="boolean" minOccurs="0"
maxOccurs="1" default="true"/ >
    </sequence>
</complexType>
```

The *source* field of the Sound node is used to attach the subtree to the rendering device(s). In the same way any node of the subtree, belonging to the AudioBIFSNode type, can have one or more children of the same type, other than other fields to manipulate the content. Particularly important is the example of the AudioFX node, which permits to attach to the scene processing algorithms written in SAOL. The children nodes provide the input channels to the SAOL orchestra, the output of the orchestra will be the input to the parent node.

```
<complexType name="AudioFxType">
    <sequence>
     <element name="nodeID" type="string" minOccurs="0"
maxOccurs="1"/>
    <element name="numChan" type="nonNegativeInt0-255"
minOccurs="0" maxOccurs="1" default="1" />
    <element name="phaseGroup" type=" Int0-255Vector"
minOccurs="0" maxOccurs="1" default="1" />
     <element name="orch" type="string" minOccurs="0"
maxOccurs="1" default=""/ >
     <element name="score" type="string" minOccurs="0"
maxOccurs="1" default=""/ >
     <element name="params" type="floatVector"
minOccurs="0" maxOccurs="1" default=""/ >
     <element name="child" type="AudioBIFSNodeType"
minOccurs="0" maxOccurs="unbounded" />
     <element name="addChild" type="AudioBIFSNodeType"
minOccurs="0" maxOccurs="unbounded" />
     <element name="remChild" type="AudioBIFSNodeType"
minOccurs="0" maxOccurs="unbounded" />
    </sequence>
</complexType>
```

In the code above, the *orch* and *score* fields are used to attach the orchestra and its optional score, respectively. At the beginning of the audio subtree, the AudioSource node permits to attach any of the allowed MPEG-4 decoders (identified by an Object Type ID) as input, being it either natural recorded and encoded sounds, or synthesized sounds again described in SAOL or through the Text to Speech Interface.

At this point the following XML code is e.g. the description of the scene graphically represented in Figure 2.

```
<Sound>
  <direction> 0 0 0</direction>
  <intensity>0.8</intensity>
  <location>0 0 1</location>
  <source>
    <AudioMix>
      <numInputs>4</numInputs>
      <matrix>0.25 0.25 0.25 0.25</matrix>
      <numChan>1</numChan>
      <child>
```

```
            <AudioFx>
                <numChan>2</numChan>
                 <orch>
                   <!-- follows orchestra code-->
                 </orch>
                 <child>
                    <AudioSource>
                          <url>DecID1</url>
                          <startTime>0</startTime>
                          <stopTime>-1</stopTime>
                          <numChan>1</numChan>
                     </AudioSource>
                  </child>
            </AudioFx>
        </child>
        <child>
          <AudioDelay>
                <delay>10</delay>
                <child>
                     <AudioSource>
                          <url>DecID2</url>
                          <startTime>0</startTime>
                          <stopTime>-1</stopTime>
                          <numChan>2</numChan>
                      </AudioSource>
                </child>
            </AudioDelay>
        </child>
    </AudioMix>
   </Source>
</Sound>
```

In the example, the audio objects ID1 and ID2 are attached to two AudioSource nodes, then the first one is post-processed through an AudioFx node, the other is delayed in the AudioDelay node, then the two outputs of AudioFx and AudioDelay are mixed together in the AudioMix node, finally the output of AudioMix nodes is sent to Sound for spatial rendering.

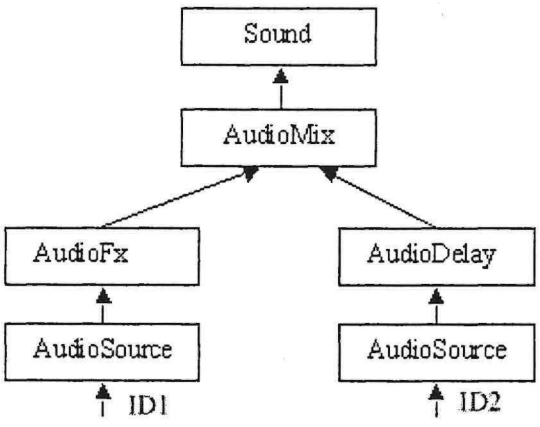


Figure 2 *BIFS scene example*

MPEG-7 Description Tools and XML

The case of the MPEG-7 description tools is straightforward, since they are already defined in DDL, which is based on XML. This means that there is (once in a while!) no annoying work to be done in order to yield an XML Schema for the low-level and high-level description schemes. In this way, once the AudioBIFS nodes are described in XML as presented in the previous subsection, it is possible to start from standard MPEG tools and, with or without further extensions, build quite interesting applications in the world of music. This development is presented with two examples in the next section.

4 EXAMPLES AND APPLICATIONS

MPEG-7 provides some high-levels description schemes that can be very useful in the case of music, and for sure the Melody descriptor is one of them. In its basic MPEG definition for instance, content is characterized by its melodic contour or melodic sequence, without any specific link with the *actual* content. This is conceived for typical application contexts like user-server queries with subsequent action, which is application dependent. At the same time, it may be possible to link an MPEG-like description with specific "decoders", like done in the MPEG-4 AudioSource node described earlier, so that it can be more practically put in relation with musical content and it can give rise to a more application-near context.

Melody and synthetic sound

In this sub-section the possibility to extend the MPEG-7 Melody DS (Description Scheme) not only as melody feature description for natural or synthetic audio, but also as synthetic music content carrier is analyzed. In such a way, a certain melody descriptor can be associated with a specific orchestra, so that the melody can be for instance listened by the synthesized instruments that are associated with it.

In order to perform a piece of music, three basic elements are necessary: instrument, performer, and score. MPEG-4 structured audio toolsets provide the functionality to describe the instrument synthesis methods (SAOL, as said earlier). Moreover, SASL (Structured Audio Score Language) describes the manner in which sound-generation algorithms described in SAOL are used to produce sound. The SA scheduler interfaces with the computer clock to perform the score on the instruments. It is possible to integrate the MPEG-4 structured audio toolset into an extended MPEG-7 melody Description Scheme to support more defined applications.

If the MPEG-7 melody DS is extended to include more elements, like for instance a SAOL orchestra, this is enough to play the synthetic music; consequently, it is possible at the same time to support not only the melody retrieval, but also a compact sound description and download.

```
<complexType name="ExtendedMelodyType">
  <complexContent>
    <extension base="mpeg7:AudioDSType">
```

```
      <sequence>
       <element name="MelodyMeter" type="mpeg7:MeterType"
          minOccurs="0"/>
       <element name="MelodyScale" type="mpeg7:MelodyScale
Type" minOccurs="0"/>
       <element name="MelodyKey" type="mpeg7:MelodyKey
Type" minOccurs="0"/>
       <element name="MelodySequence" type="mpeg7: Melody
SequenceType" minOccurs="0" maxOccurs="unbounded"/>
      <element name="Orchestra" type="string" minOccurs="0"/>
      </sequence>
    </extension>
  </complexContent>
 </complexType>
```

This means that for instance the result of a query may return not only some results, but also a fast and compact way of listening to this result and refining, eventually, the query itself. In this first extension, only a new element allowing the inclusion of an orchestra is present, making a more complex descriptor, which actually includes some form of *content* out of the MPEG-7 Melody DS as such. It is of course possible to extend even more the description scheme in order to include an enhanced MelodySequence, that we may call *HarmonySequence*, so that even polyphonic scores may be described

```
<complexType name="HarmonySequenceType">
  <complexContent>
    <extension base="mpeg7:AudioDSType">
      <sequence>
       <element name="NoteArray" type="mpeg7:NoteArrayNew
Type"/>
      </sequence>
    </extension>
  </complexContent>
 </complexType>
```

with

```
<complexType name="NoteArrayNewType">
  <complexContent>
    <extension base="mpeg7:AudioDSType">
      <sequence>
        <element name="Note" maxOccurs="unbounded">
          <complexType>
            <sequence>
              <element name="Pitch" type="MIDINoteType"/>
              <element name="Instrument"
                type="InstrmentType" minOccurs="0"/>
              <element name="NoteRelDuration"
                    type="float"/>
              <element name="NoteStartTime"
                    type="float"/>
              <element name="Lyric" type="mpeg7:Word
Type" minOccurs="0"/>
              <element name="PhoneNGram" minOccurs="0">
               <simpleType>
                <list itemType="mpeg7:PhoneType"/>
               </simpleType>
              </element>
              <!-- See Spoken Content for details -->
             </sequence>
            </complexType>
          </element>
        </sequence>
      </extension>
    </complexContent>
</complexType>

<simpleType name="MIDINoteType">
  <restriction base="integer">
    <minInclusive value="0">
    <maxInclusive value="127">
  </restriction>
</simpleType>
```

In this second and more complex case, it may even be possible to describe polyphonic scores, to associate each note to a particular instrument, characterized by specific parameters and so on, exactly in the way a score language acts.

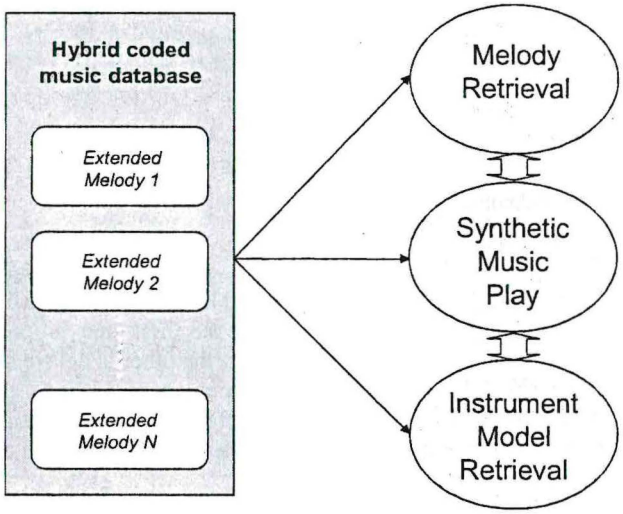


Figure 3 *The extended Melody DS supports several applications*

As already mentioned, from an MPEG-7 point of view, this implies an extension of the Description Scheme to include some form of content (another example is given in the next subsection) in relationship with the description itself. From an MPEG-4 point of view, this means an extension (or, better, an extensibility) of the BIFS mark-up language to include more fields, and in particular some metadata related to a specific content and/or a specific decoder. From a general point of view, this means the possibility to build multi-layered musical descriptions with just a few extensions to two existing standards, in a way that permits to reuse several existing components and many established results in standardized musical-related descriptions; it means also to save, as far as possible, interoperability with

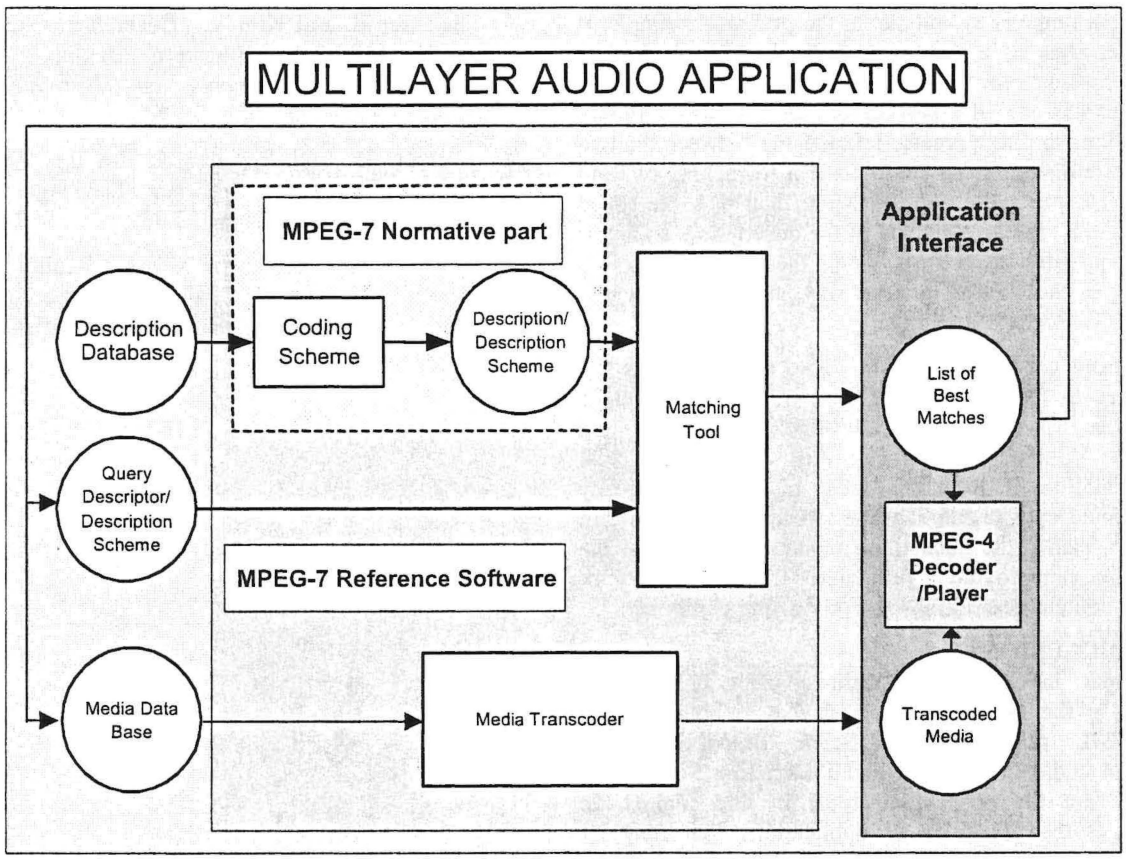


Figure 4 *Block diagram for a sample application showing the several standardized blocks, databases and tools*

these other standardized descriptions and/or related tools. The generalized application range made possible by this example is graphically summarized in Figure 3.

A more detailed block diagram is reported in Figure 4 showing the standard MPEG-4 and MPEG-7 elements constituting the described applications. Standards elements are meant in terms of "bitstreams" (content), "decoders" (processing elements) and "interfaces".

Melody and generally coded sound

In the first example the melody DS was used adding some structured descriptions to build a complete musical application. This process can appear in some way redundant, since score languages are capable of a semantically similar functionality of a melody descriptor. However at the same time a melody descriptor has a more general approach, in the sense that it carries generic metadata that are not necessarily related to a specific structured description of sound. In the second application example that is proposed here, the Melody DS again is extended in a way even more similar to the functionality provided by the MPEG-4 AudioSource node, that is a generic decoder ID (or url) can be attached together with the metadata representing it and, possibly, with some field providing some degrees of interaction with the source or the metadata themselves. The definition in XML could be as follows:

```
<complexType name="ExtendedSourceType">
  <complexContent>
   <extension base="mpeg7:AudioDSType">
    <sequence>
     <element name="MelodyMeter" type="mpeg7:MeterType"
        minOccurs="0"/>
     <element name="MelodyScale" type="mpeg7:MelodyScale
Type" minOccurs="0"/>
     <element name="MelodyKey" type="mpeg7:MelodyKey
Type" minOccurs="0"/>
     <element name="MelodySequence" type="mpeg7: Melody
SequenceType" minOccurs="0" maxOccurs="unbounded"/>
     <element name="url" type="String" minOccurs="0"
maxOccurs="unbounded" default=""/ >
     <element name="pitch" type="nonNegativeFloatType"
minOccurs="0" maxOccurs="1" default="1.0" />
     <element name="speed" type="nonNegativeFloatType"
minOccurs="0" maxOccurs="1" default="1.0" />
    </sequence>
   </extension>
  </complexContent>
 </complexType>
```

In this case melody or melodic contour metadata can be associated with a generic content, being it synthetic or natural and coded in a general way. In this case the

application range is as wide as in the previous example, if not wider. Just as a first case, both the content and its melodic metadata (or its harmonic metadata, if we consider further extensions like before) are tied to some timing information, so that a precise relationship between the two can be established, from audio content to melody or from melody into the audio content. More than that, the speed and the pitch are two elements of the description, so that it may be possible to control, using these two fields, the playback of the content in some way, as specified by e.g. the MPEG-4 scene description.

Again, from a general point of view, this represents the possibility to build a multi-layered musical descriptions application based on MPEG-7 standard elements and MPEG-4 components, in a way that the actual implementation platform can reuse most of the already implemented components that are already available within these two standards. Such implementation synergy at the same time can provide a full interoperability framework between content descriptions and processing platforms.

5 CONCLUSIONS

This paper introduces the main concept functionality provided by the last MPEG standards that are highly related with XML representations. Scene descriptions and structured audio in MPEG-4 and Description Schemes in MPEG-7 are the outstanding tools in this context. In addition, some examples of applications that may be derived from these standards are presented. Extensibility of these descriptors, which is guaranteed by XML, can help interoperability and/or integration with other available tools, interfaces and descriptions in the musical field.

REFERENCES

1. Battista S., Casalino F., Lande C. MPEG-4 : A Multimedia Standard for the Third Millennium, Part 1. *IEEE Multimedia*, (October 1999), pp. 74-83.
2. Brandenburg K., Kunz O., Sugiyama A. MPEG-4 Natural Audio Coding, *Signal Processing:Image Communication Journal, Tutorial Issue on the MPEG-4 Standard*, Elsevier, (1999).
3. Chang S.F., Sikora T., Puri A. Overview of the MPEG-7 standard, *IEEE Transactions on Circuits and Systems for Video Technology*, No. 6 (July 2001, special issue on MPEG-7), pp.688-695.
4. Chen T. and Puri A. editors Advances in Multimedia: Standards, Systems, and Networks. Dekker (New York, 2000).
5. Koenen R. MPEG-4: Multimedia for Our Times. *IEEE Spectrum*, Vol. 36, 2, (February 1999), pp. 26-33
6. Salembier P. Overview of the MPEG-7 Standard and of Future Challenges for Visual Information Analysis. *EURASIP Journal on Applied Signal Processing*, 2002:4, pp 1-11.
7. Scheirer E. and Kim Y. Generalized Audio Coding with MPEG-4 Structured Audio. *AES 17th International Conference on High-Quality Audio Coding*, (Florence, Italy, 1999).
8. Vercoe B. and Scheirer E. SAOL: the MPEG-4 Structured Audio Orchestra Language. *Computer Music Journal* 23 (2) pp. 23-35.
9. VRML (IS 14772-1) Virtual Reality Modeling Language, (April 1997) available at the vrml website: http://www.vrml.org/technicalinfo/specifications/vrml97/index.htm

Automatic Marking of Musical Dictations By Applying the Edit Distance Algorithm On a Symbolic Music Representation

Guy Tremblay
Dépt. d'informatique
Université du Québec à Montréal
C.P. 8888, Succ. Centre-Ville
Montréal, QC, H3C 3P8, Canada
+1 514 987-3000 ext. 8213#
tremblay.guy@uqam.ca

France Champagne
Dépt. d'informatique, UQAM
Montréal, QC, Canada
fc@francechampagne.org

ABSTRACT

A key practice of basic musical training is the use of *musical dictations* for ear training and training in music writing. Marking such dictations for large groups of students can be a lot of work. In this paper, we present a tool that can help automate the marking of musical dictations.

The edit distance, which computes a similarity metric between two *strings*, has been used in various areas such as string/text analysis, protein/genome matching in biocomputing, and musical applications, for example, music retrieval or musicological analysis. The tool we present can be considered an application of the edit distance to the marking of musical dictations.

Computing an edit distance on musical scores requires using an appropriate symbolic representation. We use MusicXML, an XML application for standard Western music notation. Given an appropriate Document Type De nition for MusicXML, existing Java tools can be used to obtain a MusicXML parser. Such a parser, given appropriate input les, then generates an intermediate form (DOM object) on which analyses and transformations are performed in order to compute the edit distance. In turn, the edit distance is used to give a mark as well as identify some of the key errors.

Keywords

Symbolic music representation, marking of musical dictation, edit distance and sequence comparison.

1 INTRODUCTION

A key skill in basic musical education is the knowledge of *solfège*, that is, the ability to recognize the signs and symbols of musical notation and associate them with the sounds and rhythms they represent. One important pedagogical tool for learning the basic rules of music and solfège is the use of *musical dictation*. In a musical dictation exercise, the teacher plays a music piece and the students must recognize and write down, in standard Western music notation, the piece being played. The goal is to train the students in auditory recognition of musical pieces as well as train them in basic music writing.

Marking musical dictations, like any kind of marking, can be a tedious task, especially with large groups of students as typically found in introductory classes. The goal of the project presented in this paper is to develop a tool that will help music teachers evaluate and mark their students' music dictations. A longer-term goal for such a tool is to integrate it within an intelligent tutorial system, in order to identify students' recurring weaknesses in solfège and, thus, help improve their learning.

A key element of the kind of tool we envision is a software component that can detect the similarities and differences between two music scores, both encoded in a symbolic representation appropriate for Western music notation, that is, as sequences of *notes* with speci c *pitch* and *duration*. Sequence comparison algorithms can be applied to such sequences, as typically used in text searching or biocomputing, to compute an appropriate metric that can be used for marking.

The paper is organized as follows. Section 2 discusses the basic characteristics of musical dictations and the need for an appropriate symbolic representation. The next section

0-7695-1864-8/02 $17.00 * 2002 IEEE

presents basic concepts and work related with musical sequences comparisons based on the edit distance are presented. This is then followed in Section 4 by a presentation of the heuristics to be used for marking musical dictation and a discussion of how they can be turned into an appropriate edit distance. Section 5 presents the architecture of the musical dictation marking tool which we are currently developing, while Section 6 concludes and presents future work.

2 REPRESENTING MUSICAL DICTATIONS USING XML

2.1 Key Characteristics of Musical Dictations

In a musical dictation exercise, a music teacher plays a music piece, one fragment at a time; each fragment is played several times (typically, three or four), as the piece is generally not known to the students. The student, by listening to the various fragments, must then recognize and write down the piece in standard Western music notation.

Basic music dictations are generally one voice dictation, that is, any given dictation consists of a sequence of individual notes for a single voice, each note having a specific *pitch* and *duration*. The pitch comes from a *finite* alphabet (*do*, *re*, *mi*, ...) and the duration is some fraction of the *whole*. A musical fragment can thus be represented as a sequence of pairs (p, d), with $p \in \{do, re, mi, \ldots\}$ and $d \in \{\frac{1}{16}, \frac{2}{16}, \ldots, 1\}$.[1]
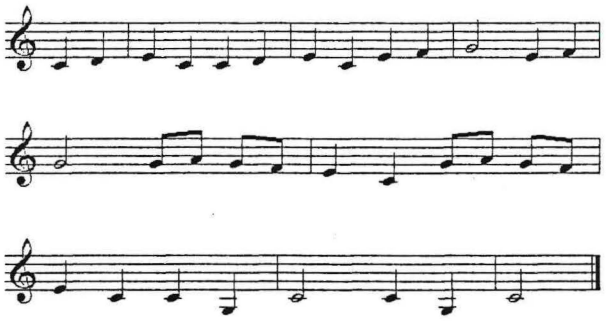


Figure 1. Musical score of *Frère Jacques*

The errors made by students while doing musical dictation exercises typically fall into a small number of categories [Beaudet, personal communication], for instance, errors regarding pitch (e.g., wrong note, missing or additional note), or errors regarding duration (e.g., wrong duration, split vs. combined note, wrong rhythm). For example, Figure 1 presents the correct score for the well-known French song "*Frère Jacques*", while Figure 2 illustrates a few typical mistakes that could have been made by a student.
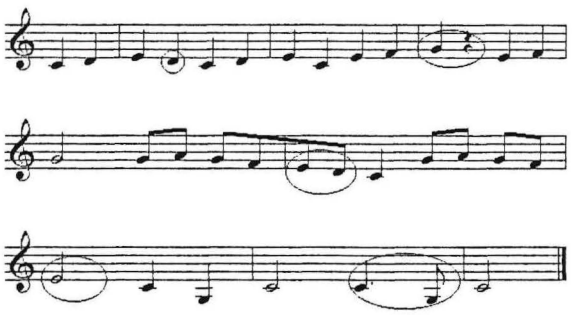


Figure 2. Typical errors for *Frère Jacques*

2.2 Symbolic Representation of Music Notation and MusicXML

In order to make the kind of comparative analysis required for marking musical dictations, an appropriate *symbolic* representation of music must be used. More precisely, both the exact pitch and duration of each note must be known, as well as the exact sequencing of the notes, and this information must be represented in a format that can be put in direct correspondence with standard Western music notation, with staff, key, notes, etc. A representation such as MIDI [23], thus, would not be appropriate: MIDI (Musical Instrument Digital Interface) is more of an exchange protocol for electronic instruments (which even attempts to describe details such as touch, attacks, nuances, etc.) than a purely symbolic notation.

Various symbolic representations for music scores have been proposed. SMDL [21] (Standard Music Description Language) is an attempt to define a general-purpose specification for music based on SGML (Standard Generalized Markup Language). However, due to its complexity, lack of reference manual and long development time, other simpler models were developed, with the result that, to our knowledge, no implementation was ever provided for SMDL. NIFF [17, 21] (Notation Interchange File Format) seems to be an interesting choice for our tool since it is "designed to allow the interchange of music notation data between and among music notation editing and publishing programs and music scanning programs." [17, p. 491] However, its adoption by industry has been very limited. The Humdrum format [11, 21] is also interesting since its goal is

[1] Note that a rest can simply be represented as a pair with a null pitch component and an appropriate duration.

to facilitate the implementation of analysis algorithms and is designed to allow multiple representations of musical fragments. Among these representations is the Kern Code which is designed for traditional Western musical notation. A Unix-based software development kit is available [17].

One important issue in the design of our tool is to eventually develop a web-based tutorial application that would include our program. A representation written in XML would be interesting and would allow us to develop a platform-independent tool with Java.

Over the recent years, interest has grown in using XML for representing various symbolic concepts. A number of XML applications have been developed in order to represent music notation, for example, 4ML [13], MML [22], and MusicXML [8]. Among these, MusicXML seems particularly interesting since it clearly aims at representing faithfully all the notions encountered in Western music notation. MusicXML is based in part on the Humdrum Kern format and describes most of its elements. Thus, it allows for the representation of all key notions, including alterations, dotted notes, triplets, lyrics, etc., as well as embodying a clean hierarchical approach (viz., score, part, measure, note). Some of the other notations appear less *symbolic* in that they seem to represent a kind of compromise between a purely symbolic notation and a digital one such as MIDI. MusicXML, at the time of this writing, also seems to be in a more stable specification state. Furthermore, a special plug-in for Finale [4], a music notation software with a graphical user interface, has recently been made available, making it possible to generate MusicXML files from music scores encoded with Finale.

With the various tools now available for XML and Java [6], using an XML representation with an appropriate DTD (Document Type Definition) makes it possible to automatically generate a parser which can read an XML representation of a music piece and can then generate a corresponding DOM (Document Object Model) object representing this music piece. This DOM object can then be manipulated using DOM-related operations, and appropriate analyses and transformations can be performed. This topic will be discussed in further detail in Section 5.

3 MUSICAL SEQUENCE COMPARISON

3.1 Detection of Similarities and the Edit Distance

Detecting similarities between two musical fragments has many applications. For instance, this can be used for identifying and retrieving musical pieces, as done in *query-by-humming* systems or in digital libraries of music [15, 2, 7, 19, 10]. Similarity detection tools can also be used for copyright infringement detection or musicological analysis [20, 12].

Given the context of our project, where a symbolic representation is a definite requirement, using a similarity measurement based on the edit distance between two sequences of notes is quite natural.

The edit distance is often used to compute a similarity metric between strings. Various forms of string edit distance have been used in various areas such as string and text analysis, protein and genome matching in bio-computing and signal processing [9, 18].

```
surgery
survery     -- Substitute g by v
survey      -- Suppress r
surveys     -- Insert s
```

Figure 3. Edit distance between "`surgery`" and "`surveys`"

The key idea behind the edit distance is to determine the minimum number of basic operations (e.g., insertions, deletions, substitutions) that can be applied to the first string in order to obtain the second. An example, presented in Figure 3, illustrates how the word "`surgery`" can be transformed into "`surveys`" using a minimal number of operations.

The general strategy for sequence comparison and edit distance computation uses dynamic programming. The cost function $C(i,j)$, which has to be minimized, can be defined as follows, where $\delta_{sup}, \delta_{ins}$ and δ_{subs} represent, respectively, the cost to *sup*press, *ins*ert, or *subs*titute a character:

$$
\begin{aligned}
C(0,0) &= 0 \\
C(i,0) &= C(i-1,0) + \delta_{sup}(\mathtt{A}[i]) \\
C(0,j) &= C(0,j-1) + \delta_{ins}(\mathtt{B}[j]) \\
C(i,j) &= \min \begin{cases} C(i-1,j-1) + \delta_{subs}(\mathtt{A}[i],\mathtt{B}[j]) \\ C(i-1,j) + \delta_{sup}(\mathtt{A}[i]) \\ C(i,j-1) + \delta_{ins}(\mathtt{B}[j]) \end{cases}
\end{aligned}
$$

More precisely, $C(i,j)$ represents the cost to go from the string $\mathtt{A}[1..i]$ to the string $\mathtt{B}[1..j]$, and the distance is the minimum cost of the operations required to make this transformation. Note that, for basic string operations, each operation is of unit cost, but it can be higher for other problems.

The asymptotic cost of this algorithm for sequences of length n is $O(n^2)$. This remains quadratic even when taking into account the special consolidation and fragmentation operations introduced for musical sequences [16], described below. For the introductory-level musical dictations we want to analyze, such a cost is clearly reasonable.

3.2 Edit Distance for Musical Sequences

Much research has been done on defining appropriate edit distances for musical fragments [16, 20, 7, 14, 1, 5]. However, to our knowledge, the present paper is the first one that tries to apply and adapt this algorithm to the specific problem of marking musical dictations.

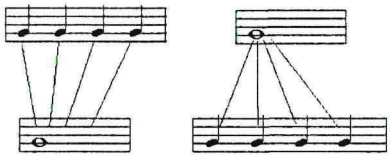


Figure 4. The consolidation and fragmentation operations of Mongeau and Sankoff [16]

Seminal work in the field of musical sequences comparison was done by Mongeau and Sankoff [16]. They used a weighted distance measure in which operations are assigned weights based on the harmonic distance between the notes and their relative *consonance*. Distinct weights are also attributed depending on whether the edit operation applies to pitch or tempo. Another interesting feature of their work is the introduction of special *consolidation* and *fragmentation* operations, as illustrated in Figure 4. Such operations are particularly interesting for our work, since these may correspond to typical student errors, where students correctly identify the pitch but do not clearly recall the exact rhythm. Consolidation would also apply when a student cannot recall a fragment and replaces it with rests or long note values.

In the context of music retrieval, where neither the exact base tonality nor the fine details of the rhythm are crucial to recognize a music piece, Lemström and Ukkonen [14] examine how the edit distance performs under tonality transposition. Thus, instead of comparing sequences of pitches, they manipulate sequences of interval differences (the distance between two consecutive notes) and conclude that an equivalent result could be obtained simply by using an appropriate distance and cost function.

Recently, Crochemore *et al.* [5] and Aucouturier and Sandler [1] have also presented work related with music retrieval. Both works deal with polyphonic melodies and do not deal directly with the specific notes and scores, working instead on the *texture* of the piece in order to recognize it under various interpretations.

Given the specific characteristics of our problem domain, the marking of musical dictations, the basic ideas introduced by Mongeau and Sankoff [16] are the most appropriate, so in the next section, we discuss how they can be adapted to our problem.

4 APPLYING THE EDIT DISTANCE TO THE MARKING OF MUSICAL DICTATIONS

4.1 A Heuristic for Marking Musical Dictations

In order to better understand how musical dictations are marked, we had a number of meetings with Ms. Luce Beaudet, a professor at the Faculty of Music (Université de Montréal) who is responsible for the music students' ear training.

Ms. Beaudet has pioneered an approach [3] to ear and solfège training where the emphasis is on understanding and analyzing a music piece based on the basic rules of harmony and common practice music. Thus, she proposes more of a macro-view to such training, where the students learn to recognize the key *patterns*, instead of simply being trained in *interval recognition* (identifying the distance between the current note and the previous one).

In basic ear training, the goal is to recognize the key patterns of tonal music, and so the rhythm is generally simple — more advanced courses put more emphasis on complex rhythm as well as atonal music. Thus, the heuristic used to mark musical dictations is that pitch errors are considered major (since they do impact the melody), whereas tempo errors are considered minor. Ms. Beaudet's approach to marking can be formalized as described below.

Let M_{pitch} and M_{tempo} be the contribution to the mark attributed to pitch errors and tempo errors, such that $M_{pitch} + M_{tempo} = 100$ (100 being the maximal mark, and with $M_{tempo} << M_{pitch}$). Let N_{notes} be the total number of notes appearing in the musical dictation and N_{beats} be the total number of beats. The weights W_{pitch} and W_{tempo} associated with each kind of error is then defined as follows:

$$W_{pitch} = M_{pitch}/N_{notes}$$
$$W_{tempo} = M_{tempo}/N_{beats}$$

Typical values used by Ms. Beaudet in her introductory classes are $M_{pitch} = 88$ and $M_{tempo} = 12$. For example, a student who makes 10 pitch errors and 3 tempo errors in the "*Frère Jacques*" musical dictation (with 32 notes and 32 beats [2]) would get the following mark:

Pitch errors	$10 * (88/32)$	=	27.5
Tempo errors	$5 * (12/32)$	=	1.9
Total errors		=	29.4
Final mark		=	70.6

In our heuristic, the cost of a pitch error is constant and does not depend on the value of the note, so we define it as

[2] In general, N_{notes} and N_{beats} are distinct, although in this particular example they happen to be the same.

$$
\begin{aligned}
C(0,0) &= 0 \\
C(i,0) &= C(i-1,0) + \delta_{sup}(\mathtt{A}[i]) \\
C(0,j) &= C(0,j-1) + \delta_{ins}(\mathtt{B}[j]) \\
C(i,j) &= \min \begin{cases}
C(i-1,j-1) + \delta_{subs}(\mathtt{A}[i],\mathtt{B}[j]) \\
C(i-1,j) + \delta_{sup}(\mathtt{A}[i]) \\
C(i,j-1) + \delta_{ins}(\mathtt{B}[j]) \\
C(i-k,j-1) + \delta_{cons}(\mathtt{A}[i-k+1..i],\mathtt{B}[j]), 2 \leq k \leq i \\
C(i-1,j-k) + \delta_{frag}(\mathtt{A}[i],\mathtt{B}[j-k+1..j]), 2 \leq k \leq j
\end{cases}
\end{aligned}
$$

Figure 5. Equations for edit distance (adapted from [16])

follows:

$$C_{pitch} = W_{pitch}$$

Rhythmic errors are computed on a beat basis. However, our sequence representation of a musical fragment, as described in Section 2.1, is based on notes instead of beats. Consequently, the duration of each note must be expressed in terms of the number of beats. Let b be the beat unit of a musical dictation, that is, the duration of one beat. Then, we define the cost of a tempo error for a note as follows:

$$C_{tempo}(p,d) = \frac{d}{b} * W_{tempo}$$

Then let E_{pitch} be the number of pitch errors made by a student. Let E_{tempo} be the number of notes containing rhythmic errors and $\{(p_1,d_1),(p_2,d_2),\ldots,(p_{E_{tempo}},d_{E_{tempo}})\}$ be the collection of these notes (more precisely, a multiset). The total penalty associated with the pitch and tempo errors made by the student is the following:

$$E_{pitch} * C_{pitch} + \left(\sum_{i=1}^{E_{tempo}} C_{tempo}(p_i,d_i) \right)$$

The final mark attributed to S's dictation is then:

$$100 - \left(E_{pitch} * C_{pitch} + \left(\sum_{i=1}^{E_{tempo}} C_{tempo}(p_i,d_i) \right) \right)$$

4.2 Turning the Marking Heuristic Into an Edit Distance

To transform the marking heuristic described above into an appropriate edit distance, we adapt the edit distance initially proposed by Mongeau and Sankoff [16]. The basic set of equations for the edit distance is presented in Figure 5, where the weights (the δ functions) have been modified for the marking heuristic as described in the following.

First, let Eq_p be a function that returns 1 when two pitches are equal and 0 otherwise, and similarly for Eq_d, which returns 1 when two durations are equal.

The cost of both $\delta_{sup}(\mathtt{A}[i])$ and $\delta_{ins}(\mathtt{A}[i])$ is the following:

$$C_{pitch} + C_{tempo}(\mathtt{A}[i])$$

The cost of $\delta_{subs}(\mathtt{A}[i],\mathtt{B}[j])$ is defined as follows:

$$Eq_p(\mathtt{A}[i],\mathtt{B}[j]) * C_{pitch} + Eq_d(\mathtt{A}[i],\mathtt{B}[j]) * C_{tempo}(\mathtt{B}[j])$$

We also introduce two cost functions associated with consolidation and fragmentation operations: δ_{cons} and δ_{frag} represent the cost, respectively, to replace a sequence of notes with a single note, and to replace one note with a sequence of notes. We only define the consolidation operation's cost in more detail, since the other is symmetric.

In order to replace a group of short notes with one long note, both sides need to have the same duration (i.e., the duration of the long note must be equal to the total duration of the group of short notes). When this is the case, the cost $\delta_{cons}(\mathtt{A}[i-k+1..i],\mathtt{B}[j])$ can be defined as follows:

$$\left(\sum_{l=i-k+1}^{i} Eq_p(\mathtt{A}[l],\mathtt{B}[j]) * C_{pitch} \right) + C_{tempo}(\mathtt{B}[j])$$

If the durations are not equal, then the cost function simply returns an infinite value, in order to preclude the consolidation/fragmentation operation from being selected.

5 A TOOL FOR MARKING MUSICAL DICTATION

Figure 6 presents the overall architecture of the musical dictation marking tool we are currently developing. We assume that the input to the tool is a pair of MusicXML files representing the musical dictation written by the professor and the student's answer. These XML files can be obtained in different ways, for instance, the Finale tool [4] now has an option to save a music score into MusicXML format.

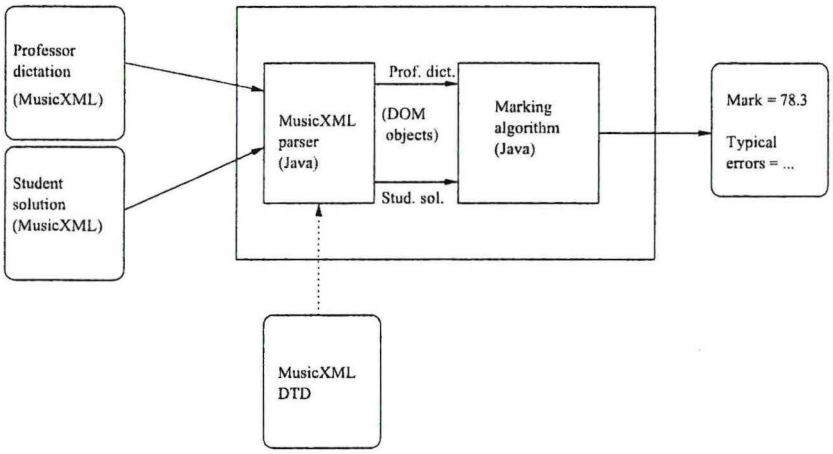


Figure 6. General architecture of our musical dictation marking tool

From a DTD for MusicXML, it is possible to generate, using an appropriate parser generation tool [6], a parser (written in Java) which can read and parse a MusicXML input file, and then construct a DOM object — essentially a syntax tree — representing the professor's dictation as well as the student's solution.

In turn, these DOM objects are analyzed and transformed into a simpler sequence representation of the musical fragments appropriate for the edit distance computation algorithm. This algorithm, still under implementation, is also being developed in Java. Based on the edit distance, the student's mark can then be computed and, from the matrix obtained through the dynamic programming implementation of the edit distance computation, the key errors made by the student can be identified.

It is important to stress that the marking strategy is indeed a *heuristic*, which means some tuning-up might be required — for example, by varying the weights and costs (see Section 4.2) associated with the various types of errors. Of course, the marks computed by the tool will also have to be validated. First, this will be done by using simple test cases (e.g., variations on *Frère Jacques*). Then, we intend to use real dictations written by students in an introductory solfège course taught at the Faculty of Music at Université de Montréal and compare the marks produced by our tool with those given by our domain expert.

6 CONCLUSION AND FUTURE WORK

In this paper, we have described a software tool to help music teachers mark their students' music dictation. Given a standard music notation tool that generates an appropriate XML representation, we are able to parse and transform this input into sequences for which an edit distance can be computed in order to give a mark.

As mentioned earlier, our short-term goal is to be able to compute fair marks for students' musical dictations based on a marking heuristic implemented using an appropriately tuned edit distance. The edit distance implementation strategy (dynamic programming) will also make it possible to provide some feedback to the students on the typical errors they have made.

A longer-term goal is to develop a more general tool and integrate it within an intelligent tutoring system. In such a context, the tool can be adapted to the students' level; for example, musical dictations of varying degrees of difficulty could be proposed to the students based on their identified weaknesses.

ACKNOWLEDGMENTS

Special thanks to Prof. Chuck Wallace (Michigan Technical University, MI) for his detailed comments on the initial version of this paper that (hopefully) helped improve the English-writing. Support for Ms. Champagne's work was provided by an NSERC's (Natural Sciences and Engineering Research Council of Canada) student scholarship.

REFERENCES

1. J.-J. Aucouturier and M. Sandler. Using long-term structure to retrieve music: Representation and matching. In *ISMIR*, June 2001.
2. D. Bainbridge and al. Towards a digital library of popular music. In E. Fox and N. Rowe, editors, *Digital Libraries*, pages 161–169, 1999.
3. L. Beaudet. Préalable à l'acte d'audition tonale. Technical report, Université de Montréal, Faculté de musique, September 1980.

4. Coda Music. Finale 2002. www.codamusic.com/coda/fin2002.asp, 2002.
5. M. Crochemore and al. Approximate string matching in musical sequences. In M. Baliik and M. Simanek, editors, *PSC'2001, Prague Stringoly Club*. Czech Technical University of Prague, 2001.
6. M. Daconta and A. Sagnich. *XML Development with Java 2*. Sams Publishing, 2000.
7. C. Francu and C. Nevill-Manning. Distance metrics and indexing strategies for a digital library of popular music. In *IEEE International Conference on Multimedia and Expo (ICME)*, July 2000.
8. M. Good. Music XML. www.recordare.com, 2001.
9. D. Gusfield. *Algorithms on strings, trees, and sequences*. Cambridge University Press, 1997.
10. G. Haus and E. Pollastri. An audio front end for query-by-humming systems. In *2nd International Symposium on Music Information Retreival (ISMIR'01)*, Bloomington, Illinois, October 2001.
11. D. Huron. *The Humdrum Toolkit: Reference Manual*. Center for Computer Assisted Research in the Humanities, 1991.
12. A. Kornstädt. The *JRing* system for computer-assisted musicological analysis. In *2nd International Symposium on Music Information Retreival (ISMIR'01)*, Bloomington, Illinois, October 2001.
13. M. Legh and L. Montgomery. 4ML. www.4ml.org, 2001.
14. K. Lemström and E. Ukkonen. Including interval encoding into edit distance based music comparison and retrieval. In *AISB'2000 Symposium on Creative and Cultural Aspects and Applications of AI and Cognitive Science*, pages 53–60, Birmingham, United Kingkom, 2000.
15. McNab and al. The New Zealand digital library MELody inDEX. *D-Lib Magazine*, 1997.
16. M. Mongeau and D. Sankoff. Comparison of musical sequences. *Computers and the Humanities*, 24(3):161–175, 1990.
17. S. Mounce. NIFF page. www.student.brad.ac.uk/srmounce/niff.html, 1997.
18. G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, March 2001.
19. D. Ó Maidín and M. Fernström. The best of two worlds: retrieving and browsing. In *Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-00), Verona, Italy*, December 2000.
20. K. Orpen and D. Huron. Measurement of similarity in music: A quantitative approach for non-parametric representations. *Computers in Music Research*, 4:1–44, 1992.
21. E. Selfridge-Field, editor. *Beyound (MIDI) – The Handbook of Musical Codes*. MIT Press, 1997.
22. J. Steyn. Music markup language (MML). www.mmlxml.org, 2001.
23. E. Tholomé. *MIDI à votre portée*. Editions Radio, 1990.

A Proposal for a Regular Grammar To Parse Jazz Chords Notation

Denis L. Baggi
Institute CIMSI of SUPSI
Scuola Universitaria Professionale della Svizzera Italiana
Galleria 2
CH-6928 Manno, Switzerland
+41 91 610 89 60
dbaggi@cimsi.cim.ch

ABSTRACT

A piece of improvised Jazz is identified by its harmonic structure of *changes*, placed in the *harmonic grid.* For each chord, or *change*, a particular notation is used which is well known by Jazz musicians, and appears also in other contexts, such as guitar charts.

While there are minor differences and inconsistencies in this notation, the meaning is always clear to practitioner of the art. Purpose of this work is to propose a simple grammar, or standard, which, while formalizing the notation, does not impose constraints, allows extension and easy recognition of unknown symbols, new symbol combinations and definitions.

Keywords

Jazz Changes, Harmonic Grid, Formal Grammars, Regular Grammars.

1 INTRODUCTION

In improvised Jazz, *melody* has a relative importance. In many pieces, it is often used to *expose the theme* at the beginning of a piece and to close it, as in a pyramidal form where the climax is represented by the improvised paraphrases. Some masterpieces are well known in which the original melody of the tune is hardly played: *Body and Soul* by Coleman Hawkins, *These Foolish Things* by Lester Young, *Embraceable You* by Charlie Parker. Moreover, more often than not the melody is supposed to be known and thus hardly written.

The opposite is true for the harmonic structure of the piece, consisting of a sequence of chords or *changes*, which identify, in a relatively precise way, the musical elements a Jazz musician can use in his improvised paraphrases. It is therefore compulsory, for the "composer" of a new piece, to provide the changes structure to those musicians playing with him. Even though, often, improvising musicians do skip this stage and revert to well-known structures such as the *12-bar blues*, *rhythm changes, standards* of which the melody is often changed in an almost unrecognizable way, the fact remains that the structure of every piece can be annotated thanks to a representation of the changes.

2 THE HARMONIC GRID

This is the table that contains the notation for each change. Since most Jazz pieces consist of either 12 bars or 32 bars, the most popular grids are shaped in rows and columns with the patterns 3 x 4 and 4 x 8. **Figure 1** is the representation of an evolved 12-bar blues, *Blues for Alice,* and **Figure 2** is that of an AABA piece, *I'll Wait and Pray.*

The harmonic grid is not part of this proposal, hence only a short explanation will be given. Each square represents a measure or *bar* of a tempo with signature 4/4. The symbols for the change may occupy a whole square, in which case the chord lasts a whole bar or more if the symbol % is used subsequently. A "/" <slash> splits a bar in two if needed, and a "," <comma> separates chords falling on each beat – hence no change lasts less than a beat. For further details see [1].

3 THE NOTATION FOR A *CHANGE*

While there is some inconsistency and redundancy, a good approximation for chord notation can be given as follows:

I. the first symbol is the letter **A** to **G**, upper case (this is the only symbol that will always be present)
II. it can be followed by the symbols ♯ or ♭
III. then further letters may follow, such as **m**, **d**, **o**, **M**
IV. or numbers like **7, 9, 11, 13, 6, 5**
V. and characters like Δ, (,), +, ♯, -, ♭, **Ø.**

As in every musical notation, a composer may decide to violate rules, introduce new symbols and syntax [2]. Nevertheless, the set of above is a good starting point to explain the meanings:

- the letters **A** to **G**, possibly followed by ♯ or ♭, indicate the **root** of the chord, which is always a superposition of **thirds (major** and **minor)** above the root. A **single note** with no further symbols defines by default a *major triad* (major third and fifth). Hence **C** indicates the chord **c – e – g,** and **A♭** that with **a♭ – c – e♭**
- the symbols **-** and **m** indicate that the first third is flattened, as in a *minor chord.* Hence **D-** indicates **d – f – a.** Symbols **o** and **d** are used for diminished chords: **Cd** indicates **c – e♭ – g♭.** While **M7,** and also Δ, indicate a major seventh chord: **FΔ** has notes **f – a – c – e**
- numbers such as **7, 9, 11, 13** indicate added thirds on top of the original triad to reach that interval. Hence **G7** indicates **g – b – d – f, D9** has **d – f♯ – a – c – e, A13** has **a – c♯ – e – g – b – d – f♯**
- number **6** indicates the added sixth on top of a chord: **F6** is **f – a – c – d, Cm6** has **c – e♭ – g – a**
- number **5** is used only to indicate an alteration of the fifth, as in **C(-5)** with **c – e – g♭** and **E7(#5)** with **e – g♯ - b♯ - d.** Note that **parentheses** are used to increase readability
- lastly, diminished chords are of various types and can be indicated in various ways. **Cd7** is a *fully diminished* (diminished seventh) chord with notes **c – e♭ – g♭ – a,** while **CØ** is *half-diminished* (minor seventh) with **c – e♭ – g♭ – b♭** (same as **C-7(-5)**
- these symbols and groups can be combined, as in **C7(♯11)13,** a chord with **c – e – g – b♭ – d – f♯ – a,** as long as the combinations make musical sense to the author of the piece.

Once more, this description is possibly incomplete and inaccurate and had been adopted for the input syntax of a working program [3, 1]. References to Jazz changes notation can be found in the specialized literature [4, 5].

Lastly, notice that for the purpose of Jazz improvisation, and accompaniment such a guitar chord playing, this notation is clearer than that of classical music, such II → V → I, based on *degrees.* Because, to a player, a chord like, say, Dm7 is well defined and remains the same no matter the key and degree. While it is true that an improviser may have to define passing notes of the scale implied by the chord in function of the context, experience dictates the proper choice, regardless of the key. This because Jazz, although mostly tonal, works on harmonic progressions independently of the key *(Sweet Georgia Brown, How High the Moon)* or with tunes for which the key can hardly be defined *(Ruby My Dear, Giant Steps).* The classic notation would be impossible, unless a complicated mechanism were provided to indicate changes of key, and even that would often not work.

4 REGULAR GRAMMARS

In Chomsky's classification, *regular* or *Type 3* grammars are defined as those who can be reduced to, or shown to be equivalent to, grammars with the following type of production:

$$A \rightarrow bB$$
$$A \rightarrow b$$

[6, 7]. It is also possible to see that such grammars parse (or generate) *regular expressions* as strings; that, given a regular expression, there

is an algorithm constructing a *finite state automaton* accepting it, and that *non-deterministic finite automata* can be converted to *deterministic* ones. And finally, that – still with an algorithm – finite automata can be realized with a *logic circuit* consisting only of *Boolean functions (AND, OR, NOT)* and of *delay elements* (e.g., *FLIP-FLOPs).* Thus all representations: *Type 3 or regular grammars, regular expressions, finite state machines, and circuits with gates and flip-flops* are *isomorphic.*

Moreover, regular grammars are a special case of *context-free grammars*, namely, of the type, in BNF form, used to describe computer languages such as ALGOL and C (which are equivalent to expressions accepted by *pushdown automata*, those with a stack, which finite state machines do not possess). In particular, it can be proved that such a regular language can be seen as a special case of the language defined by a LR(1) grammar, where an LR(k) grammar is one for which there is only one foreseeable derivation for a string up to at most *k* symbols – hence, the look-ahead for parsing does not extend beyond *k* symbols. Compiler compilers, such as *yacc* under the UNIX operating system [8], are designed for LR(1) grammars, hence such programs could be used to easily implement a grammar (even though *scanning algorithms* such as a *lexical analyzer* could be used to the same effect, except for the formalism).

5 THE PROPOSAL

The work proposed here aims to:

- define in a formal way a grammar to parse the notation of Jazz changes, extending it as much possible to all the examples that can be found in Jazz literature
- include a simple mechanism to allow a user to extend, or personalize, the grammar, to include new symbols or structures
- realize together with the grammar a *chord compiler* which could be used to: define chord notation; allow a user to check his composition; build an evolving standard to be commonly used for the notation of Jazz harmonic grids, without imposing constraints on the composer.

6 CONCLUSIONS

This standard for Jazz Chords Notation will consist of a publicly available program that could be used by a Jazz composer and performer as a common reference, without any coercion on the use of prefabricated rules and with access to all details. While we expect it to be beneficial to all, it will particularly be useful to beginners and students of Jazz harmony.

ACKNOWLEDGEMENTS

This work is part of IMS project **A MUSICAL APPLICATION STANDARD USING THE XML LANGUAGE FOR INTELLIGENT MANUFACTURING OF MUSIC FOR CDs, DVDs, WEB**, abstract endorsed on August 12, 2002 (see www.ims.org).

REFERENCES

1. Baggi, Denis, **Capire il Jazz – le strutture dello Swing,** Quaderni dell'Istituto CIM della Svizzera Italiana, ISBM 88-900298-2-X, 2001.
2. Cage, John, **Notations,** Something Else Press, ISBN 0-87110-063-0, 1969..
3. Baggi, Denis, **NeurSwing: An Intelligent Workbench for the Investigation of Swing in Jazz,** in: **Readings in Computer-Generated Music**, IEEE Computer Society Press, Los Alamitos, CA, ISBN 0-8186-2747-6, 1992.
4. Sabatella, Marc, **A Jazz Improvisation Primer - A Whole Approach To Jazz Improvisation**, ADG Productions, Lawndale, California, 1996. Also available on the WWW at http://www.outsideshore.com.
5. Russo, William, **Composing for the Jazz Orchestra,** University of Chicago Press, Chicago 1961. (Note: this is an example out of many).
6. Hopcroft, John E. and Ullman, Jeffrey D., **Formal Languages and Their Relation to Automata,** Addison-Wesley, ISBN 0-201-02983-9, 1969.
7. Hopcroft, John E. and Ullman, Jeffrey D., **Introduction to Automata Theory, Languages and Computation,** Addison-Wesley, ISBN 0-201-02988-X, 1979.
8. **YACC – Yet Another Compiler Compiler,** Documentation of 4.0 to 4.3 and 2.8 to 2.11 BSD Unix, Berkeley Source Distribution Service, University of California, Berkeley, 1983-1990. See also **LEX, A Lexical Analyzer.**

B♭M7	G0	Am7	D♭7
Gm7	C7	FM7	Gm7

Figure 1. Harmonic grid of *Blues for Alice* by Charlie Parker.

Fm7/ B♭7	B♭m7/ Fm7,E7	E♭M7/ D7	G♭(+5)/ G♭m7,Gm7	Fm7/B♭7	Fm7/E7	E♭M7/ D(+5)	D♭7,C7/ G♭7
Fm7/B♭7	B♭m7/ Fm7,E7	E♭M7/ B♭(+5)	G♭(+5)/ G♭m7	B♭m7/A7	EM7/ G♭m7,B♭7	Fm7/ E7	E♭M7/ %,G7
B♭m7/A7	EM7/ G♭m7,B♭7	Fm7/E7	E♭M7/ %,G7	Cm7/ F♯m7,B7	EM7/ G7	Fm7/ D♭m7	G7,D♭m7/ C7
Fm7/B♭7	B♭m7/ Fm7,E7	E♭M7/ D7	G♭(+5) /G♭m7	Fm7/ B♭7	Gm7/ C7	Fm7/ B♭7,E7	E♭M7

Figure 2. Harmonic grid of *I'll Wait and Pray,* by Gerry Valentine, reworked by John Coltrane.

Framework for a music markup language

Jacques Steyn
Consultant
PO Box 14097
Hatfield 0028
South Africa
+27 72 129 4740
jacques@musicmarkup.info

ABSTRACT

Objects and processes of music that would be marked with a markup language need to be demarcated before a markup language can be designed. This paper investigates issues to be considered for the design of an XML-based general music markup language. Most present efforts focus on CWN (Common Western Notation), yet that system addresses only a fraction of the domain of music. It is argued that a general music markup language should consider more than just CWN. A framework for such a comprehensive general music markup language is proposed. Such a general markup language should consist of modules that could be appended to core modules on a needs basis.

Keywords
Music Markup Language, music modules, music processes, music objects, XML

1 INTRODUCTION

The use of markup languages exploded after the introduction of the World Wide Web, particularly HTML, which is a very simple application of SGML (Standard General Markup Language). The latter is obviously much more powerful than HTML in terms of power of description, but it could be argued that, based on sheer volume, HTML is more powerful in terms of its use in practical applications. Similarly, markup languages for music such as HyTime and SDML are very powerful and for the purpose of comparison can be likened to SGML. What is lacking is an HTML-like music markup language; one that is as simple, yet powerful enough.

Creating such a language has become possible after the introduction of XML, but there is as yet no widely accepted language for music, and those that have been introduced focus only on small and particular subsets of CWN (Common Western Notation). Known attempts of XML-based music markup languages are [13]: 4ML (Leo Montgomery), FlowML (Bert Schiettecatte), MusicML (Jeroen van Rotterdam), MusiXML (Gerd Castan), and MusicXML (Michael Good), all of which focus on subsets of CWN. ChordML (Gustavo Frederico) focuses on simple lyrics and chords of music. MML (Jacques Steyn) is the only known attempt to address music objects and events in general.

In this paper I will investigate the possible scope of music objects and processes that need to be considered for a comprehensive or general music markup language that is XML-based. To begin with, I propose the following basic requirements for such a general music markup language.

2 REQUIREMENTS FOR A MUSIC MARKUP LANGUAGE

A general music markup language should

- conform to XML requirements as published by the W3C
- use common English music terminology for element and attribute names
- address intrinsic as well as extrinsic music objects and events
- be simple: a user with basic music knowledge should be able to create applications using a text editor
- be modular: each distinguishable main component of music should be addressed by its own module
- address universal music (such as all possible tuning systems or definitions of music)

3 MARKUP LANGUAGES

Markup languages such as HyTime and SDML, that approach marking music in general, are not XML-based. One possibility would be to translate HyTime and SDML into an XML conforming format, and we would have an XML-based general music markup language. Let us briefly consider these languages.

1. HyTime

Hypermedia/Time-based Structuring Language [9].

HyTime and SMDL originated in July, 1986 on the initiative of Charles F. Goldfarb, one of the fathers of SGML. HyTime is an architecture for the description of music structure and hypermedia. The draft document was released on 19 June 1989, and finally proposed as a standard in January 1991. HyTime became a standard on November 1, 1992 as ISO/IEC 10744. Although HyTime was originally developed for music, its concepts proved to be so powerful that it has been applied to many diverse fields, first by the Association of American Publishers for the representation of electronic documents. Braille devices also make use of this standard.

0-7695-1864-8/02 $17.00 * 2002 IEEE

HyTime is a very complex language and does not meet our requirement of simplicity, and it obviously is not XML-based. Despite its power it has never reached widespread acceptance. In fact, the originators themselves chose to develop a subset of HyTime to specifically focus on music. That subset is known as SMDL. Following the HyTime route would thus not result in a productive solution for our purposes.

2. SMDL

Standard Music Description Language [15].

SMDL was proposed as a standard in January 1991. SMDL is a HyTime application and conforms to ISO 8897, the Standard Generalized Markup Language (SGML). SMDL focuses on application-neutral archival storage of music and has not really extended to other kinds of applications. SMDL was created to be machine-readable and not really human-friendly. It thus also does not meet our requirements to be human-friendly, and its focus is on archiving, not addressing real-time performance issues.

SDML is an abstract language that introduces a host of technical terms used for tags which are unknown to the average musically informed user. How many users, including the more serious, would know the meaning of terms such as *cantus, gamut, nominal pitch, fictum adjustment, music ficta gamut,* and others? A new markup language would by nature introduce new terms, but I maintain that the highly abstract and technical nature of SDML terminology prevents it from being widely implemented for popular use. As mentioned at the outset, we need a music markup language that is as relatively simple as HTML, so SDML will also not do for our purposes.

3. SMIL

Synchronized Multimedia Integration Language [17].

Presently the only standard XML-based language that addresses non-notation aspects of music is SMIL, which focuses on multimedia content. SMIL was designed to mark multimedia objects in terms of time, layout and hyperlinks within SMIL documents. It does not address the intrinsic characteristics of music objects at all, and although SMIL can be used to refer to music objects extrinsically, it does not explicitly address issues such as a playlist. It would be possible to hack SMIL to create a playlist by marking several clips with switches, but that would not be a very elegant solution.

So HyTime, SDML and SMIL do not provide a solution for a general XML-based music markup language as envisaged here, and existing XML-based markup systems focus only on subsets of CWN.

* * *

Before a music markup language can be envisaged, the set of objects which it will describe (or mark) need to be defined and clarified. Answering the question: "Which music objects must be marked?" will determine what the strengths and weaknesses of a music markup language would be. The object "music" thus needs to be defined.

A working definition for the purpose of this paper is that music is technology based. This definition includes the spectrum ranging from the instruments (i.e. musical instruments) used to perform music, to the equipment used to record or reproduce music. The only non-technology music there is are the sounds of nature (including the human voice). From this perspective instruments to produce music sounds form the core of music production, while equipment used to manipulate, record or reproduce music are peripheral. To put it simply, instrument technology is used to create performed music, while production technology is used to manipulate that performed music. The first level of music concerns the fundamental focus of music in an abstract manner, and the second level concerns that which manipulates performed music, the distal aspects.

4 INTRINSIC FEATURES OF MUSIC

I distinguish between two aspects of the fundamental focus (intrinsic features) of music: the *core* of music and *peripheral aspects* of music.

1. Music core

The music core consists of those features without which music sound is impossible. They are thus intrinsic music objects and processes that concern, on an abstract level, concepts such as notes, duration, intervals, frequency, pitch and similar related aspects for which certain well-established technical terms are used in musicology. These intrinsic objects and processes would form the core modules in a modular-based application.

No music could exist without these intrinsic objects and processes. But the same intrinsic aspects can be manipulated by peripheral aspects in various ways to produce different results.

2. Peripheral aspects of music

The music core can be manipulated to achieve different results. Peripheral aspects concern the "things" done to the core objects and processes. Different textures (i.e. peripheral matters) can be added to the core that will result in different sounds. An abstract note can be executed on a saxophone or piano, resulting in the paradoxical same but different thing. For example, an abstract *A* note (which is describable and markable in terms of frequency and time, i.e. the core) can be given different expressions by adding either saxophone or piano textures to the core.

5 EXTRINSIC MUSIC OBJECTS AND PROCESSES

The intrinsic aspects of music can be further manipulated by additional technologies, which can be divided into two broad categories: technologies used for recording music and technologies used for performing music.

Performed music

The technologies I have in mind here are such as effects units, or triggering devices such as MIDI-enabled pads and mats. Certain control systems, such as mixers, are also classified here.

Control systems have been addressed to a limited extent by SMIL. The one limitation of SMIL in this respect is that it is inward looking (an intra-systemic markup language); it does not provide the possibility of marking the larger issues such as creating playlists, or metadata (such as for CD sleeves), which is addressed by MML to a limited extent. SMIL also does not address macro control systems such as mixers.

Recorded music

There are, very broadly speaking, two categories of recorded music: graphic recording and sound recording.

a. Graphic recording

Intrinsic music objects can be expressed in some or other writing or notation system, such as the piano roll format, or the CWN format. A composer would use some or other writing or notation system to pen a music score, the function of which is to contain suggestions as to its performance. Manuscripts are meant to be records and not changed by general users, who use them as guidelines for performance.

A music markup language used for marking such music would need to describe notation systems faithfully. As mentioned above, most XML-based attempts at a music markup language focus on this aspect of music, and specifically on CWN.

b. Sound recording

Expressed music can be recorded using electronic recording techniques such as used in recording studios. Graphic recording has been around for many centuries, while sound recording has been in existence only for about a century. In studio parlance, this category of music would be "canned music". The resulting product cannot be manipulated by general users, except for limited control such as volume, or a bit of EQ-ing. In other words, no intrinsic features of music can be manipulated after being recorded. Results of recorded music are distributed on CDs, DVDs and in various formats, such as MP3. A general music markup language should enable an end-user to create playlists and other methods of organization to select specific songs from the multitude of available songs in any format of choice. It should also be possible to select the ambiance (such as playing the song in a small room or large hall) as many popular home entertainment systems now allow. Apart from some limited effort in MML, there is no known attempt to address this aspect of music for markup purposes.

6 NOTATIONAL MARKUP LANGUAGE

A notation markup language is twice removed from the object of description. A notation system (once removed from the music object) abstractly describes some music micro-objects such as the relation between two frequencies. Implied here are aspects such as the tuning system chosen, the frequency chosen as fixed reference point (e.g. A440 or A442), what the exact duration of a note value (e.g. a quarter note) would be, and many more important features not addressed by notation systems. These aspects are not explicitly addressed in notation systems, but implicitly present. Translating a notation system into a markup system would thus be twice removed from the core. A first level of description is lacking.

A music markup language that is based on notation would be merely a system that translates the abstract notations into the terminology and syntax of the particular music markup language. This is easy to do: map symbols and signs to markup element and attribute names. This is done by creating a lookup table with notation terminology in one column, and corresponding music markup language terminology in the adjoining column. The only challenge here for an XML-based music markup language would be naming conventions, nesting issues and making decisions about which objects would be elements, and which would be attributes.

A music markup language focusing only on CWN cannot do justice to a large section of music. Consider the use of a score. A musician using typical CWN needs years of training to read and perform the score, and then for the purposes of a specific performance the conductor, music director, or band leader decides on the finer nuances for interpreting the score, either in an authoritarian manner or negotiated. No notation system captures all the possible information. A typical jazz score specifies the parameters within which the musicians will play. Chance music, such as AMM, explained by Prévost [19], reacts to incidents that occur totally by chance. No traditional score can predict what may happen. For such music, its notation can only be used after the fact. It should be obvious that notation systems do not address all these issues, consequently, a music markup language that addresses only the traditional score also will not address music comprehensively.

A system focusing only on notation has limited functionality, especially in the world of computing. It focuses on the graphic record of music, typically used for archival or reference purposes. Such a system was useful in the days prior to sound recording technologies, but really

limited today. What is needed is a music markup language that can be used to mark performed music, and consequently the intrinsic music objects and events. I maintain that such an approach must form the foundation of any attempt at creating a music markup language. If done successfully, it should be possible to attach any additional music features on a needs basis by adding their respective modules. The core should thus contain universal and abstract features of music and not be biased toward CWN. MML is an attempt to address this.

7 SOME COMPLEXITIES OF MUSIC

The kind of music markup language I have in mind will have to address some of the complexities of music. Here I take the following working definition of music: music is a function of frequency and time.

Music cannot be regarded just in terms of sounding frequencies. In an orchestra instruments may keep quiet for stretches in the piece. By following Zen philosophy, John Cage regarded silence, the absence of sound, as part of musical expression. In the MIDI environment declaring Note Off, implying a possible silence, is important. One important structural aspect of music for a markup language is then the presence or absence of sound. Such characteristics are based on a time continuum. Music perception also follows a timeline because of the psychological arrow of time in which an individual consciousness is entrenched. For a music markup language a time element must consequently be considered essential.

But "time" has many different meanings that need to be clarified. Apart from calendar time, which is metadata about the piece of music, there is the duration of the performed or recorded piece of music in which smaller time elements can be found. With respect to the smaller time units, the cyclical repetition of musical units can be determined on many different levels, ranging from music phrases, to smaller measures, traditionally called "bars", which in turn may be divided into yet smaller units. The duration of the piece of music can be mapped to absolute clock time in terms of hours, minutes, seconds and milliseconds. The smaller time units are mapped onto this "external" time. By extending or reducing external time, the tempo of a song may be changed.

For synchronization purposes computers need to refer to some or other clock. Linking music time to an external clock would thus be important for any general music markup language. But equally important are the relationships between smaller time units that are relative in the sense that they may be altered and result in different moods for the same song. The funeral music of New Orleans Jazz bands is often a slower version than their much quicker march counterparts when measured against external time. But so would be any band that plays a piece very slowly. What makes this particular funeral music unique is the changing of the smaller internal time relationships: changing the march into a swing.

Apart from time, music must also be defined in terms of frequency. A question that arises from this aspect is whether all possible frequencies can be regarded as music. The sounds of traffic noise, birds singing, hooters blowing, footsteps, engines idling, a lawnmower and so on are around us but do not become music unless focus is directed to them as such by the performer. In a performance art environment the lawnmower engine noise becomes a statement and part of the production, and thus "music" (and not recordable as music by CWN). The use of this sound is planned. If it is a chance-based performance the exact moment of the introduction of the lawn mower sound may not be planned, and hence not possible to indicate in a time-based notation system, yet the decision to include the sound is planned, and may be marked. For such a score the only aspect that can be marked is a description of the sound, and perhaps a random parameter of time. After the performance the sound can be fixed on a timeline. This suggests that time is not a more basic concept than frequency. But of course it would not be very useful to mark all the detail frequencies of a lawnmower, just as it would be impractical for most uses to mark all the physical frequency details of a note played by a a saxophone. Here it may be wise to follow the approach of CWN, which is a generic abstraction from all the possible detail. In this sense a music markup language could follow the concepts of CWN, but that does not mean a music markup language should be mapped to CWN or limited to that notation system.

A general music markup language that can handle all the mentioned complexities is needed. With MML I attempt to address such a music markup system and propose the following modules to handle such complexities in a structured manner.

8 MODULES OF A GENERAL MUSIC MARKUP LANGUAGE

The following modules are proposed for a general music markup language. Each module contains a set of elements and attributes. The core modules are the Frequency and Time Modules, which will apply to most applications of MML, but not always required. For example, in practice it should be possible to use only the Lyrics Module without reference to any music notes. The envisaged modules are:

1. Frequency module
2. Time module
3. Organization module
4. Texture module
5. Effects module
6. Performance module
7. Control module

8. Notation module
9. Lyrics module
10. MIDI module
11. Synthesizer module
12. more...

In this proposed model, the Frequency and Time Modules are the core modules as music could be conceptualized as a function of time and frequency. All intrinsic music objects and events are described in terms of these core modules. Other modules are added on a needs basis. For example, the specifics of CWN would be handled by the Notation Module which also would, as envisaged, handle any other of the available notation systems. Any music piece or song that is described in terms of the core modules can be processed in any number of directions, such as to be visually displayed by some or other notation system and which can be further manipulated by XSL or other methods, translated to be used by a MIDI system, or sent through an effects bank, and so forth.

The terms chosen for element and attribute names used in the modules come mainly from CWN, which provides a solid point of departure as these terms are commonly used in the world of music. However, in MML these terms have abstract meanings. In practice an MML note may be expressed as a CWN note, but it may also be expressed or executed in other ways. So CWN terms may map uni-directionally to MML terms, but an MML term may map multi-directionally to various expressions, one possibility which may be CWN.

I will now turn to the components of the modules. Deciding which objects or events should be elements or attributes depends on whether time or frequency is taken as basis. After trying out several options, I decided to take time along an x-axis, and frequency on the y-axis, just as in CWN. The implication of this is that some or other relative time notion has to be the basis on which frequencies are indicated. CWN provides a concept for this: the "bar". In MML, a "bar" may, or may not map to the CWN "bar", depending on the characteristics of a particular piece of music. The MML "bar" would thus be an element, while many of the frequency characteristics would be attributes associated with this element.

1. Frequency Module

In the Frequency Module the following components are distinguished: **cent**, **note**, **scale** and **tuning**.

The first two attributes (**cent**, **note**) concern aspects of the traditional concept of "note". The **scale** element concerns the small-scale relationships between notes, while **tuning** refers to large-scale relationships between all the notes within a particular system, as well as mapping that system to a frequency table.

In order to meet the basic Requirements 2 and 4, the terminology established by CWN is used. Although CWN does not describe the totality of music, it is a very useful system. However, in the model proposed here, the terms are abstracted and are not restricted to visual representations.

The term "cent" is not commonly used among ordinary musicians, but some term is required for finer frequency distinctions than provided by "note". It is thus unfortunately necessary to introduce this term.

In CWN the concept "note" is a graphic symbol on a score that suggests to the musician or researcher both a relative frequency and relative duration at the time of execution. This is the sense in which the term "note" will be used in the proposed model. The actual physical frequency value of a note will depend on the selected tuning system. The duration of the note is handled by the Time Module. By defining a note in this manner, it can be easily mapped to any notation system, or systems such as MIDI. In CWN all this additional information is tacit, but a markup language requires these aspects to be made explicit. A music markup language would thus always require an explicit statement about the tuning system used, and the mapping of notes within the tuning system to absolute frequencies. In practice there could be a default system so that only deviating systems need to be stated explicitly.

The implication of this is that a note marked as "C" has no meaning when decontextualized, yet it can be described in the markup language. It only gets meaning by being associated with a particular tuning system. When associated with a tuning system such as equal temperament, it would obviously map easily to CWN, but this abstract note may also map to different tuning systems, which may perhaps produce quite different results.

A **noteset** attribute is used for sets of notes that may recur in different frequency bands. It thus contains the Western tuning system concept of "octave", which is a recurring noteset at 8-note intervals. The selected tuning system will determine the meaning of the noteset used, so if **noteset** must have the meaning "octave", the Western tuning system must be selected. This general approach will thus allow a note marked as "C" to be used with any possible tuning system. To indicate the different recurring cycles in MML the number of the cycle precedes the note name. So when octaves are used, a "C" in octave 4 will be written as "4C".

Rests (which are usually expressed in terms of time in CWN) are merely the absence of frequency over a period of time. In other words, where a frequency is expected, but nothing is marked, the implication would be silence, and consequently, in terms of CWN, a rest. In a markup language rests can thus be handled in different ways. Following CWN they can be marked with explicit time values. Following a technology such as MIDI, note lengths may be assigned absolute length values (i.e. duration), and

whenever there is no duration, a rest is present by implication.

From an XML point of view the choice between these approaches is immaterial as an XML parser can in any case only process the markup text. To do anything musically useful with the markup, an additional programmatic layer is required that would translate the markup either into music notation, or MIDI code, or other technology. A decision will need to be made whether to allow only one method of marking either explicit or implicit rests, or both. If any method can be used, the application will need to handle both methods.

2. Time Module

In the time module the following components are distinguished: **tempo**, **note**, **bar**, **beat** and **tick**. Concepts such as length (i.e. duration) will be indicated either relatively with abstract numbers attached to note names (e.g. C:4 for a quarter C-note), or absolutely in terms of standard *hours:minutes:seconds:milliseconds* measures. Length values are always interpreted in terms of the **tempo** values.

The **tempo** attribute refers to the duration of music measures against absolute time. A note has both frequency and time, so the **note** attribute must be contained in this module as well. In descriptions it is possible to refer to a note's time without reference to its frequency, and vice versa.

The MML **bar** element is similar to the CWN "bar", but not restricted to its music notation meaning. In MML it concerns the rhythmic pattern of music measures. For most popular music there would be a one-to-one correlation between the CWN bar and the MML **bar** element. The **beat** and **tick** attributes refer to smaller time components within a **bar**.

At this point in the evolution of MML the above seem to be sufficient to address all the basic requirements of the details of possible core modules. All examples that have been marked thus far with MML could be handled by these, but it is possible that the core may need to be extended due to practical demands.

The other modules are either peripheral modules or distal modules.

3. Organization Module

The Organization Module concerns the generic extrinsic aspects of music, used for the organization of music on a larger scale. Components of this module include aspects such as compiling an album that consists of songs, and setting up a playlist that enables the user to select songs from various albums and from various formats. In terms of synthesized music control aspects such as selecting programs (**program** element) and banks (**bank** element) would also be contained in this module. Elements contained in this module are: **album**, **playlist**, **head** (which would have children such as **title**, **link** and **meta**) and **song** (which would have children such as **phrase**, **classes**, **div** and **span**).

4. General Module

The main function of this module is to contain general markup requirements such as the **comment** and **commentary**, but it would also contain elements relating to marking repetitions in music.

Music, viewed in terms of a function of frequency and time, could be defined more precisely as repetitive frequency patterns over time. These repetitions may vary from repeating a few notes to much larger themes (such as the popular AAB pattern). It is possible to create element handles that would mark repetitions. To mark all the minute details of music explicitly results in huge markup documents. Abbreviation methods are envisaged in this module to serve as a kind of shorthand which an additional processor should handle. XML parsers will not be able to expand the collapsed markup. Abbreviated forms may be economical, but would always require additional programs to XML parsers. Nevertheless, as XML parsers in practice will in any case require plug-ins to generate performed music or to display notation graphically from the textual markup, this may perhaps not be an issue as even abbreviated forms remain well-formed and valid, and thus meet the requirements of XML.

5. Texture Module

From a physics point of view there are at least two aspects that have an effect on sound texture. Borrowing from the terminology of the world of synthesized music, these aspects are envelope (**envelope** element) and harmonics (**harmonics** element). The amplitude intensity (i.e. volume) also has an effect on the core sound, so it has to be included in this module. For example, a note played softly or loudly has different textures. The volume here concerns intrinsic music aspects rather than the overall aspects that would be handled by the volume of a control system. The intrinsic volume will be called **intensity**, while the extrinsic volume would go by the name **volume**, handled by the Control Module.

6. Control Module

The Control Module would handle generic distal aspects of music, most of which are presently addressed in the SMIL 2.0 specification. These are, among others, **volume**, and on/off (when a song starts playing or stops playing and handled by a **status** attribute). The position of the sound source (i.e. **elevation** and **azimuth**) with reference to the listener would also be addressed here, as well as patterns of **accent** (i.e. relative loudness within bars of phrases).

7. Effects module

Music is never "pure": the environment has a big influence on exactly what a performed or played piece of music will sound like. The core of music can also be synthetically manipulated as is done in sound recording studios. It is thus necessary to introduce an effects module that can be used to manipulate the core sound to produce unique sounds.

Borrowing from synthesized music, the following aspects are addressed in the effects module. For the sake of categorization three sub-modules are identified based on which aspect of music the effect is applied: time, frequency and filters. The distinctions below reflect commonly used terminology but needs more research as there is considerable overlapping.

- *Time Effects Module*
  - Elements: **echo**, **delay**, **chorus**, **reverberation**, **feedback**, **flanging**
- *Frequency Effects Module*
  - Elements: **compression**, **feedback**, **equalization**, **pitch-shift**, **expansion**
- *Filters Effects Module*
  - Elements: **ringmod** (ring modulation), **resonance**, **limiting**, **noisegate**, **noisereduction**.

8. Performance Module

As presently envisaged this module does not concern the performance aspects of SDML (such as baton) but rather the control of synthetic music aspects such as foot, hand and breath controls. The aspects handled in this module thus relate more to MIDI than to SDML. At present the identified features are handled by attributes (***foot***, ***pedal***, ***portamento***, ***breath***) of the **control** element.

9. Notation Module

Most XML attempts to create music markup languages address notation which focus on simple music and not complex music events. A comprehensive general music markup language will have to address more, such as the following.

Reference needs to be made of mapping to Unicode music symbols [20] and how the details of a note should be marked. In the proposed model the abstract note is marked by using the Core Module elements. These are expressible by attaching other modules to the core set, for example to the MIDI module, or when a graphic record is required, to the Notation Module. The proposed model of MML, although utilizing the terms and concepts of CWN, should thus not be confused with CWN. In the MML model a notation system such as CWN is a subset of the Notation Module, which contains the tools for describing and marking all possible notation systems. When required, the CWN part of the Notation Module is attached to the Core Module elements.

The Notation Module thus contains markup for notation specific elements which are irrelevant to other modes of music. Some of these are: **key**, **cleff**, **staff**, **bind**, **tie**, **slur**, **rest**, **text** (such as *Andante*). Other features, such as dotted notes, and sharps and flats, and canceling them are handled by reserved symbols used with textual content. The **text** element handled one kind of music text. The language of the lyrics text is handled by a lyrics module.

10. Lyrics Module

The lyrics module will handle the lyrical text associated with music. There are several requirements that need to be addressed such as mapping the language of the lyric text to the music. Language also has repetitive patterns such as syllables. There is not necessarily a one-to-one correlation between language syllabic beats and music bar beats. Some syllables **stretch** across notes, while some are shortened (**squash** element), resulting in more than one mapping to a single note. This is only true for notation, as a note would sound on each syllable in performed music, no matter how many syllables are squashed onto a single written note.

The other aspect that needs to be addressed in this module would be mapping different verses to the same music sound sequence, and other aspects such as coda.

A challenge that needs to be addressed is how to synchronize the markup for music notes and the markup for lyrics. It would obviously not be possible to handle this with a typical XML processor, which will merely render the markup in sequential order. If, for example, the music notes markup is written first in the document, followed by the lyrics markup, an XML parser would merely render all the notes first, then the lyrics. It would be up to an additional processor to match these layers in order to be rendered together.

To enable matching, in MML it is proposed that such matching should be made by referencing the ***barid*** attribute of the **bar** element with a ***barref*** attribute of the **lyric** element. This would imply that the entire markup should be in memory, to be rendered according to some or other style and media preference. It may be possible to follow the DOM (Document Object Model) in this context [3,4].

11. Synthesizer Module

Sound cards of computers nowadays typically contain synthesizer chips. A general music markup language should also address this. As synthesizer chip makers have different approaches in their architectural structuring of the music components of chips, I propose an abstract Universal Synthesizer which could be used to map lookup tables to the idiosyncrasies of different manufacturers. That this would be necessary can be testified by anyone who has had to master the terminology used by different synthesizer manufacturers. Terms such as sound set, banks, instruments, parts and a host of others often refer to confusingly different aspects of the same possible set of

things. The terms proposed here are introduced based on about 20 years of synthesizer experience and thus not necessarily unbiased.

Some elements proposed for the Synthesizer Module are: **synth**, **prim** (primitives), **core**, **effects**, **soundset**, **texture**, **instrument**, **band** and **adsr** (or a more complex variation of this).

Other aspects that perhaps should belong to a sub-module, but needs further investigation, would refer to the circuitry and specific music generation aspects of the synthesizer. This would address the fact that some synthesizers allow different paths to be taken (e.g. an effect could apply either after a certain sound is generated, or somewhere in the process of the sound being built). Envisaged components are: **circuit** element with attributes such as ***input***, ***output***, ***target*** and ***exclude***.

12. MIDI module

The MIDI module would include element and attribute names that map to MIDI Controllers and functions. It is envisaged that this module could serve as a mode of transport between not only MIDI-enabled devices, but any device that can interpret the MML MIDI module. Several MML modules would apply in a MIDI environment: Frequency, Time, Effects, Control and Performance.

9 CONCLUSION

The proposed general music markup language, in this case MML, is a work in progress and far from complete. It is possible that further modules will be introduced, or that the organization of modules change due to practical demands. But even in its incomplete state it presently seems to be the only XML-based attempt to describe a very large scope of the domain of music. Other current attempts at marking music focus on a subset of CWN, which is useful in the early days of an XML-based markup language addressing music issues, but which do not address important issues such as performed music or playlists. Hopefully MML can serve as a basis for future joint efforts to comprehensively describe music using XML as basis.

10 REFERENCES

1. Campbell, D MML-to-MIDI Perl script http://www.musicmarkup.info/scripts/
2. CSS 2.0 (Cascading Style Sheets) 12 May 1998 http://www.w3.org/Style/CSS/
3. DOM 1 October 1998 http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001
4. DOM 2 WD October 2001 http://www.w3.org/TR/2001/WD-DOM-Level-2-HTML-20011025
5. Grande, C. The Notation Interchange File Format in Selfridge-Field, E. (ed): 1997, 491-512
6. Markup languages (W3C) http://www.w3.org/MarkUp/
7. MIDI Manufacturer's Association http://www.midi.org/
8. MML (Music Markup Language) http://www.musicmarkup.info/
9. Newcomb, S.R., Kipp, N.A., Newcomb, V.T. HyTime *Communications of the ACM*, November 1991, 67-83
10. Prévost, E. No sound is innocent Wiltshire, UK: Anthony Rowe Ltd, 1995
11. Rastall, R. The Notation of Western Music, JM Dent & Sons Ltd: London, 1983
12. Robin Cover's HyTime http://www.oasis-open.org/cover/hytime.html
13. Robin Cover The XML Cover Pages: XML and Music (March 03, 2001) http://xml.coverpages.org/xmlMusic.html
14. Selfridge-Field, E. (ed) Beyond MIDI: the Handbook of Musical Codes, Cambridge, Mass: MIT Press, 1997
15. Sloan, D., Newcomb, S.R. HyTime and Standard Music Description Language, in Selfridge-Field, E. (ed) 1997, 469-490
16. SMDL (Standard Music Description Language) http://www.hightext.com/IHC96/ihc96271.htm#STANDARD-18
17. SMIL 1.0 (Synchronized Multimedia Integration Language) 15 June 1998 SMIL 2.0 (Synchronized Multimedia Integration Language) 07 August 2001 http://www.w3.org/TR/REC-smil/
18. Sundberg, J. The Science of Musical Sounds, Academic Press, INC: New York, 1991
19. SVG (Scalable Vector Graphics) 19 July 2001 http://www.w3.org/Graphics/SVG/
20. Unicode Music Symbols http://www.unicode.org/charts/PDF/U1D100.pdf
21. W3C (World Wide Web Consortium) http://www.w3c.org/
22. XML 1.0 (Extensible Markup Language) 10 February 1998 http://www.w3.org/XML/

Modeling Music using XML
— Some Basic Considerations —

Markus Lepper
Technische Universität Berlin, Fakultät IV,
Institut für Softwaretechnik und Theoretische Informatik, ÜBB,
Sekr. FR 5–13, Franklinstr. 28/29, D–10587 Berlin,
E-mail: lepper@cs.tu-berlin.de

Abstract—
The paper tries to give basic outline of a holistic and analytic approach to capture all „thinkable" kinds of music in a generic and aspect-oriented XML-based architecture.

I. Idealized Aims for a (Not-So-)Far Future

It may hardly be called an exaggeration to say that the ongoing development of international standards of data formats and encodings based on XML realizes a **change of paradigms**. This is not because of the quality of the XML-level-one de nition [1], — indeed there are certain ugly design a ws due to its historic genesis — but due to the potential of *genericity* and *compositionality*, which has just started to display its power.

The reason for the wide-spread applicability of XML is the very trivial fact of rapidly increasing transmission bandwidth and processing power. This allows a very old paradigm from the rst days of ancient UNIX design to become accepted nowadays: That *text* is the adequate basis to represent complex matters. But in contrast to ancient UNIX, these texts are *structured* in a speci ed way, and the information content is coded implying a certain *abstraction* from the physical appearance of the text object.

While it was practice in the eld of information representation during the last decades to talk about bits and bytes on the encoding level, now we can (and have to) speak on a *semantic* level. This implies the opportunity to realize some requirements which are especially important in (or even characteristic for) the eld of *æsthetic* production:

Genericity of data formats is a central issue, since the expressiveness, syntax and vocabulary of project-speci c data representation should in no ways be limited *a priori*.

Furthermore *compositionality* allows the combination and mutual reference of widely diversive data, — an important contribution to creative liberty when using digital machines for æsthetic production.

[1] With „level-one" we mean the core de nition of the XML-document formats, not regarding schema de nition languages, as given in [BPSMM].

So (in the long run) we want e.g. to ...

- Feed the output of some off-the-shelf composition algorithm into a score rendering program of another vendor and into a sequencer or performer of a third vendor,
- attach analytic results or didactical information to a musical score or to audio data,
- generate and store the relations between audio data and notation data in different formats,
- synchronize audible sound data, visual information and control parameters for some on-the-stage installation to describe *synæsthetic* pieces of art,
- feed the output of a notation program into some self-de ned *ad-hoc* transformation or analysis software, and *vice versa*.
- generate electronic interpretations from notated music by algorithmic transformation, while keeping the different stages of data consistent in a mostly automated way.
- fill eg. a CALS-based table model or an SVG-based diagram with fragments of music notation when authoring a text in the eld of musicology.

All these tasks can of course already be done nowadays, but we have to pay the price of tedious programming of data converters, which are always specialized and hardly reusable.

What de nitely can *not* be done today is an *automated* generation of those trans-coders and analyzers, which requires a standardized de nition of *semantics* (together with some encoding de nition).

The conict between the (necessarily) x ed semantic of a desirable standardization on one hand, and the need of combinability, extensibility and integrability of *ad hoc* inventions of own structure de nitions, is a „creative conict ", requiring solutions which push these „idealistic" postulations down to the level of mathematically concrete architecture de nitions.

The following text wants to make an according proposal.

II. Analysis

A. Coding and Semantics

W.r.t. data format de nitions for supporting inter-operability, the central problem is *not* a matter of encoding. Let us call

0-7695-1864-8/02 $17.00 * 2002 IEEE

„encoding" each transformation from a semantic model S into some grammar of „physical" data items P. It is easy to see that in fact *all* encodings, which are complete and *injective* functions $c_n : S \rightarrowtail P_n$ are totally equivalent w.r.t. information contents: Since we always can construct $c_n^{-1} : P_n \twoheadrightarrow S$, we can for each two encodings c_m, c_n give a trans-coding function $c_m \circ c_n^{-1} : P_n \twoheadrightarrow P_m$.

So coding *per se* is **not an issue**!

Of course two different encodings will probably differ w.r.t. the performance of certain applications which operate on the data. This is an important aspect which must be respected, but should stay in the second line, as long as the basics are not yet defined[2].

So the central problem is not encoding, but *modeling*. What is it, what we want to encode, — what is this semantic model S which has to be mapped to P_n ?

In case of technical applications this question is answered by the operational semantics of e.g. some given protocol: We easily can encode SQL terms, SIP interactions, TCP requests etc. in XML by choosing any arbitrary *ad-hoc* transformation, because their semantics are specified by construction, e.g. using mathematical models of state machines, grammars etc.

But in case of *cultural phenomena* the situations is totally different: The first step for constructing any semantic model must be an analysis of the existing practice and of the naïve, informal pre-knowledge. The results of this analysis maybe can form a basis for constructing the necessary semantics.

In other words: We have to answer precisely the question „What *is* music" before we can start to define any encoding[3].

For this purpose we will now present a minimized meta-meta-model[4] of cultural interaction, apply it to musical practice and try to find out the unavoidable consequences for adequate models of music.

B. *Interludium: Basic Meta-Meta-Model*

Our analysis is based on a very (very, very) simplified KANTian theory of cognition.

Roughly spoken, it says ...

- that there *does exists* one single „objective reality" (the Ding-an-ſich), totally independent from all human perception,
- but that this „objective world" is not perceivable by humans at all. All we can perceive is completely bound to our Erfahrung (experience), which is always a common product of that Ding-an-ſich and our Wahrnehmungsvermögen (Mode of Perception).

[2] Esp. because the decisions of to-day should take into account not only existing technology, but even more the future development of the next two decades, which seems to be the life-cycle of standards.

[3] The naïve objection, that there do exist many different encodings (with their underlying, but mostly not explicitly defined, semantics) which are well-proven in practical application, is disproved by the every-day experience of composers (as well in traditional style using notation programs, as in avant-garde-style using sound processing languages) that very often the means offered by some standard or product have to be *abused* to achieve the desired effects, e.g. squeezing some MIDI pitch encoding to micro-intervals etc.

[4] Which perhaps better should be called „infra-model".

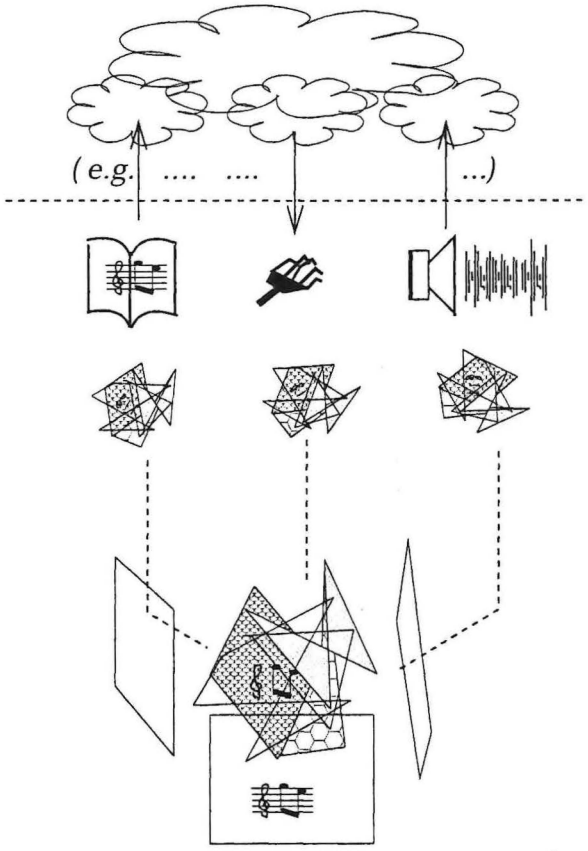


Fig. 1. Interchanging Different Aspect Models Related to „the same" Ding

This common product is always and necessarily „undecryptable". It can be compared to some encrypted text, where the private key has infinitely many digits: Mode of Perception and Ding together produce the experience, and all three are totally separated and incomparable aspects of the world.

So the only world which ever can appear to a human mind is a Mental Model. This world is called Wirk-lichkeit, because only internal to this mental model the notions of „cause and consequence" (= Wirk—ung) are valid, and only internal to this world any notion of time does make sense. This world is specific for (and internal to) any single human mind, and it is impossible to interchange any information between two of these universes: No one can ever transfer to another mind directly the concrete feelings connected with the personal reception of „salty", „red" or „BEETHOVEN's Fifth Symphony"[5] [6].

This KANTian meta-model solves the old problem of the „praeſtabilierte Harmonie" by LEIBNIZ *et.al*, i.e. it explains, why mathematics can describe the Wirklichkeit[7], — it unifies the

[5] We call this tranßendentale Geſchiedenheit.

[6] Even if neuro-psychology would be able to record the electrical patterns connected to such receptions, any transfer to another material brain will certainly cause something rather similar, but (thanks God) substantially total different, — just because the brains, to which this patterns are applied, *are* different.

[7] The English word „reality" and the corresponding German *Realität* refer

correct arguments of the idealism hypothesis with those of the realism hypothesis, — it states that all further attempts in theory of science and of cognition do operate on a „model-theoretic" layer of reality, not on something „physical", — and it helps a lot to clarify the use of language when talking about pieces of art.

C. Application to Music

Applying this meta-meta-model to music we therefore can suppose that there is one single, self-identic, independently of all human reception autarkicly existing Ding called „BEETHOVEN's Fifth Symphony".

But this Ding is never perceivable, — instead there exist *models*, and every human mind carries one personal model of „BEETHOVEN's Fifth Symphony", constructed by the accumulation of his/her life-long experiences with all those situations, which by convention are classified as being related to this Ding.

As told above, no direct exchange of these internal models is possible between humans. But of course we are able to *act*: From the viewpoint of the acting mind, actions are operations on the mental model, which are physically (and by means which are themselves unperceivable) related to the Ding. So the modifications we impress into the Ding can of course be perceived by other humans, influencing their mental models.

In case of music these actions can be: — playing a melody on the piano, — writing notes onto a board, — talking about structures and harmonics, — replaying a tape recording, — conducting an orchestra.

All these operations do modify a tiny-tiny part of the large Ding-an-sich, which is the universe. But since we do not operate on the Ding directly, indeed these actions imply the pre-going construction of specialized sub-models of the mental model „BEETHOVEN's Fifth Symphony", which model certain *aspects* of the totality, which then can be imprinted onto the universe to allow inter-human communication.

If we call these sub-models *physical aspect-models*, we can say that...

Theorem 1 *our mental model of any piece of art is exclusively* constituted *by our (social) experiences when interchanging* physical aspect-models.

Figure 1 illustrates this psycho-social interactions and the different projections centered around the transcendental Ding.

D. Establishing the Relation from Aspect Models to the Ding

One important consequence of this fact is ...

Theorem 2 *the relation from physical actions and the corresponding aspect-model to the Ding must always* explicitely *stated in an* axiomatic *way.*

to the Latin *res*, which means German *Ding*, so that *Realität* can be considered synonymous to Ding-an-Sich, the autarkicly existing, but un-perceivable world. The terminus *Wirklichkeit* but refers to „Ursache und Wirkung" (cause and consequence) and means those mental models. The English language does not support this distinction.

Imagine an orchestra during warming-up before a concert or rehearsal, when you hear some violinist playing something which sounds like ...



When asking the player, what it *is*, what he or she has played, you can get very different answers:

1) „I played the first notes of my part, the part of the second violin — I did the same thing I will have to do in a few minutes, right at the begin of the performance of the piece."
2) „I played the well-known beginning of the piece as you could read it in any piano-score, and I could have played it on any instrument which produces well-tempered pitches, but accidentally I have a violin in my hands."
3) „I played the well-known *motif* from the piece, which happens to appear everywhere in the orchestral score."
4) „I played the beginning of the recapitulation section, because there the conductor wants a different *timbre*, which I just practiced."

In cases 2 and 3 the player *refers explicitly* to an abstract model of analysis, which contains only a certain combination of rhythmic values and pitch classes, — in the other cases he or she constructs an index into the larger context of a certain audible interpretation. The same physical action can be *meant* as a model for totally different things of totally diversive contexts.

Another example:

When looking at some written or printed musical text, in most cases parameters like the kind of paper and the chemical composition of the used ink is *not* part of this model: If I want to play a piano piece I only look at the position of the small black circles w.r.t. the horizontal lines.

But some methods in historical musicology deal explicitly with ink, paper, chemistry when looking at a piece of written music. Furthermore it does make a difference w.r.t. law and prosecution, if the copy I am playing from has been legally purchased or illegally photo-copied, — leaving the compositorial structure of the sonata totally untouched.

E. A Possible Classification of Aspect-Models for Music

One first attempt to classify these models of music may identify three areas of formats, cf. figure 2, which can be called...

- Action
- Notation
- Concept

One example for a pure model of **action** flavor is the MIDI encoding, which originally only intends to reflect the activities of a keyboard player, described by the physical coordinates of key presses and releases.

Some older examples are the TABULATURA-style notations, specifying directly the mechanical activities which shall produce the meant musical Ding of music.

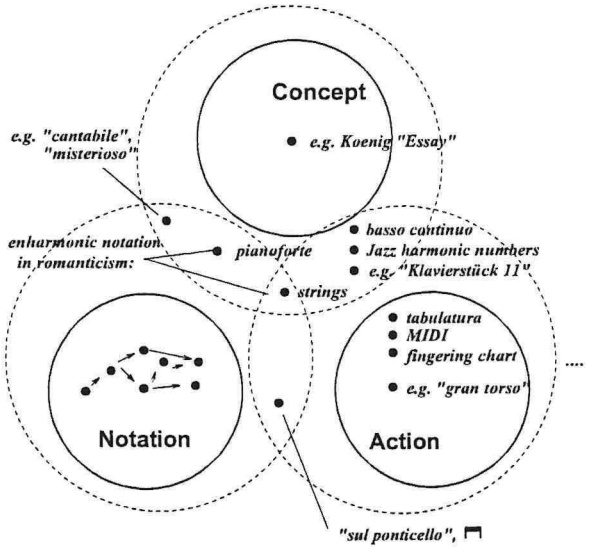


Fig. 2. A possible classification for Aspect-Models of Music

An impressive recent example is HELMUT LACHENMANN's „*Gran Torso*" for string quartet, where the activities the player physically perform make up the material which is organized by compositoric decisions.

The **notation** area is populated with the different, most diversive procedures for fixing music as scripture. Please note that already only the western European tradition during four centuries did produce a whole *family* of notation systems. Since these include even *contradicting* definitions of „encoding" (e.g. w.r.t. the scope of validity of additional accidentials), any attempt to construct a unifying and comprehensive modeling including all these notations must necessarily fail.

The notation of **concepts** became relevant to practice primarily in the last half century with the evolution of the *Darmstadt*-style avant-garde („Neue Musik"): E.g. in G.M.KOENIGs electronical piece „*Essay*" the formulae and data which *generate* the music are notated as lists of integers, and all compositoric/constructive decisions are presented openly to the reader or interpreting musician. Similar with STOCKHAUSEN's „*Studie Eins*", where an important part of the score is made up by the detailed description of the production process.

Any encoding of music cannot be called exhaustive, unless it is capable of representing even these pieces and notations.

Of course these areas indeed overlap, — something like *CANTABILE* is firstly a semantical directive for interpretation („like singing"), therefore something conceptual, but is contained in the notation, and requesting some physical way of producing notes (legato or portamento, medium range of volume, etc.), thus indicating actions.

Some other termini like *A TEMPO ALLEGRO* historically began as indication of semantics („happy and gay"), and ended as mere technical advice how fast to tune your metronome.

The *BASSO CONTINUO* notation and the jazz-style harmonic number system is part of the concept world. But having practiced enough to play a Generalbaß *prima vista*, it becomes a kind of action indication, — the hand nearly automatically is formed to the grip corresponding to the numbers, and you only have to hit it down.

„*Klavierstück Elf*" by STOCKHAUSEN is just a large poster on which some fragments of notated music are dispersed, and has to be played following the accidental jumps of the eyes of the pianist. This also can be considered as a combination of action notation and conceptual contents.

Beginning with C.PH.E. BACH, the *enharmonic* notation becomes more and more relevant as an expression of **concept**: especially when a pianoforte is the instrument, where a naïve and verbatim realization by pitch modification is impossible, enharmonic notation is *purely* conceptual (cf. BEETHOVEN, op. 110, the a" repetition preceding the *ARIOSO*, SCHUBERT, Sonata in a-minor, etc.)

When instruments with flexible pitch are involved (cf. MOZART, KV 550/I, return section to the recapitulation section, WAGNER, „Schicksalfrage-Motiv", featuring `{c, d-sharp, g}` as a c-minor pseudo-function), enharmonics *may* additionally carry action semantics, because intonation may be influenced by the notation[8].

III. CONSTRUCTION

A. The Hypothesis of Events

While these „philosophical" considerations may seem to be somehow far-fetched, indeed the opposite is true: Ninety-nine percent of music representation and interpretation will work fine without them, but as soon as advanced compositoric or analytic techniques are applied, there will be a point where all *closed* notation systems will necessarily not suffice.

Indeed does the transcendental nature of music have a sever impact on every-days work with musical objects, even in the most trivial contexts, — but mostly in a non-obvious way.

Let us now continue our analysis and concretize it towards the construction of a meta-model:

Let S be a transcendental Ding, a piece of music, and $M_1 \ldots M_m$ be some models related to S by axiomatic proposition.

A reasonable hypothesis, which indeed is the basis for nearly all existing model languages and their encodings, is ...

Theorem 3 *The transcendental Ding S and each model M_n can be thought of being composed as a collection of* Events, $E(S)$ *and* $E(M_n)$.

The axiomatically proposed reference between S and M_n includes the axiomatic proposition of one certain mapping relation $E(S) \leftrightarrow E(M_n)$.

But since the transcendental Ding itself is not accessible to our experience, this does not help at all. Luckily we can conclude that ...

[8] But this is a derived, second-order phenomenon, since even in absence of enharmonics the intonation of a string quartet is *always* an act of interpretation: In the most simple C-Major context you always have e.g. at least *three* physical pitch classes, all notated with „e" (the third of the tonic and the fifth of the dominant to the parallel of the subdominant, and the fifth of the triple-dominant), between which each interpretation has to decide.

Theorem 4 *The axiomatic proposal of two distinct models M_1 and M_2 being related to the same Ding, must imply the definition of a mapping relation between the corresponding sets of events $E(M_1) \leftrightarrow E(M_2)$, consistent with both $E(M_1) \leftrightarrow E(S)$ and $E(M_2) \leftrightarrow E(S)$,*

Please note that we have in neither case any *a priori* information about the structure of these mappings, if they are functions or even injective functions, if they are total or surjective etc. This indeed may cause severe practical problems, as will be shown in some examples later (cf. section III-D).

The next theorem is also an abstraction of every-day's practice and underlies most existing model languages:

Theorem 5 *Every event $e \in E(M_n)$ of some model M_n can be* totally *characterized by a family of (maybe partial) functions $P_p : E(M_n) \nrightarrow V_p$, where the ranges V_p of the functions P_p may be different for different p.*

The values $P_p(e)$ are usually called the „parameters of the event e".

Theorem 6 *Usually there are some ranges of parameters V_p, on which a* total order *is defined.*

And that's all. Sorry.

Indeed further analysis shows, that it is impossible to sensefully extract or define any further property of events and parameter functions, without massively narrowing the domain of representable musical concepts and data.

Practical Application: Consider e.g. a musical piece given (a) as full orchestral score, (b) as a set of orchestral parts, (c) as a set of different MIDI-interpretations, (d) as a set of different CSound-interpretations, and (d) a wave file containing the recording of a production. Each of these formats has totally different ways of indicating starting point, duration and pitch of the single events, and functions for converting from one format to the other may be rather complicated (e.g. requiring further parameters for an interpretative act like in the Rubato project [Maz90], or are simply context dependent, since global tempo indications have to be considered!) or even not existing at all (e.g. between different interpretations).

In our approach it does not make any sense to declare one of those formats to be the „real" music, and the others as „only derived". Instead all formats are first-order residents with equal rights, each format reflecting just different aspects of the abstract events an sich .

B. Foundation

Indeed we can use the results of the transcendental analysis almost immediately to give a sound foundation to the mathematical formulation of all possible models of music.

This is done by a kind of co-algebraic approach, which states, that the „objects" are not accessible *per se*, but that different *observations* can be made on a certain object, by applying to it one out of a family of „enquiring functions".

So we *can* model transcendental objects as such, as long as we do make observations only on their counterparts on model-level. We define ...

Definition 1 *Whenever enquiring for some information from a model of a piece of music S, there exists a fixed, known and finite set of „Abstract Events" A. This set is exclusively characterized by the fact that it is the one and only common domain for all parameter functions P_p.*

It is always true that $E(S) \subset A$.

This seems to be not much, but indeed it is a very practical and powerful approach: It implies that e.g. when modeling a violin glissando or a crescendo or a parameter interpolation in an electronic context, it must always be sufficient to model only a *finite* set of data points. For an uncomplicated mathematical foundation of transformation and language semantics this is a necessary requirement.

In practice this means that all modeling of a given piece of music is done by incrementally adding parameter functions, the domain of which is always the same, namely a finite set (e.g. of natural numbers) representing *abstract events*. The *only* purpose of this set of abstract events is to establish a relation between the *domains* of two given parameter functions. Abstract events are never accessible by our perception, and we cannot state anything about them, beyond their usage as elements of the overall common domain[9]!

Practical Application: Please note that also all „auxiliary" score items like

- tempo indications and interpretation directives,
- key signatures and measure indications,
- titles, texts and rehearsal marks,
- etc.

have to be modeled as „Abstract Events" to fit into our framework.

C. Parameters and Scales

Since no further statements can be made on these morphisms, all semantic discussion must take place w.r.t. the parameter functions.

For sake of inter-operability, compositionality and precise semantics we suggest to refine the *range* side of the parameter functions by splitting it into to parts, which may be specified *separately*: The first of which indicates an appropriate *scale* which governs the parameter, the second contains a vector of some scale-specific numeric values:

$$P_p : E(M_n) \nrightarrow K_p \times \overline{V_p}$$

$$\overline{V_p} == V_{p,1}, \ldots, V_{p,k}$$

$$\begin{aligned} \forall p, n \quad \bullet \quad & V_{p,n} \equiv \mathbb{Z} \ \vee \ V_{p,n} \equiv \mathbb{Q} \\ \vee \quad & V_{p,n} \equiv \mathbb{R} \ \vee \ V_{p,n} \equiv \mathit{Text} \\ \vee \quad & V_{p,n} \equiv \mathbb{P}\mathbb{Z} \ \vee \ V_{p,n} \equiv \mathbb{P}\mathbb{Q} \ \vee \ V_{p,n} \equiv \mathbb{P}\mathbb{R} \end{aligned}$$

... where K_p is the set of appropriate *scale objects* for the parameter described by P_p, and $\overline{V_p}$ is a tuple of corresponding arguments into this scaling definition. Please note that these arguments can be restricted to be *simple*, i.e. no free types or structures are required on this level of construction.

It is a central requirement that the scale objects are themselves *definable within* any musical score! Consider e.g. figure 3, which shows schematically an example very common in avant-garde style notation. It is easily seen that there are three

[9] Using XML-speak as a metaphor, this is like using `IDREF` attributes with corresponding values, without ever using a corresponding `ID` attribute.

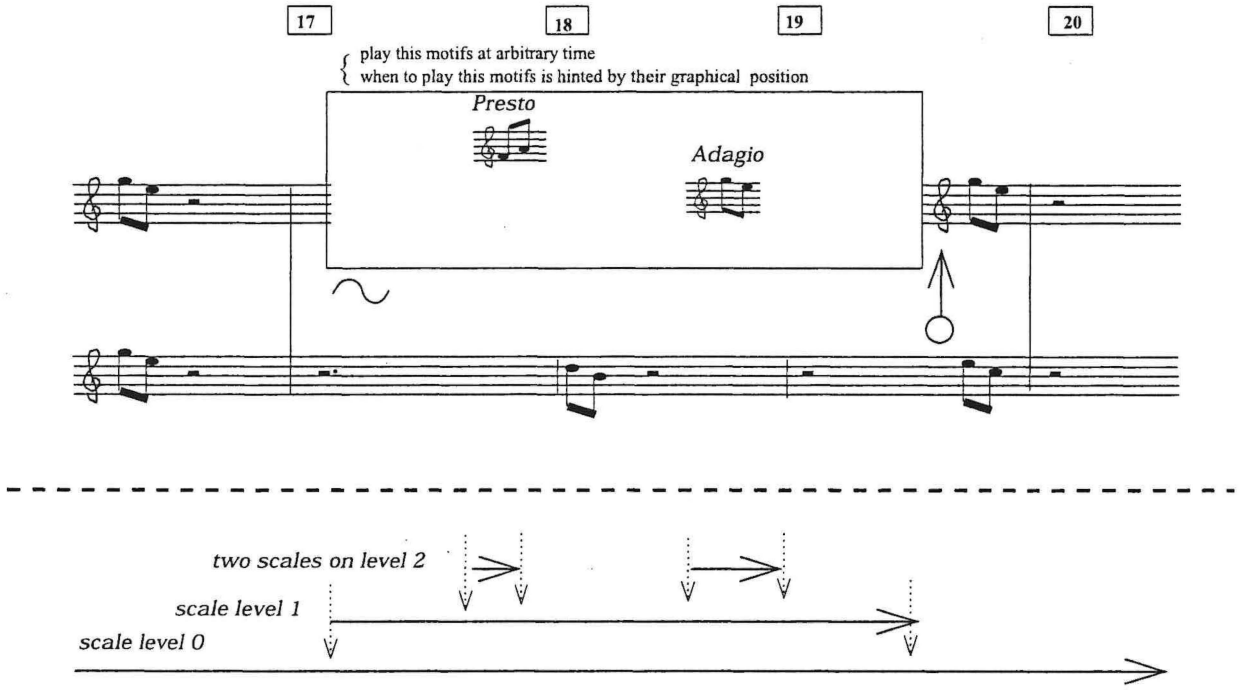


Fig. 3. A simple example of nested time scales

levels of scales, to which events are related, and which are defined in a nested manner[10] [11].

So we have to provide means for defining scale objects *within* a musical document, which can be accomplished by treating them similar to abstract events, yielding ...

Definition 2 *It is always true that* $K_p \in A$.

Now we can define scale objects by parameter functions P_k, similar to „normal" events, by ...

$$P_k : K \nrightarrow K \times \overline{V_p}$$

... where K is the union of all K_p, so that we need to additionally impose the requirement of non-circularity[12].

Practical Application: Initially there must be of course some predefined scales, which serve as the semantic foundation for the scale definition mechanism. These *a priori* scales will in most cases be oriented on some predefined „physical" models, cited from existing non-musical standards.

Additionally we define a meta-parameter-function called `constr`, pointing for each scale into a set of predefined *scale constructors*:

$$\texttt{constr} : K \rightarrow \mathcal{K}$$

In the example of nested scales, the values of the parameter functions `start` and `end` where given by recurring to abstract events. To express those kinds of equalities, we allow the „scale and arguments" pairs to be deputised by „parameter function and abstract event" pairs.

Thus the overall information structure which has to be modeled[13] turns out to be

$$P : A \nrightarrow \left((K \times \overline{V}) \cup (A \times P) \right)$$

D. A Possible Syntax for the Generic Semantics

Some possible realization which can be mapped to our semantic model is given in figure 4. Please note, that this grammar constitutes a generic superset, since it does not reflect the

[10] The two variants of the interpretation directive („when to play this motifs is hinted by their graphical position" vs. „play this motifs at arbitrary time") gives different semantics to the position parameter function w.r.t. audio interpretation, but does not affect the semantics w.r.t. score rendering.

[11] The author had to face in his work as composer an even more complicated situation: From a tape recording of BACH, KdF, Cp XVIII, a sequence of fragments had to be extracted, the starting point of which was determined by regular triolic/quintolic division on the **notation** level, while the duration was determined in „centimeter of tape". These fragment were arranged in time according to some other, again metric raster.

[12] The requirement of well-foundedness would suffice, but since complicated fix-point semantics seem not to be necessary, we make this harder requirement.

[13] Of course only the „correctly typed" expressions are part of the language. Since the overall typing requirements are too complex to be expressed by mere indexing, the indices have been omitted.

context conditions the parameter scales and the scale constructors impose on the types of their numeric parameters. Only the documents conformant with these are of course valid.

Please note that grammar in figure 4 is meant as the grammar of an *abstract* syntax, ie. does abstract from precedences etc.

Practical Application: The construct *EquivalenceMap* reflects the fact, that sometimes during a refinement process $1 : n$ or $n : 1$ identities have to be stated between abstract events, but that $m : n$ equivalences should be avoided :

E.g. a chord, which on notation level or as a notion of harmonic analysis is a *single* event with a pitch parameter of a `set`-type range, has to be modeled as a multitude of events w.r.t. time and intensity, when describing a certain piano *interpretation*, — a single node with a triller ornament must be split to many events in a Midi realization, etc.

E. Stacked Semantics and Packed Representation

According to our experience in language transformation, eg. using XSLT-scripts (cf. [Cla]) and visitor concepts, it seems outmost important that the semantic model (a) is „flat", i.e. realizes nothing more than an orthogonal function similar to

$$AbstractEvent \times ParameterIdent \nrightarrow Scale \times ParameterValues$$

and (b) that it does not make any use of implicit context information on the coding level, beyond the dependencies given explicitly by scale object creation and reference.

On the *coding* side the situation is different, and some hierarchical constructs can be quite useful, e.g. for (a) saving disk space and transmission time, and (b) to increase readability when working directly on the XML-representation, e.g. for debugging purposes.

Our approach offers two layers of hierarchy, namely (a) scoping of parameter functions and (b) track overlaying.

1) Scoping of Parameter Functions: Since normally certain parameter functions do have *identical values* for a certain range of events (or even for all), our grammar provides a means for scoping parameter values over *sets* of events and further parameter function specifications. So you can write (with the constructor ranges indicated schematically by indentation) :

```
scale 17
 http://ISO/music/pitchcodes/welltempered
scale 18
 http://ISO/music/durations/metric
           /sumsOfRationals

track 0  2002/05/30-15:12:11  mytool
param track 0
  param scale pitchclass 17 on
    param scale duration 18 on
      param value duration 1 4 on
        param value pitchclass c on 211
        param value pitchclass d on 212
        param value pitchclass e on 213
      param value duration 3 16 on
        param value pitchclass f on 214
      param value duration 1 32 on
        param value pitchclass g on 215
          param value pitchclass f on 216
  param value octave 1 on 211 212 213 214
                          215 216
```

2) Stacking of Tracks: Since in most cases transformations and further processing of a given musical document by a second tools implies only the modification of certain parameters, or even only attach additional information, leaving existing data untouched, we make use of the concept of *data tracks*: Each track may define parameter functions, while „inheriting" all parameter function values not overridden in this track.

This is especially useful for attaching eg. graphical rendering hints, results of analysis or interpretation data to a given musical score.

Together with the appropriate constructors for scales, and with the mechanism of defining scales and parameters separately, we can eg. create a *transposition* of the score above by just adding ...

```
scale 19  http://ISO/music/pitchcodes
                    /pitchclasstransposition

track 1 modifies sameDoc 0
   2002/05/30-15:12:11  mytool
param track 1
  param scale 19 +1 on  211 212 213
                        214 215 216
```

The same mechanism can be used (1) to derive instrumental parts from whole scores by exchanging the *geometric* scales, (2) to keep different *versions* of a composition in one document, (3) to create *interpretations* from notations by adding micro-intervalic derivation information and micro-timing, etc.

F. Mapping it to XML

Mapping of the language given in figure 4 to XML is straightforward. In a first step all nonterminals beginning with a capital letter are transformed into an appropriate *free type* definition, then for each constructor of this free type an XML tag is chosen.

Practical Application: Of course this mapping is in reality nontrivial, and different aspects have to be considered, some of them listed in the following:

1 —

The structure of XML-objects following the XML kernel specification is

$$Node == Tag \times (QIdent \nrightarrow Text) \times \mathsf{seq}(Node \cup Text)$$

If it were

$$Node == Tag \times (QIdent \nrightarrow \mathbf{Node}) \times \mathsf{seq}(Node \cup Text)$$

then XML-`ATTRIBUTE`s would be perfect for encoding all the non-positional and associative part of the information, while the „content" would carry the position and context-dependent part. But since the designer of SGML decided thirty years ago to restrict the range of all attribute functions to mere text, all pure-associative information which targets into *structured* domains cannot be expressed by the XML-`ATTRIBUTE` mechanism.

So whenever we want an open architecture which also allows the encoding of *expressions* of some sub-language to describe parameters, and we do not want these expressions to „fall out" of the structure of the document (of course you can always put expressions into pure

```
MusicalDocument   ::= eventHi ℕ trackHi ℕ (dataItem)+
dataItem          ::= TrackDeclaration | ParameterSpec
                    | EquivalenceMap | ScaleDeclaration | UriAliasDeclaration

ScaleDeclaration  ::= scale ℕ scaleConstructor ParameterSpec
TrackDeclaration  ::= track ℕ (modifies TrackId)? Date ToolIndication
TrackId           ::= (URI | sameDoc) ℕ
ToolIndication    ::= Vendor (Toolname)? (Version)?
EquivalenceMap    ::= equiv ℕ to ℕ+
                    | equiv ℕ+ to ℕ
UriAliasDeclaraion ::= alias Ident URI

ParameterSpecs    ::= param track ℕ ParameterScope

ParameterScope    ::= ParameterSpec on (ParameterScope | abstractEventId)+
ParemeterSpec     ::= param scale pIdent ℕ
                    | param value pIdent (ℕ | ℚ | ℝ | Text | abstracEventId | Set)+
Set               ::= set ℕ+ | set ℚ+ | set ℝ+

abstractEventId   ::= ℕ
pIdent            ::= URI | Ident
scaleConstructor  ::= URI | Ident
```

Fig. 4. A possible language for Musical Documents

text, but all XML-tools will not be able to process the expressions automatically any more), we should minimize or even avoid the usage of ATTRIBUTE-based encoding. Indeed these are not necessary from the viewpoint of mathematical semantics, since there are always equivalent representations using ELEMENT tags.

2 —

Further more we stress again that there should be minimal usage of *context* information, e.g. when giving entry time by relative distances, the references to the pre-going event should be made *explicitely*. This will substantially facilitate „external" processing of musical documents by standard processing tools, and since the XML encoding has to be converted into some internal format anyway, we do not see any advantage in giving any semantics to the document relative position of an item, which is just defined on the coding side.

3 —

The reader may have noticed that this paper, while speaking on XML, does not contain a single pair of angle brackets.

This is by intention.

According to our experiences with several standardization processes we strongly recommend to use professional devices of *sufficient abstractness* whenever discussing language specifications, like „abstract syntax", „denotational semantics" etc. Even the *concrete* front-end representation is much more easily specified as eg. a „constructor algebra" or by EBNF, the mapping of which into XML tags and content models is almost canonical.

IV. Filling in the Genericity

What is **not** covered in this paper and by our approach, is the „substantial" standardization, ie. the definition of (a) the semantics of the different scale constructors and (b) the meaning of standardized parameter identifiers.

The existing standards or standardization efforts (eg. [ISO], [m:n95]) contain valuable definitions which should be taken as a starting point.

The track concept presented herein allows to mix standardized and proprietary notions *ad libitum*. So the standardization process could be performed in an incremental way. Different ways of encoding the same parameter function may even be permitted to co-exist, because sufficiently precise definitions of their semantics will always induce a canonical mapping between them.

Further more it is possible to attach arbitrary *ad-hoc* parameters to any given structure: E.g. a composer might want to introduce a parameter like „color" to the printing of his notes, or even a parameter „taste", taking values from sweet to salty[14].

A more senseful application would be to attach *arbitrarily* defined results of musical analysis or auxiliary parameters for a special synthesizing algorithm to any existing, predefined or „downloaded" musical structure, so lifting the expressive power of compositionality, which is inherent to XML-documents and -declarations, to the level of æsthetic application.

References

[BPSMM] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. *Extensible Markup Language (XML) 1.0 (Second Edition)*. W3C Recommendation, http://www.w3.org/TR/2000/REC-xml.

[Cla] James Clark. *XSL Transformations (XSLT) Version 1.0*. W3C Recommendation, http://www.w3.org/TR/1999/REC-xslt.

[ISO] ISO/IEC. Standard music description language (sdml). Technical Report DIS10743.

[Maz90] Guerino Mazzola. *Geometrie der Töne - Elemente der Mathematischen Musiktheorie*. Birkhäuser, 1990.

[m:n95] Notation interchange file format. Technical report, 1995.

[14] ;-)

Towards a Symbolic/Time-Based Music language based on XML

Goffredo Haus, Maurizio Longari
LIM-DSI
Università degli Studi di Milano
via Comelico, 39
20135 Milano, Italia
+39 02 5835 6222, +39 02 5835 6296
haus@dsi.unimi.it, longari@dsi.unimi.it

ABSTRACT
The representation of Symbolic Music Information is the fundamental element of Music Information Processing. Among the several approaches developed in history the markupped one seems to be the most promising. Starting from SMDL, some XML proposals are briefly analyzed. We present our approach of layered representation of Symbolic Music Information based on the space-time organization of events. Our model splits music information in layers of representation and takes the symbolic one as the gravity center. Moreover, the symbolic layer contains a space-time structure by means of which all other layers are brought together. Then we discuss other XML standard that can be usefully related to the XML representation of music information, such as security and vector graphics standards. Lastly, we deal with some open problems in the field of music processing that might be of interest for our view of symbolic representation.

Keywords
Symbolic Music Representation, XML, SMDL

1 INTRODUCTION

Music Information Processing is a discipline as old as computer science. The representation of Symbolic Music Information is its fundamental element. Early attempts were made with DARMS, MUSIC V and other ASCII-based codes.

The advent of structured text (Standard Generalized Markup Language SGML) has raised the idea of representing this information in human-readable fashion. Structuring such information gives the possibility of representing several aspects of music and more sophisticated concepts hidden in Music Information. SMDL [25][28] (Standard Music Description Language) was the first attempt in this direction. The only drawback was the lack of ad-hoc tools for its implementation.

Another approach has come from the area of electronic music, namely, Musical Instrument Digital Interface (MIDI) and MIDI files. This file format contains sequences of timed symbolic music events. This is the most popular format used in electronic performances and for exchange of electronic scores. Unfortunately, its symbolic music expressiveness is very poor and it is not sufficient for professional, even semi-professional, music composition. It is, however, interesting that it can be located between performance and symbolic representation.

In the past few years, the synergy between Internet and eXtensible Markup Language (XML) [4] has brought to the fore the representation of content meta-data by means of markupped languages. Symbolic information can be thought as the intrinsic meta-data content of music. The purpose of this work is to explore the possibility of presenting our solution for symbolic music information using XML syntax.

The main concepts of this approach are *layer-subdivision* (spanning from signal to structural) and *space-time constructs*. We give herewith some definition of terms widely used through the article.

SMI: Symbolic Music Information (general, structural and music content information)

Source Material: graphic score images, notation files, digitized registrations, videos

Music Work: the set of source material and its related symbolic representation.

A case study in which these concepts are fully used is the Scala Project [14][16], developed at LIM [39]. In this project, the notion of Spine is the key element for the integration of scores (space dimension) with audio files (time dimension)[15].

In the next paragraphs, we briefly survey SMDL and several XML applications for the representation of music information. Then we describe the concepts of Layered SMI and its integration with markupped languages. In the last paragraphs, some related topics and open problems are discussed.

2 MARKUPPED SMI

The first approach to the representation of SMI with markup languages was that of SMDL. It was developed as the application of SGML for the representation of music information. Because of the complexity of SGML syntax, and of the fact that it was not diely applied, SGML was abandoned. The concepts of timed/based structures were maintained for the development of the much more successful standard HyTime (Hypermedia Time/Based Description Language [9][23]). In fact, a second review of SMDL was made that included the concepts developed in HyTime framework.

In the past few years, several proposal have been made to represent Symbolic Music Information by means of the XML standard. We will briefly discuss the topics of SMDL and then those of XML representations.

SMDL/HyTime

The Standard Music Description Language (SMDL [24]) was created in 1984 from an idea by Charles Goldfarb. The idea was to create an environment in which the several aspects of music representation could be considered and used. SMDL is an instance of HyTime and it inherits the functionality of this standard [9][23]. It divides the aspects of music representation in four domains: Logical (cantus), Visual, Gestual and Analytical.

The Logical Domain represents the intent of the composer when he writes a music piece. The Visual Domain incorporates graphic information and score notations. The Gestual Domain contains performance and executive information. The Analytical Domain embraces all music background information.

The Logical Domain (called *cantus*) codes music symbols in SGML syntax. Other domains use the hyperlink capabilities of HyTime to relate to other representation of specific music information [9].

The SGML syntax of SMDL is defined in the standard draft ISO 10743 [25]. The structure of information has been developed over a system of time-dependent axis called FCS (Finite Coordinate Space, an architectural form of HyTime) which is composed of cantus events (notes, rests, symbols). The other domains may refer to this representation to link the content of music information.

MusicXML

The definition of this proposal [20] has been under development by Michael Good of Recordare. The content information is based on the MuseData and Humdrum [27] formats that explicitly represent symbolic music information in two dimensions (time/measure and part/instrument). Since XML has a hierarchical structure, the representation can be actualized as one of two conceptual hierarchies at a time. In the former representation, called Partwise, the sequence of measures is coded within each part. In the latter representation, called Timewise, the score is subdivided in measures, and subsequently each measure is subdivided in parts. Within each measure, or within each part, the symbols of the music events of the score are coded. An XSLT (eXstensible Stylesheet Language Transformations) [1] program has been developed that allows automatic translation between the two orderings. The MusicXML proposal is meant to represent a sufficient interchange format for notation, analysis, retrieval and performance application. The content represented by the format is score-oriented, e.g. the notes are represented as symbolic and graphical objects.

MusiXML

The aim of the proposal written by Gerd Castan [5] is to define a standard format for exchanging music score information. The structure is subdivided into three sections: Bibliographic Data Section, Logical Data Section and Filter Data Section. The last two sections are based on the idea of writing the minimal amount of redundancy information that separate content from style. The Logical Section is based on concepts of work-segments and Threads containing symbolic music content. The Filter Section is subdivided into *extracts* and *staffs*. One extract is a grouping of staffs with bibliographic data, brackets, braces and references to the staffs. A staff contains references to threads and some other attributes. An XML DTD and an XML Schema [8] definition of MusicXML are available. Note that the MusicXML Schema is richer than the MusicXML DTD, because the former provides tools that the latter does not.

MusiCat & MDL

MusiCat, defined by Perry Roland [26], is a catalogue-oriented encoding format. A MusiCat document is organized as a catalogue document and has four main sections: Header, Body, Front and Back. The purpose of the description is to encode new and existing music catalogues, to favor popularization of electronic storage and transmission of music information. In fact, the schema doesn't provide elements to describe music content, it lets instead other notation formats represent it. Only one element (<notation>) containing text that describes the music content is provided (in [18], the Humdrum format is referenced as an example).

MDL (Music Description Language) is the complementary definition for MusiCat to represent content music information in XML. It is a rich format with plenty of elements and attributes. The score is organized as the timewise structure of MusicXML. The attributes are subdivided in domains like those of SMDL (ie. Logical, Visual, Gestural and Analytical) and it gives

the option to define new ones.

WEDELMUSIC format
It is the format for the representation of music information within the WEDELMUSIC project [36]. It is declared as to be an XML compliant format with multimedia capabilities, including identification, classification, symbolic, visual, versioning, printing, protection, image score, image, document, performance, video and lyric aspect. The underlying idea is to build an open format for the representation of WEDEMUSIC objects. It keeps separate the visual/formatting aspect and the symbolic one. In the description presented at the WEDELMUSIC conference [3], the structure and some examples about the identification, classification and music notation modelling are explained.

Musical Notation Markup Language (MNML)
Developed at IRDU (Internet Research and Development Unit) of the National University of Singapore, this proposal is an attempt to represent the entire music and lyric contents of a music score. MNML fully describes the basic melody and the lyrics of the piece. Capability for defining complex layout and other publishing features, performance annotations, and other embellishments are not considered [22][37]. A prototype of MNML editor and search engine has been developed.

Music Markup Language (MML)
It has been developed at University of Pretoria under the direction of Jacques Steyn. The development of the proposal is carried out with student projects. The aim is to define a markup language that can represent the structure of music and of music related processes. The authors have also developed some experimental software to graphically render an MML file [17].

MuTaTeD
MuTaTeD [21] stands for Music Tagging Type Definition. It is a project coordinated by Carola Boehm that intends to validate the concept of integrating the two existing music representation standards, SMDL and NIFF, and to provide a meta-standard for music mark-up (SMDL DTD)[25].

MusicML
Developed by Jeroen van Rotterdam, it is a simple draft that can represent only basic music information. The author has also developed a Java applet to parse and visualize MusicML files. [19].

ChordML
This is a simple DTD developed by Gustavo Federico that can represent only lyrics and associated chords [6].

3 LAYERED SMI

SMI is organized in layers. Each layer models a different degree of abstraction of music information. An analogy

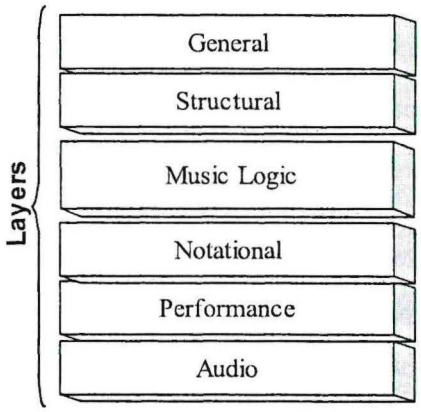


Figure 1. Structure of a Layered Symbolic Music Representation

can be made with the ISO/OSI layers architecture.

Essentially we distinguish among General Information, Structural Information, Music Logic Information, Graphic/Notational Information, Performance and Audio Information.

Music Logic Information is the core frame of our view of SMI. This information is composed by two fundamental concepts: the Spine and Logical Organized Symbols.

Spine is a structure that relates time and spatial information. With such a structure, it is possible to move from some point in a Notational instance to the relative point in a Performance or Audio instance. Logical Organized Symbols are the common ground for the music content. Music symbols are represented in the XML syntax making music content explicit to applications and users. Below we explain in details the concept of each layer.

General
The General Information layer is the place where metadata information about the music work is coded, such as:

a) description of the music event (date, place)
b) casting information
c) the table of related music data files, referring to all layers, with one or more files for the summarization of each layer
d) the table of related multimedia data files, such as images, videos, and the like
e) the table of related analog media
f) technical information about related import/export/restoring/cataloguing/other operations

g) general notes..

Any of the above mentioned items may contain all the related links to other layers.

Structural

The Structural information layer is about the explicit description of music objects and their relationships, from both the compositional and musicological point of view. There is no currently available standard which could be considered for this layer, but we can draw from:

- ANSI SMDL [25], which considers the topic, without giving any concrete tool for describing it
- High level music programming languages and formal tools for modeling [2], and particularly the descriptional approach based on Music Petri Nets [7][10][11][12][13].

The music information contents within this layer do not contain explicit descriptions of time ordering and absolute time instances of music events, while describing causal relationships among chunks of music information and related transformations and positioning within the music score, as they happen in the frame of compositional/decompositional - i.e. synthesis/analysis - processes.

Notational

Graphical and notational information regards the visual domain of music information. This layer embraces all the modalities in which the music can be written or read. This domain includes references to the content of images of scanned scores, and notational files in different formats such as NIFF and ENIGMA. Information contained in this layer is tied to the spatial part of Spine structure, allowing the localization of symbols.

Performance

This is the connection layer between symbolic performances and the SMI. Examples of source material considered in this context can be MIDI files, Csound or SASL/SAOL (MPEG4) compositions. Timed events can be suitably linked and synchronized to the SMI by means of the time part of Spine structure. Note that, in this context, time representation is expressed in relative coordinates.

Audio

The Audio layer relates to Source Material that contains musical audio information. It is the lowest level of our view. In order to automatically relate it to the time part of the Spine structure, it is necessary to extract features indicating the actual temporization of musical events. Formats representing audio information can be subdivided in two categories: compressed or not compressed. Not compressed audio can be PCM/WAV, AIFF and Mu-Law. Compressed audio can be subdivided

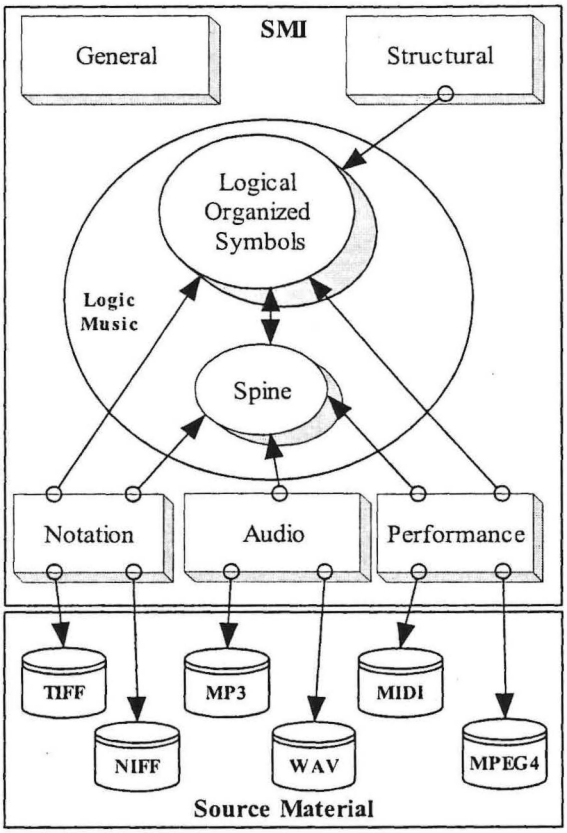


Figure 2: Relationships between SMI layers and between SMI and Source Material.

in lossy (e.g. MPEG and DOLBY) and lossless (ADPCM).

XML formats and layered SMI

Layered information is partially covered by some of the XML formats cited before. The most similar language is the WEDELMUSIC format that explicitly encode information on links to score images, notation, timed symbolic, audio and video files. The synchronization aspect is left to the application implementing the language.

MusicXML, MusiXML, MML and MDL represent symbolic information from different points of view. A common assumption is that the richness of the contents must allow visual or aural rendering. They do not consider explicit integration with other kind of information such as audio, video, graphical or structural representation.

MuTaTeD and MNML are closed projects and the specifications are no longer available. MusicML and ChordML are very simple formats and they do not represent information about any other music aspect.

4 XML REPRESENTATION OF LAYERED SMI

Each layer described so far has a role in the context of the integration of the Music Work. XML organizes information in a hierarchical structure. Each layer is represented in a sub-tree of the Music Work. Relationships between layers (and thus between sub-trees) are implemented by means of XML references. Relationships between SMI and Source Material are implemented as descriptions of extracted parameters. As we can see in Fig.2, the central element of the representation is the Music Logic Information, to which all the other layers must refer. We will give a deeply description of this layer in the next paragraph.

The General information layer does not link directly to any layer, since it describes information about the whole Musical Work. Its XML representation is straightforward and does not need any particular description.

The Structural layer represents objects and relations between them. Objects are portion of the Music Logic Symbols layer. XML representation of a specific structural methodology is an open issue of this work.

The Notational Layer is the layer where visual information can be described. In our view, it should be possible to describe the presentation of the Music Logic Information in any media. Therefore, we split the structure of this layer in two sub-layers (or sub-trees): *layout description* and *linking information*. The former provides the tools to describe printing (e.g. A4 or Letter paper) or screen layout (e.g. desktop, palm or electronic book display). This description links directly to the Logical Organized Symbols and the Spine events. The latter provides descriptors for the inclusion and mapping of score images and binary notation file formats (NIFF, ENIGMA or even MIDI files) to the Music Logic Information layer. Furthermore, this is the place where information about security (watermark, signature, etc.) of each single graphic and notation file of Source Material can be inserted.

The Performance layer should contain generic information about the source material and the actualization information (such as synthetic instrument description, metronome indication and so on). Actualization parameters are the bridge between the time part of the Spine structure and the timed symbolic information, since both represent timed information in relative coordinates. Thus, XML representation needs not be much detailed.

The Audio layer needs instead the representation of extracted features in order to be suitably integrated in SMI. And these features can be directly linked to the time part of the Spine structure.

5 MAX FRAMEWORK

MAX stands for Music Application using XML. As shown before, the core of our approach is the Music Logic Information view.

Spine

The Spine is a space-time structure composed of uniquely identified events. This is in fact the structure that relates Performance/Audio and Notational layers. Each event has two coordinates relative to the preceding event in its dimension, because some symbols can have meaning only in space or in time dimension. The first coordinate is the temporal coordinate of the event. It describes how much time must pass after the actualization of the preceding event. This time is expressed in Virtual Time Unit (**vtu**, as described in SMDL draft [25]). The second coordinate is expressed in a relative spatial unit that we call Virtual Logic Position Unit (**vlpu**). Its purpose is to represent the reference point for the vertical alignment of different symbols in different parts or staffs.

A problem arising in the definition of such a framework is the discrepancy between Notation and Performance layers caused by repetition symbols. The question is: should the Spine structure be tied to the Performance or to the Notational layer?

If Spine refers to the Performance layer, it is necessary to create multiple references for symbols lying in repeated segments. On the contrary, if Spine refers to the Notational layer, a mapping from Performance/Audio to the Spine layer is needed. This is in fact a still open issue of our representation and a deep investigation is needed before a definitive choice can be taken. For the time being, we have chosen to be tied to the Performance/Audio layers.

Logical Organized Symbols

The root element of Logical Organized Symbols is a score. Since the rationale of our work is to separate the content from the presentation, we first distinguish between parts and staffs. Different parts can be printed on the same staff, while one part can span different staffs. Hence, the staffs do not seem to be an essential part of the logical view. We represent them as a list with associated properties and identifiers. Each symbol is then linked to its own staff by means of an identifier reference.

Thus the part is the most important element of our view. Each part is composed by a sequence of measures, within which the Notation Symbols are coded. Each symbol belongs to a voice that is referenced as a property.

Each Measure must have a reference number in order to be clearly identified and associated to the relative measures in other parts.

The music symbols can be rests or chords. As in NIFF

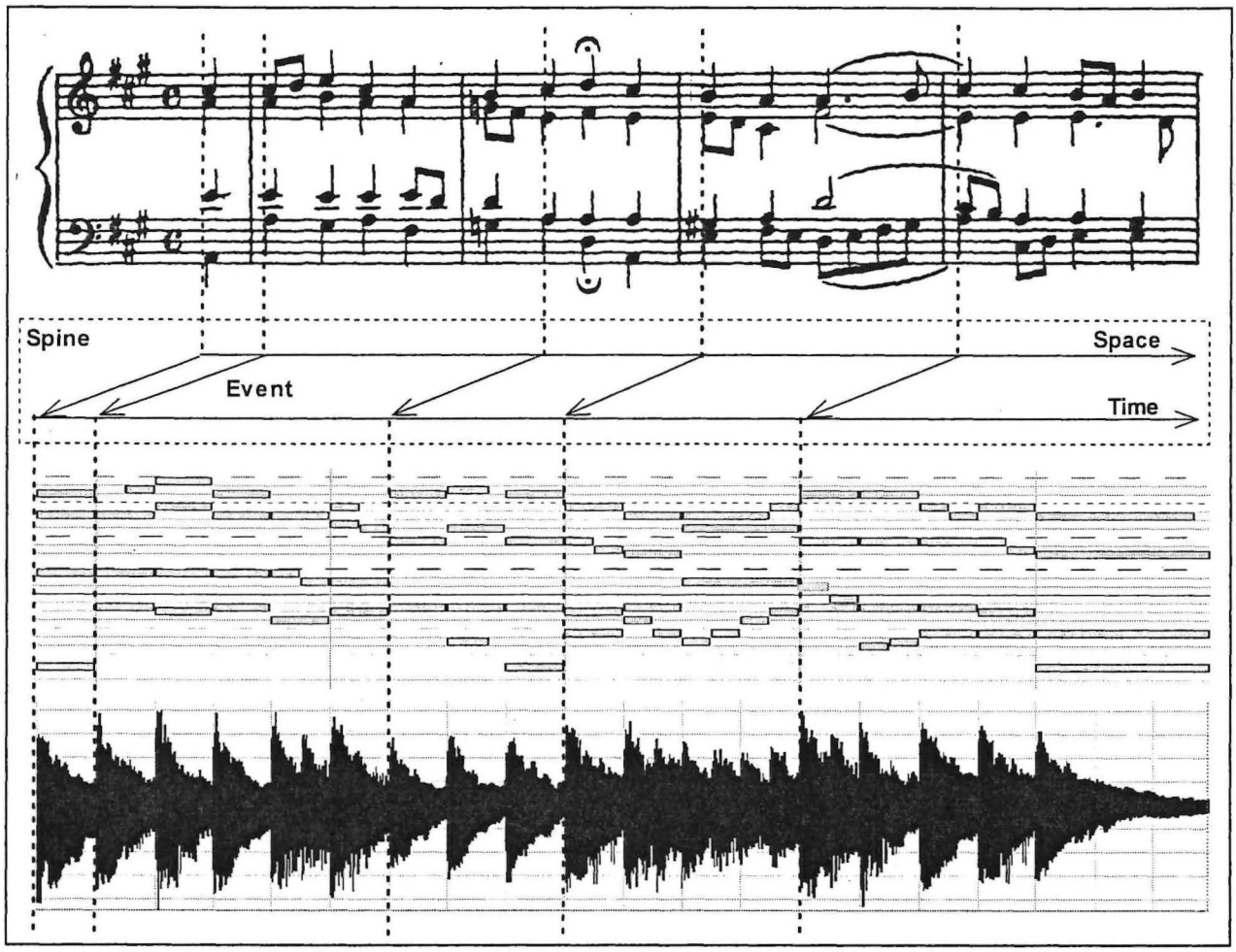


Figure 3 Spine. Relationships between Notational and Performance/Audio layer instances.

[38], a single note is a particular case of a chord with only one notehead. Each chord and rest contains the link to the relative event in Spine, so that it can be localized spatially and temporally. Some other property can be also encoded about the logical description of the symbol. Note that the properties specified in this context have meanings only with respect to the definition of the symbol, because timing and localization information are managed by the Spine events.

The representation of beams, tuplets, dynamics, ties and other grouping symbols or embellishments can be efficiently managed as external elements that relate to Music Symbols or to the Spine events. For example, the "crescendo" symbol must affect all the symbols present on a staff, and in this case it is much efficient to relate the symbol to the events in Spine, instead of linking to all the parts that relate to that staff.

There are several symbols that can affect the aural rendition of score, such as repetition bars, jumping signs and multiple endings. These structures can be summarized in XML with some elements that refer to segments in the Spine (since the repetition affect all the parts in the score). This formal tool defines projection of music symbols from the logical layer to the Performance/Audio layers. It avoids the encoding of redundant information and further separates the contents from presentation.

Lyrics can be thought as another logical component of our structure. The text is subdivided in syllables. Each syllable is tied to Spine and its relative part. This organization allows to overcome the problem of repetition in the score while the lyrics changes and allows multiple lyrics to be encoded and extracted easily (e.g. for creation of librettos). It also simplifies the organization of text in layout rendering.

Layout

Logical Organized Symbols can be automatically rendered by the device-specific application, or can be

specified by means of the Layout elements. The main container of this structure is the Layout, which is made by one or more Pages. On each Page, there can be one or more Systems or none. The System contains staffs and lyrics at the same level. Systems are related indirectly to Logical Music Symbols through the Spine structure. Particular rendition of symbols and non-conventional signs can be easily modeled with SVG (Support Vector Graphics)[34], an XML standard for the definition of vector graphics.

Performance
In this layer, we describe properties to render aurally the music symbols and descriptors of Timed Symbolic Music Information (e.g. MIDI, SAOL, Csound). Rendering of music symbols can be performed by means of interpretative models that are depicted later in this article. Timed Symbolic information is naturally linked to the time part of a Spine structure, since event temporization is expressed in relative unit measures. Thus the descriptors must not encode the information about each event, but only general information about the instance and its relation to the SMI (e.g. the instance can be relative only to one instrument, or it may span only a part of the whole piece).

Audio
In this layer, information must be encoded that is extracted from the audio instance of the piece. Beside the usual generic data on the audio file, we think it is useful to encode actual timing parameters such as piece length, beat and localization of musical events. Moreover, this information is independent from the format in which the audio information is stored, since compressed formats are uncompressed before being played back.

6 RELATED TOPICS

The development of XML standards covers general topics that can be used in our context. We have identified some standards that could cover some of the topic of interest, such as security, layout symbol customization and versioning. These topics have also been addressed in the WEDEMUSIC format [36], to develop new methodologies to store this information.

Security
Ever since the widespread adoption of Internet as an exchanging medium, the topic of security has gained a great interest. In the context of our work, we distinguish between the security encoded in the Source Material and the security expressed in the symbolic domain.

The Security in Source Material is out of the scope of this project. However, we face the problem of security in the symbolic domain. Here, we must further discriminate between security in XML (authentication SAML, access control XACML and digital rights management XRML) and security for XML (XML Signature and XML Encryption).

Security in XML is useful for the establishment of music computing environments, and of music services, based on XML. Security for XML is needed for the protection and ownerships validation of Music Works.

SAML
SAML (Security Assertion Markup Language) [30] is an XML-based security standard to exchange authentication and authorization information. It is being defined by the Security Services TC (SSTC) under OASIS. The working group envisions SAML as the exchange definition for authentication and authorization information.

XACML
XACML (eXtensible Access Control Markup Language) [31] is a core scheme for representing authorization and entitlement policies in XML. It is being defined by the XACML Technical Committee under OASIS. XACML is also expected to suggest a policy-authorization model to guide implementers of the authorization mechanism.

XRML
XrML (eXtensible rights Markup Language)[29]is a general-purpose, XML-based specification grammar for expressing rights and conditions associated with digital content, resources, and services. It is based on the rights expression language concept developed at the Xerox Palo Alto Research Center (PARC). XrML 2.0 expands the capabilities of a Digital Rights Language, usually seen in connection with the authorized use of protected digital content.

XML Signatures
The mission of this working group [32] is to develop an XML-compliant syntax for representing the signature of Web resources and portions of protocol messages (anything referenceable by a URI), as well as procedures for computing and verifying such signatures. This is a joint Working Group of the IETF and W3C.

XML Encryption
The mission of this Working Group (WG) [33] is to develop a process for encrypting/decrypting digital contents (including XML documents and portions thereof), and an XML syntax structure to represent the encrypted contents and information enabling an intended recipient to decrypt it.

Versioning
Personalization of scores is a common task for an orchestra director preparing the show. Often these modified versions have important value themselves, when made by important artists, and keeping tracks of these changes can be very complex. The most common are changes in music symbols, articulation, embellishments

or indications written in any language. In our framework, this can be implemented simultaneously from the point of view of contents and graphical representation (particular signs can be modeled with SVG syntax).

7 OPEN PROBLEMS

The research in the area of computer applications in music has at least two fundamental open issues of interest for our conception of Musical Work: *Timbre* and *Interpretative Models*.

The timbre problem concerns how to automatically extract timbral features from complex audio signal, so that timbre modelling could be improved by means of automatic synthesis of timbral models and their parameters. For example, while we are quite able to get MPEG SASL codes that extract features from audio signals with constraints, there are no currently available coder which could concretely be used to get MPEG4 SAOL audio codes.

On the other side, we have the interpretative problem: how to translate NIFF codes (i.e. scores) into MIDI codes (i.e. performances) and viceversa? There is a need for a functional unit, currently not considered in the music information life cycle, that is, an Interpretative Models. In this way, for example, we can get timed, instantiated music information, coded in MIDI format, starting from score NIFF codes which contain only time-ordered music information. Vice versa, we could eventually have good automatic transcription from audio signals to music scores.

The development of such topics will be considered for the integration in the MAX framework to provide a complete high-level description of multilayered music information.

8 CONCLUSIONS

We have described the topics and concepts we intend to implement and develop in the context of the Recommended Practice for the Definition of a Commonly Acceptable Music Application Using the XML Language, carried out by XML Music Application Working Group (WG) [35].

Future work planned addresses mainly:

- integration of all available, meaningful experiences of other research groups working on XML & Music topic
- development of innovative topics of interest, outlined in this paper; any cooperation is welcome.

As for all other XML standards, the main purpose of the MAX framework is to be extensible and open to the community of users of computer applications in music.

9 ACKNOWLEDGEMENTS

This work has been made possible by the effort of researchers and graduate students at LIM.

This research is partially supported under a grant by the Italian National Research Council (CNR) in the frame of the Finalized Project "Beni Culturali". The authors wish to acknowledge D. Baggi for improving the English language quality of the paper.

REFERENCES

1. AA. VV. Extensible Stylesheet Language (XSL) Version 1.0 *W3C Candidate Recommendation* 21 November 2000 http://www.w3.org/TR/xsl/
2. AA.VV. Standards in Computer Generated Music multiplatform mixed mode CD-ROM (Macintosh, Windows, Unix + CD-DA tracks), (G. Haus & I. Pighi Editors, 1996) IEEE Computer Society Press.
3. Bellini, P. Nesi, P. WEDELMUSIC Format: An XML Music Notation Format for Emerging Applications in *Proc. of WEDELMUSIC2001*, (Florence Italy, November 2001) IEEE Computer Society, 79-86.
4. Bray, T. Paoli, J. Sperberg-McQueen, C.M. Maler, E. Exstensible Markup Language (XML) 1.0 (second Edition) http://www.w3.org/TR/REC-xml
5. Castan, G. Common music notation and Computers: Index. http://www.s-line.de/homepages/gerd_castan/compmus/index_e.html
6. ChordML Web site http://www.cifranet.org/xml/ChordML.html
7. De Matteis, A. Haus, G. Formalization of Generative Structures within Stravinsky's The Rite of Spring in *Journal of New Music Research*, 25,1 (Amsterdam, 1996) Swets & Zeitlinger B.V., 47-76.
8. Fallside, D. C. XML Schema Part 0: Primer. http://www.w3.org/TR/xmlschema-0/
9. Goldfarb, C.F. Newcomb, S.R. Kimber, W.E. Newcomb, P.J. Information Processing – Hypermedia/Time-based Structuring Language (HyTime) - 2nd edition. *ISO/IEC JTC 1/SC 18 WG8 N1920rev*, May 1997.
10. Haus, G. Sametti, A. SCORESYNTH: a System for the Synthesis of Music Scores based on Petri Nets and a Music Algebra *IEEE Computer* 24, 7 (July 1991) 56-60.
11. Haus, G. Rodriguez, A. Formal Music Representation; a Case Study: the Model of Ravel's Bolero by Petri Nets in "Music Processing in *Computer Music and Digital Audio Series*, (Madison WI, 1993) G. Haus Editor, A-R Editions, 165-232.
12. Haus, G. Sametti, A. Modelling and Generating Musical Scores by Petri Nets in *Languages of Design*, 2,1 (Amsterdam, 1994) Elsevier Publ.,7-24.
13. Haus, G. Sametti, A. Score Analysis/Resynthesis Environment of the Intelligent Music Workstation in *Journal of New Music Research*, 24,3 (Amsterdam, 1995) Swets & Zeitlinger B.V., 230-246.

14. Haus, G. Rescuing La Scala's Audio Archives. *IEEE Computer, 31(3), pp.88-89, IEEE CS Press*, Washington, 1998.
15. Haus, G. Longari, M. Coding Music Information within a Multimedia Database by an Integrated Description Environment. *Atti del XII° Colloquio di Informatica Musicale*, AIMI/Università di Udine, Gorizia, 1998
16. Haus, G. Pelegrin, M. L. Music Processing Technologies for rescuing Music Archives at Teatro alla Scala and Bolshoi Theatre in *Journal of New Music Research*, (Amsterdam, 2001) Swets & Zeilinger.
17. MML Web site http://www.mmlxml.org
18. MusiCat Web site http://icarus.lib.virginia.edu/~pdr4h/musicat/
19. MusicML DTD site link http://www.oasis-open.org/cover/musicML-DTD.txt
20. MusicXML DTD web site http://www.recordare.com/xml.html
21. MuTaTeD http://www.pads.ahds.ac.uk/mutated
22. Narayan, K. Chiam, P. Wei, Y. Kok, L. Tin Wee Y. and T. Delivering Music Notation over the Internet: Syntax, Parser, Viewer and GUI-Input System. *Proc. of Inet'96,* http://irdu.nus.edu.sg/music/
23. Newcomb, S.R. Kipp, N.A. Newcomb, V.T. The HyTime. Hypermedia/Time-based Document Structur-ing Language. *Communication of the ACM 34:11*, November 1991.
24. Newcomb, S.R. Standard Music Description Language complies with hypermedia standard. *COMPUTER*, vol. X N.Y. 24:7, pp. 76-79, July 1991
25. Newcomb, S.R. Standard Music Description Language. ISO/IEC DIS 10743 ftp://ftp.ornl.gov/pub/sgml/WG8/SMDL/10743.ps, June 1995.
26. Roland, P. Extensible XML4MIR: Markup Language for Music Information Retrieval. *Proc. of Music IR 2000*
27. Selfridge-Field, E. Beyond MIDI The Handook of Musical Codes. *The MIT Press*, Cambridge, Massachusetts, 1997
28. Sloan, D. Aspect of Music Representation in HyTime/SMDL. *Computer Music Journal*, Cambrige, MIT Press 17:4, pp 51-59,Winter 1993
29. http://www.xrml.org/
30. http://www.oasis-open.org/committees/security/
31. http://www.oasis-open.org/committees/xacml/
32. http://www.w3.org/Signature/
33. http://www.w3.org/Encryption/2001/
34. http://www.w3.org/TR/SVG/
35. http://www.lim.dsi.unimi.it/IEEE/XML.htm
36. http://www.wedelmusic.org
37. http://www.oasis-open.org/cover/mnml199906.html
38. http://www.musique.umontreal.ca/personnel/Belkin/NIFF.doc.html
39. http://www.lim.dsi.unimi.it

MusicXML in Practice: Issues in Translation and Analysis

Michael Good
Recordare LLC
PO Box 3459
Los Altos, California 94024 USA
+1 650 965 7267
info@recordare.com

ABSTRACT
Since its introduction in 2000, MusicXML has become the most quickly adopted symbolic music interchange format since MIDI, with support by market and technology leaders in both music notation and music scanning. This paper introduces the key design concepts behind MusicXML, discusses some of the translation issues that have emerged in current commercial applications, and introduces the use of MusicXML together with XML Query for music analysis and information retrieval applications.

Keywords
MusicXML, music notation, music information retrieval, music analysis, XQuery, Finale, Dolet.

Copyright © 2002 Michael Good. All rights reserved.

1 INTRODUCTION TO MUSICXML

Many people have recognized the possible benefits of using XML for music representation [1, 2, 7]. Similarly, many efforts have been made over the years to come up with a higher-level interchange format for music notation than is provided by Standard MIDI Files [12, 16]. MusicXML combines these two trends, using XML technology to develop a new interchange standard for music notation.

MusicXML has been developed from a commercial perspective as opposed to a research perspective. Recordare's business model is founded on the commercial opportunities enabled by a standardized, Internet-friendly format for symbolic musical data. We needed to quickly demonstrate the viability of an interchange standard in the context of commercial musical applications. Our major technical risk was that we would develop yet another interchange format that could not or would not be adopted by the market leaders in music application software.

MusicXML Design Techniques

Previous efforts at music interchange standards have tended to fail in one of two ways. In the case of SMDL [14], the design goal was overly ambitious, leading to an overly complex language with very few implementations. In the case of NIFF [6], the design goal was insufficiently general. The graphics-oriented focus of the language made it adequate for scanners and notation programs with graphics-oriented formats. Sequencers, databases, and notation programs with more underlying awareness of musical semantics were ill-served by the graphical focus. A critical mass of application support never developed.

From our industrial perspective, we looked to MIDI and HTML as exemplars for developing a new music interchange standard. MIDI and HTML are both powerful enough to solve a good variety of industrial-strength problems. On the other hand, they are simple enough to learn and implement so that people can learn the basics easily, smoothly adding additional features over time.

XML 1.0 shares these dual characteristics of power and simplicity. XML's widespread adoption throughout the information technology industry allows music software to make use of the technology investments made by much larger industries.

Several techniques were used to develop a usable, useful, powerful interchange standard:

- The initial design of MusicXML was based on two of the most powerful music representation formats from academic research: MuseData [9] and Humdrum [10].
- The design and implementation of MusicXML was done iteratively, using an evolutionary delivery approach [4]. This ensured that the design elements were both implementable and useful across applications.
- Initial implementations focus on the market and technical leaders for major music applications, such as Finale and SharpEye Music Reader. By working with the most fully-featured products in their market areas, we could be confident that we were creating an industrial strength interchange language vs. a research prototype.

MusicXML Application Support

MusicXML's success is apparent from its quick adoption compared to any symbolic music interchange format since MIDI. Figure 1 shows the software products and projects supporting MusicXML as of July 2002 [11].

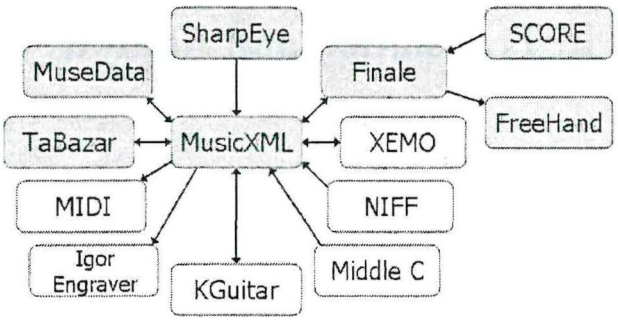


Figure 1: MusicXML Support as of July 2002

Finale 2003, SharpEye Music Reader, and TaBazar are all shipping with MusicXML support on Windows. Recordare's Dolet software supports MuseData and Finale 2000 to 2003. Finale 2003 can import SCORE files, allowing a two-step conversion into MusicXML. MusicXML files imported into Finale can be printed to the FreeHand MusicPad Pro electronic music stand, providing a new electronic format for MusicXML scores.

Project XEMO has demonstrated a Java-based MusicXML Notation Viewer, currently in alpha test, running on Windows, Macintosh OS X, Linux, and Solaris systems. Middle C Software has announced support for MusicXML in their future scanning software products. NoteHeads has announced plans to import MusicXML file in version 1.7 of their Igor Engraver notation product. The KGuitar open-source guitarist environment for Linux, FreeBSD, and Solaris systems added MusicXML support in version 0.4.1. The NIFF and MIDI converters have been developed as internal prototypes at Recordare.

MusicXML's successful adoption contrasts not only with past interchange languages, but also with other XML formats for symbolic music representation proposed in the past several years. Most of these formats cannot represent the full range of music possible in MusicXML. None has any commercial product support as of July 2002, much less support from music software industry leaders.

2 MUSICXML DESIGN ISSUES

MusicXML follows MuseData and other formats in separating underlying musical representation from the specifics of a particular engraving or music performance. As with MuseData, the three domains are combined within a single format. The "logical domain" of music is found in MusicXML's elements, while details of the visual and performance domain are found in MusicXML's attributes. There are also dedicated elements for `<print>` and `<sound>` where attributes associated with logical domain elements were not sufficient.

The integration of the three domains into a single format speaks to the need to cover an adequate range of music applications in a single notation format. The distinction between elements and attributes facilitates the segmentation of domains both for learning MusicXML and building applications. Distinctions between domains tend to be cleaner in theory than in practice. Given MusicXML's commercial focus, it made sense not to be overly rigorous about these theoretical distinctions.

To introduce how MusicXML represents musical scores, here is the musical equivalent of C's "hello, world" program for MusicXML. Here we will create about the simplest music file we can make: one instrument, one measure, and one note, a whole note on middle C:

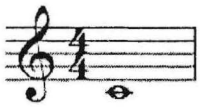


Figure 2: A Musical "Hello, World"

Here is the musical score represented in MusicXML:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC
  "-//Recordare//DTD MusicXML 0.6b Partwise//EN"
  "http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise>
  <part-list>
    <score-part id="P1">
      <part-name>Music</part-name>
    </score-part>
  </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>
        <clef>
          <sign>G</sign>
          <line>2</line>
        </clef>
      </attributes>
      <note>
        <pitch>
          <step>C</step>
          <octave>4</octave>
        </pitch>
        <duration>4</duration>
        <type>whole</type>
      </note>
    </measure>
  </part>
</score-partwise>
```

For scores of this simplicity, MusicXML's design roots are clearly apparent. This is basically an XML version of the MuseData representation.

Several of MusicXML design elements, including the interchangeability between partwise and timewise formats, have been described previously [5]. Here we will focus on some additional design aspects that have proven to be

important for music translation, and that look to be important for future work in musical analysis.

One key design choice is that each aspect of music semantics is represented in a different element. This provides the greatest flexibility for diverse music applications, especially once music information retrieval is included in the application mix. Our example analysis programs below will demonstrate some of the benefits of this design choice.

Another key design element carried over from MuseData is the importance of separately representing what is heard vs. what is notated [13]. Take the issue of note duration. MusicXML follows MIDI and MuseData by putting the denominator of music duration, the number of divisions per quarter note, in a separate, usually unchanging `<divisions>` element. The whole note is represented both in sound, as a `<duration>` of 4 divisions, and as a graphical `<type>` of a whole note. It is useful to have both, since notation programs work more easily with the graphical type, while sequencers work more easily with the duration values. In other cases, such as jazz, sounding duration is different than written duration, so both elements are required for an adequate representation of both musical sound and a musical score.

This type of dual representation of sound and graphics, so crucial to support diverse industrial applications, contrasts with the graphical representations used in NIFF and the WEDELMUSIC XML format [1]. NIFF is a binary format, but if we translate its binary elements directly into an XML document, our middle C whole note would look something like this:

```
<Notehead Code="note" Shape="2" StaffStep="-2">
   <Duration Numerator="1" Denominator="1"/>
</Notehead>
```

The `StaffStep` attribute tells us that the note is two staff steps, or one line, below the staff. But what is its sounding pitch? To determine that, we need to check the clef and key signature, handle any accidentals that preceded the note in this measure, look for any accidentals in a note that may be tied to this one, and interpret any 8va markings. This is a lot of computation for one of the two most basic elements of music notation: what pitch is sounding? Fortunately, the other basic element, the timing of the note, is represented much more directly.

The very indirect nature of pitch representation makes NIFF and other graphical formats unusable for most performance and analysis applications. It even makes for problems in its intended use in the visual transfer between scanning and notation applications. The NIFF importer included in Sibelius 2.11 has bugs that are directly correlated to missing one or more of the multitude of steps needed to accurately determine musical pitch from NIFF data. Graphical formats have a long history in music representation, and are appropriate as internal formats for many applications, but have severe problems when used as the foundation of a general music interchange format.

3 MUSICXML TRANSLATION ISSUES

When we first started building programs to move between MusicXML and other music formats, we called them converters. Conversion implies the centrality of the change from one format to another. We have since realized that a more productive metaphor might be that of translation; the interpretation of one form of human expression into another [15]. Our first software translation products are named after the 16th-century French translator Etienne Dolet.

At Recordare we have produced four MusicXML translators to date: two-way translators for Finale and MuseData, and one-way translators from NIFF and to Standard MIDI Files. Each translation brought up different issues of interpretation that need to be successfully addressed to make an effective interchange format.

MuseData Translation

Our MuseData translator was built together with the initial design of MusicXML. The first version of MusicXML was primarily an adaptation of the MuseData format into XML form, with the addition of a timewise format to simulate Humdrum's two-dimensional lattice structure within a hierarchical language.

Our design decision was to make MusicXML a superset of MuseData, so that we could do a 100% conversion (as we thought of it then) from MuseData into MusicXML and then back again. We adopted features in entirety even when we were unclear of their utility for a general-purpose translation language. Since the initial design, we have removed some of the documented features that are not used in practice, and included some undocumented features that are indeed used in MuseData files available from CCARH.

Given that MusicXML covers a superset of MuseData features, we encountered no major translation difficulties with this first piece of software. This gave us confidence that the XML language was indeed strong enough to serve as a basis for music representation without hidden problems that would only be revealed through implementation experience. The translation issues have emerged later, with more programs translating back and forth to MusicXML. These programs may make use of features that are present in MusicXML but not in MuseData, or may be used to translate classical repertoire from later eras than MuseData's design focus. For instance, translating pieces by Chopin, Mahler, and others with multiple large tuplets causes problems when the number of divisions needed for precise durations leads to notes with durations that cannot fit into a 3-character MuseData field.

NIFF and MIDI Translations

We next added translators for two binary formats: from NIFF to MusicXML and from MusicXML to Standard

MIDI Files. By testing translation with NIFF's highly graphical format and MIDI's performance-only format, we could determine whether MusicXML really did have the scope to handle a variety of music formats far afield from our MuseData and Humdrum starting points.

Building the NIFF translator is what gave us our more detailed understanding of the problems with highly graphical formats for music interchange. While our prototype works fine for testing purposes, it was clear that building an industrial-strength NIFF translator, while possible, would take a great deal of time and effort. We decided instead to persuade the author of the most commercially important application writing NIFF files (SharpEye Music Reader) to write MusicXML files as well. SharpEye was the first product to support the MusicXML format, and their implementation experience demonstrated that MusicXML could indeed be implemented successfully by third party developers.

Given the existence of MuseData to MIDI translators, we were not surprised when the MusicXML to MIDI translator posed no major challenges. An interesting aspect of both these translators is our use of XML as an intermediate format for both the NIFF and MIDI files. This creates an easier-to-program structure for these binary formats.

Finale Translation

Translating to and from Finale posed the largest challenges for MusicXML to date. As a fully-featured industrial application, it poses the expected challenges of dealing with a program whose feature set exceeds that of the interchange format. In many cases we added features that were necessary to support effective import from SharpEye to Finale (for instance, system and page breaks), but others are still unsupported.

The more interesting issues come from the differences in structure between Finale and MusicXML files. In many of the fundamentals, there is a great deal of similarity. Finale's underlying frame structure (a single measure on a single staff) has up to four layers, and each layer can have one or two voices. The layers and voices are similar to MusicXML's `<voice>` element. Moving between layers and voices are handled in Finale by means very similar to the `<backup>` and `<forward>` elements in MusicXML.

When we get to articulations and expressions, things work very differently in Finale and MusicXML. Finale is designed to be open-ended and extensible, so there are few of the bult-in abstractions present in MusicXML. These abstractions must instead be inferred from the definition of a musical symbol in the Finale database.

As an example, what Finale structure should translate to a `<staccato>` element in MusicXML? Is it:

- An articulation whose font character looks like a staccato dot, or
- An articulation with a performance definition that shortens the note length?

In practice we have found that the font definition works more reliably. Finale is a notation program, and people generally pay much more attention to appearance than playback within Finale files. But using this definition limits your translation to fonts that you have seen before, so that you know what musical glyphs are associated with each code in a music font.

Currently, the major barrier to even better Finale/MusicXML translations is the incomplete documentation for the Finale format provided in the current Finale 2000 plug-in developer's kit. This is akin to a translator working just by the context of word usage, with only an incomplete set of dictionaries for reference. Coda has suggested that this documentation may be improved in a future version of the developer's kit.

4 MUSICXML AND MUSICAL ANALYSIS

Music analysis and retrieval using large datasets of symbolic musical data has been hampered by the lack of an adequate, standardized format for symbolic music representation supported by commercial software tools. This gap makes it difficult to acquire and reuse either musical data or musical tools. The tools that are developed for music analysis research do not have the technical underpinnings to scale up to large-scale commercial usage of music information retrieval. The need to use databases to build collections of symbolic music information is well understood [7], but the technology has been lacking.

Building scalable database systems is a costly undertaking. It makes more sense for music applications to leverage the investment of other, better-funded application areas such as electronic commerce, as long as that technology is adequate—not necessarily ideal—for the needs of musical applications

XML has the potential to finally break through the database barrier through the efforts of the World Wide Web Consortium's XML Query working group. The group's mission is "to provide flexible query facilities to extract data from real and virtual documents on the Web, therefore finally providing the needed interaction between the web world and the database world. Ultimately, collections of XML files will be accessed like databases." [18]

The current focus of the XML Query Working Group is the XQuery 1.0 language. Though this language is still a work-in-progress, available only in working draft form, there are already a dozen prototype implementations available for evaluation. These come both from major relational database vendors like Oracle and Microsoft as well as native XML database vendors like Software AG.

The combination of an XML language for music and an XML query language is not sufficient by itself to break

through the database barrier for music information retrieval. The two languages must be able to work together to solve musical problems. Early XQuery working drafts had significant problems in this area, lacking powerful facilities to deal with queries that combine aspects of sequence and hierarchy. These shortcomings have been addressed in the XQuery 1.0 working draft of April 30, 2002, and we have now been able to build our first interesting musical queries using XQuery and MusicXML.

Given XQuery's importance and scope, it is likely to be some time yet before the language definition is completed, issued as a W3C recommendation, and commercial tools made available for effective development of XQuery applications. Fortunately, for research purposes, many analysis applications can be developed effectively today with existing tools: the XML Document Object Model (DOM) [17] and the XML Path Language 1.0 (XPath) [3].

Musical analysis is not just applicable in musicological research; it can also be useful in music publishing. For instance, as Recordare publishes its editions of classical art songs, it is helpful to show the range of each song. This process can be automated by a musical analysis program working on the MusicXML data. Figure 3 shows a screen shot from a program that generates a distribution graph of the pitch range for any particular part in a piece of music. Here we are computing the range for the voice part of the last song in Schumann's *Frauenliebe und Leben*, Op. 42.

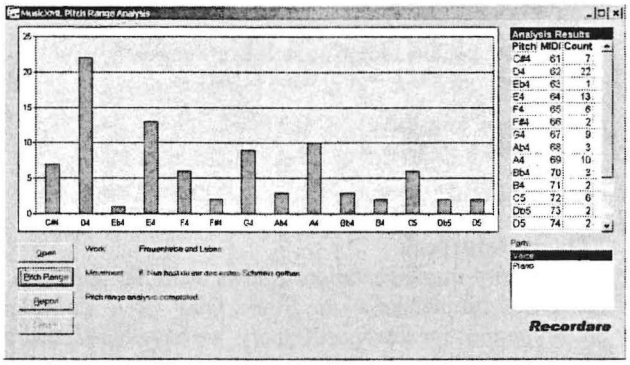


Figure 3: Pitch Range Distribution Analysis Program

Figure 4 shows the synopsis produced by clicking on the "Report" button. It focuses on the low and high notes.

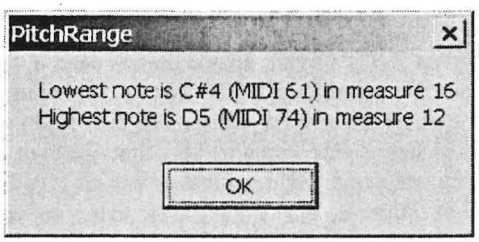


Figure 4: Pitch Range Synopsis Report

The program that generates this synopsis report is easy to write in MusicXML. For comparison, we will show two implementations. The first uses the DOM, programmed in Visual Basic 6.0 with Microsoft's MSXML3 parser. An equivalent program can be built using XQuery. Our example uses the QuiP 2.1.1 prototype program from Software AG, which is based on the April 30 working draft of XQuery 1.0. QuiP and XQuery are both works in progress, so the syntax of a working program is likely to change by the time XQuery becomes a formal recommendation from the World Wide Web Consortium.

DOM Approach

The DOM approach is implemented within a function that takes a MusicXML document and MusicXML part ID as input, and returns the dialog box string as output. After the initial variable declaration and initialization, the variable `oNodes` is assigned to all the `<pitch>` elements within the `<part>` specified by the PartID parameter. The selection is made using XPath 1.0 syntax.

The program then loops through each pitch, calling the `MIDINote` function to compute the MIDI note value from the different components of the `<pitch>` element. If the resulting pitch is lower or higher than any seen before, the spelling of the note is saved in a variable, using a separate `SpellNote` function on the same `<pitch>` element. The measure containing the extreme pitch also saved.

After all the pitches are searched, the program returns a string composed from the saved values for the lowest and highest MIDI pitches, along with their musical spellings and the measure where they were first encountered.

```
Function FindRange _
  (ThisXML As DOMDocument30, _
   ByVal PartID As String)

Dim oRoot As IXMLDOMElement ' Root of XML document
Dim oNodes As IXMLDOMNodeList ' Pitches to analyze
Dim oElement As IXMLDOMElement ' Current pitch
Dim oMeasure As IXMLDOMElement ' Parent measure
Dim lPitch As Long          ' Current pitch
Dim lMinPitch As Long       ' Lowest MIDI pitch
Dim sMinPitch As String     ' Spelling of low pitch
Dim lMaxPitch As Long       ' Highest MIDI pitch
Dim sMaxPitch As String     ' Spelling of high pitch
Dim sMinMeasure As String ' Measure for low pitch
Dim sMaxMeasure As String ' Measure for high pitch

lMinPitch = 128
lMaxPitch = -1

Set oRoot = moXML.documentElement
Set oNodes = _
    oRoot.selectNodes( _
       "//part[@id='" & PartID & "']//pitch")

' Search each pitch for the lowest and highest
' values, saving the spelling and measure number.
Do
  Set oElement = oNodes.nextNode
  If oElement Is Nothing Then Exit Do
  lPitch = MIDINote(oElement)
  If lPitch < lMinPitch Then
```

```
    lMinPitch = lPitch
    sMinPitch = SpellNote(oElement)
    Set oMeasure = _
      oElement.selectSingleNode _
       ("ancestor::measure")
    sMinMeasure = _
      oMeasure.getAttribute("number")
  End If
  If lPitch > lMaxPitch Then
    lMaxPitch = lPitch
    sMaxPitch = SpellNote(oElement)
    Set oMeasure = _
      oElement.selectSingleNode _
       ("ancestor::measure")
    sMaxMeasure = _
      oMeasure.getAttribute("number")
  End If
Loop

FindRange = "Lowest note is " & sMinPitch & _
  " (MIDI " & lMinPitch & _
  ") in measure " & sMinMeasure & vbCrLf & _
  "Highest note is " & sMaxPitch & _
  " (MIDI " & lMaxPitch & _
  ") in measure " & sMaxMeasure

End Function
```

The `MIDINote` function builds the MIDI note number by reading the `<octave>`, `<step>`, and `<alter>` elements in turn to build the note number value. The `CLng` function called here casts the string returned by the XML Element into a 32-bit integer (the `Long` type in Visual Basic 6.0).

```
' Return MIDI note value from a MusicXML pitch
' element, ignoring microtones.

Function MIDINote _
  (ThisPitch As IXMLDOMElement) As Long
  Dim oElement As MSXML2.IXMLDOMElement
  Dim lTemp As Long        ' Temporary pitch

  ' Get octave
  Set oElement = _
    ThisPitch.selectSingleNode("octave")
  lTemp = 12 * (CLng(oElement.Text) + 1)
  ' Get pitch step
  Set oElement = _
    ThisPitch.selectSingleNode("step")
  Select Case oElement.Text
    Case "a", "A": lTemp = lTemp + 9
    Case "b", "B": lTemp = lTemp + 11
    Case "c", "C": lTemp = lTemp + 0
    Case "d", "D": lTemp = lTemp + 2
    Case "e", "E": lTemp = lTemp + 4
    Case "f", "F": lTemp = lTemp + 5
    Case "g", "G": lTemp = lTemp + 7
  End Select
  ' Get alteration if any
  Set oElement = _
    ThisPitch.selectSingleNode("alter")
  If Not oElement Is Nothing Then
    lTemp = lTemp + CLng(oElement.Text)
  End If

' Assign and exit
  MIDINote = lTemp

End Function
```

The `SpellNote` function is even more straightforward, as the only conversion that needs to be done is to go from the numeric `<alter>` value to a text symbol for the sharps and flats in the note spelling.

```
' Spell the pitch as a string, e.g. "C#4"

Function SpellNote _
  (ThisPitch As IXMLDOMElement) As String

  Dim oElement As IXMLDOMElement
  Dim sSpell As String  ' Temporary string
  Dim sAlter As String  ' Alteration string

  ' Get pitch step
  Set oElement = _
    ThisPitch.selectSingleNode("step")
  sSpell = UCase$(oElement.Text)

  ' Get alteration if any
  Set oElement = _
    ThisPitch.selectSingleNode("alter")
  If Not oElement Is Nothing Then
    Select Case CLng(oElement.Text)
      Case -2: sAlter = "bb"
      Case -1: sAlter = "b"
      Case 0: sAlter = vbNullString
      Case 1: sAlter = "#"
      Case 2: sAlter = "##"
      Case Else
        sAlter = "(" & oElement.Text & ")"
    End Select
    sSpell = sSpell & sAlter
  End If

  ' Get octave
  Set oElement = _
    ThisPitch.selectSingleNode("octave")
  sSpell = sSpell & oElement.Text

  ' Assign and exit
  SpellNote = sSpell

End Function
```

XQuery Approach

Our XQuery implementation follows a similar approach to the DOM implementation. Since QuiP is a standalone prototype tool for learning XQuery, we have hardcoded the file name and part ID that were parameterized in the DOM example. This example takes a very simple approach to the query, reviewing all the pitches twice in order to locate the minimum and maximum values. Once we have these values, we then find the pitch elements whose MIDI note values match the high and low values. XQuery results are returned in XML format, so we do not need a `SpellNote` function. We simply output the first `<pitch>` elements that match each of the extreme values, and then find the number of the measure that contains the first instance of these matching elements. XQuery makes use of XPath 2.0 and does not support the `ancestor::` axis, so our query assumes the `<measure>` element is the grandparent of the `<pitch>` element. Therefore this query will only work with partwise MusicXML files, not timewise files. We have revised the syntax slightly to better match the XQuery

working draft, using the `string` function where QuiP 2.1.1 used the `string-value` function.

```
define function MIDINote(element $thispitch)
returns integer
{
  let $step := $thispitch/step
  let $alter :=
    if (empty($thispitch/alter)) then 0
    else if (string($thispitch/alter) =
             "1") then 1
    else if (string($thispitch/alter) =
             "-1") then -1
    else 0
  let $octave :=
    integer(string($thispitch/octave))
  let $pitchstep :=
    if (string($step) = "C") then 0
    else if (string($step) = "D") then 2
    else if (string($step) = "E") then 4
    else if (string($step) = "F") then 5
    else if (string($step) = "G") then 7
    else if (string($step) = "A") then 9
    else if (string($step) = "B") then 11
    else 0
  return 12 * ($octave + 1) + $pitchstep + $alter
}

let $doc := document("MusicXML/Frauenliebe8.xml")
let $part := $doc//part[./@id = "P1"]
let $highnote :=
  max(for $pitch in $part//pitch
      return MIDINote($pitch))
let $lownote :=
  min(for $pitch in $part//pitch
      return MIDINote($pitch))

let $highpitch :=
  $part//pitch[MIDINote(.) = $highnote]
let $lowpitch :=
  $part//pitch[MIDINote(.) = $lownote]
let $highmeas :=
  string($highpitch[1]/../../@number)
let $lowmeas :=
  string($lowpitch[1]/../../@number)

return
  <result>
    <low-note>{$lowpitch[1]}
      <measure>{$lowmeas}</measure>
    </low-note>
    <high-note>{$highpitch[1]}
      <measure>{$highmeas}</measure>
    </high-note>
  </result>
```

This query returns the following result in XML:

```
<?xml version="1.0"?>
<result>
  <low-note>
    <pitch>
       <step>C</step>
       <alter>1</alter>
       <octave>4</octave>
    </pitch>
    <measure>16</measure>
  </low-note>
  <high-note>
    <pitch>
       <step>D</step>
       <octave>5</octave>
    </pitch>
    <measure>12</measure>
  </high-note>
</result>
```

Melody retrieval provides a more typical XQuery example, using a FLWR (for-let-where-return) expression. Here we are looking for the instances of the *Frere Jacques* theme in the key of C. We simply this query to look just for the pitch step sequence of C, D, E, C. This query also assumes a partwise MusicXML file. It will match instances of the pitch sequence that cross `<measure>` boundaries, but will not match across `<part>` boundaries:

```
<result>
{let $doc :=
    document("MusicXML/frere-jacques.xml")
let $notes := $doc//note
for $note1 in
        $notes[string(./pitch/step) = "C"],
    $note2 in $notes[. follows $note1][1],
    $note3 in $notes[. follows $note2][1],
    $note4 in $notes[. follows $note3][1]
    let $meas1 := $note1/..
    let $part1 := $meas1/..
    let $part2 := $note2/../..
    let $part3 := $note3/../..
    let $part4 := $note4/../..
    where string($note2/pitch/step) = "D"
      and string($note3/pitch/step) = "E"
      and string($note4/pitch/step) = "C"
      and (string($part1/@id) =
           string($part2/@id))
      and (string($part2/@id) =
           string($part3/@id))
      and (string($part3/@id) =
           string($part4/@id))
    return
      <motif>
        {$note1/pitch} {$note2/pitch}
        {$note3/pitch} {$note4/pitch}
        <measure>{$meas1/@number}</measure>
        <part>{$part1/@id}</part>
      </motif>
}
</result>
```

When run against a simple three-part round of *Frere Jacques* prepared in Finale and exported to MusicXML, the query returns six instances of the motif, the first of which is shown below:

```
<?xml version="1.0"?>
  <result>
    <motif>
      <pitch>
        <step>C</step>
        <octave>5</octave>
      </pitch>
      <pitch>
        <step>D</step>
        <octave>5</octave>
      </pitch>
      <pitch>
        <step>E</step>
        <octave>5</octave>
      </pitch>
```

```
      <pitch>
        <step>C</step>
        <octave>5</octave>
      </pitch>
      <measure number="1" />
      <part id="P1" />
    </motif>
    <motif>
    <!-- Remaining 5 motifs removed
         for brevity -->
  </result>
```

5 CONCLUSION

MusicXML has built on the collective work of the XML and music representation communities to become the most widely adopted symbolic music interchange format since MIDI. As the language develops, it will encounter further challenges in the areas of translation and analysis. Our commercial experience to date bodes well for handling translation issues as MusicXML expands to include more data for tablature, percussion notation, and sequencer applications. Our recent XQuery experience gives us new hope that industry-standard XML database tools, combined with MusicXML-based representations, will provide powerful new tools for problems in musical data analysis and information retrieval.

ACKNOWLEDGEMENTS

Eleanor Selfridge-Field, Walter B. Hewlett, Barry Vercoe, and David Huron provided valuable advice and encouragement, along with their outstanding prior work in music representation. Graham Jones, Ian Carter, Jane Singer, William Will, and Craig Sapp were especially helpful during the Dolet for Finale beta test.

REFERENCES

1. Bellini, P. and Nesi, P. WEDELMUSIC format: An XML music notation format for emerging applications. In *Proc. First International Conference on WEB Delivering of Music* (Florence, November 2001), IEEE, 79-86.

2. Castan, G., Good, M. and Roland, P. Extensible markup language (XML) for music applications: An introduction. In *The Virtual Score: Representation, Retrieval, Restoration*, ed. W. B. Hewlett and E. Selfridge-Field (Cambridge, MA, 2001), MIT Press, 95-102.

3. Clark, J. and DeRose, S., eds. XML Path Language (XPath) Version 1.0. World Wide Web Consortium Recommendation, November 16, 1999. http://www.w3.org/TR/1999/REC-xpath-19991116.

4. Gilb, T. *Principles of Software Engineering Management*. Reading, MA: Addison-Wesley, 1988.

5. Good, M. MusicXML for Notation and Analysis. In *The Virtual Score*, ed. W. B. Hewlett and E. Selfridge-Field (Cambridge, MA, 2001), MIT Press, 113-124.

6. Grande, C. The Notation Interchange File Format: A Windows-compliant approach. In *Beyond MIDI*, ed. E. Selfridge-Field (Cambridge, MA, 1997), MIT Press, 491-512.

7. Haus, G. and Longari, M. Music information description by mark-up languages within DB-Web applications. In *Proc. First International Conference on WEB Delivering of Music* (Florence, November 2001), IEEE, 71-78.

8. Haus, G. and Pollastri, E. An audio front end for query-by-humming systems. In *Proc. 2nd Annual International Symposium on Music Information Retrieval* (Bloomington, IN, October 2001), 65-72.

9. Hewlett, W. B. MuseData: Multipurpose representation. In *Beyond MIDI*, ed. E. Selfridge-Field (Cambridge, MA, 1997), MIT Press, 402-447.

10. Huron, D. Humdrum and Kern: Selective feature encoding. In *Beyond MIDI*, ed. E. Selfridge-Field (Cambridge, MA, 1997), MIT Press, 375-401.

11. MusicXML Software page. http://www.musicxml.org/software.html.

12. Selfridge-Field, E. *Beyond MIDI: The Handbook of Musical Codes*. Cambridge, MA: MIT Press, 1997.

13. Selfridge-Field, E., Hewlett, W. B., and Sapp, C. S. Data models for virtual distribution of musical scores. In *Proc. First International Conference on WEB Delivering of Music* (Florence, November 2001), IEEE, 62-70.

14. Sloan, Donald. HyTime and Standard Music Description Language: A document-description approach. In *Beyond MIDI*, ed. E. Selfridge-Field, (Cambridge, MA, 1997), MIT Press, 469-490.

15. Steiner, G. *After Babel: Aspects of language and translation*, 3rd edition (Oxford, UK, 1998), Oxford University Press.

16. *The Complete MIDI 1.0 Detailed Specification.* Document version 96.1. Los Angeles, 1997, The MIDI Manufacturers Association.

17. Wood, L. et al., eds. Document Object Model (DOM) Level 1 Specification, Version 1.0. World Wide Web Consortuum recommendation, October 1, 1998. http://www.w3.org/TR/REC-DOM-Level-1/.

18. World Wide Web Consortium XQuery web site: http://www.w3.org/XML/Query.

The Music Encoding Initiative (MEI)

Perry Roland
Digital Library Research & Development Group
Alderman Library, P.O. Box 400149
University of Virginia
Charlottesville, VA 22903-4149 USA
+1 434 982 2702
pdr4h@virginia.edu

ABSTRACT
This paper draws parallels between the Text Encoding Initiative (TEI) and the proposed Music Encoding Initiative (MEI), reviews existing design principles for music representations, and describes an eXtensible Markup Language (XML) document type definition (DTD) for modeling music notation which attempts to incorporate those principles.

Keywords
Text Encoding Initiative (TEI), Music Encoding Initiative (MEI), music notation, music representation, eXtensible Markup Language (XML), Document Type Definition (DTD) design

1 INTRODUCTION

The primary goal of the Text Encoding Initiative (TEI) was the creation of a comprehensive yet extensible standard for the encoding and transmission of textual documents in electronic form. However, the project's goals went beyond simple transcription. While not required to create a printed version of the text, elements that allowed authors to encode descriptive and interpretative information were seen as essential for computer-assisted analysis of an encoded text. The success of the TEI endeavor is demonstrated by the large and varied body of texts which have been encoded, many of which are available free of charge via the World Wide Web.

The need for a comprehensive standard for music has long been recognized. And now the possibility of a Music Encoding Initiative (MEI) is tantalizingly close. Since the beginning of TEI in 1987, SGML and its new incarnation XML have grown in popularity. The development of HTML has made the creation and rapid growth of the World Wide Web possible. Domain-specific markup languages for both text and data, such as ThML for theological texts and Scalable Vector Graphics (SVG) for images, have been created in abundance. Yet no clear progress has been made in the encoding of music. Why is this?

2 SCOPE OF THE MEI

The first difficulty with advancing toward a TEI-like standard for music is identifying the proper scope of the encoding scheme.

In attempting to deal with this problem, the Standard Music Description Language (SMDL) chose to provide a markup language for any music from any locale and of any time. Despite achieving the status of an international standard, SMDL has not gained widespread acceptance, however, because it defines the term "music" much too broadly to effect a practical solution.

At first glance the TEI appears to have an overly broad scope as well. Text, just like music, has logical, visual, aural, and analytical aspects. However, the scope of the TEI is limited in a subtle yet significant way. The TEI was not designed to be the "human expression encoding initiative". Instead it is primarily concerned with the markup of those expressions which are, or can be, expressed as text; that is, those expressions which take a *written* form. Furthermore, the TEI is mute regarding the "proper" way to compose text. Even when texts are initially created using the TEI DTD, they are still essentially transcriptions of an ur-text.

Similarly, the MEI does not attempt to encode all musical expression, but instead limits itself to the *written* form of music, i.e. common music notation (CMN). Like the TEI, the MEI must also remain unconcerned with how music is created. It is not primarily an aid to musical composition just as the TEI does not function as an aid in the creation of text.

Some may see the adoption of CMN as the basis for encoding as too limiting. Legitimate arguments could be made for an entirely new form of music notation for the

0-7695-1864-8/02 $17.00 * 2002 IEEE

purpose of electronic transcription. However, common music notation is applicable to a wide range of contemporary and, perhaps more importantly, historical music. It has been eloquently described by Selfridge-Field as "the cornerstone of all efforts to preserve a sense of the musical present for other and later performers and listeners" [9]. Given its expressiveness, extensibility, nearly universal usage, and longevity, there seems to be little reason not to adopt CMN as the starting point for the MEI.

The fact that the MEI fundamentally conceives of music as notation does not limit its usefulness for encoding performance and analytical information. While it cannot rival a human rendition, a basic performance suitable for many purposes may be mechanically derived from the notation. Of course, any additional information necessary to complete this process may also be encoded. Likewise, descriptive and critical information may be included to assist bibliographic and analytical applications.

Ultimately, a limited scope makes the design of a representation easier. For example, both the pitch and rhythm models can be greatly simplified when non-CMN requirements are not considered.

3 MEI DESIGN PRINCIPLES

The MEI relies upon basic design principles already identified as important for music representations.

Comprehensive

Simple codes which represent only the information necessary for a particular application seem to be more efficient because they require less development effort and less complex processing software; however, a comprehensive code is better able to capture the interdependency of the elements within a score. Also, because the simple encoding method requires a new encoding for each application, a comprehensive coding scheme is "more conducive to the establishment of a permanent data base of encoded musical scores, and to the ultimate prevention of duplicated effort" [11].

Declarative

Declarative representations, i.e. those which state knowledge *about* something, are preferred over procedural ones, i.e. those which state *how to do* something. Declarative knowledge can be examined and combined while procedural knowledge tends to be inaccessible. Declarative representations are more modular because they limit interactions between separate entities. In addition, knowledge can be added to a declarative representation easily while procedural representations allow addition only by modification followed by a required debugging process [2].

Explicit

In so far as possible, all relations and knowledge should be explicitly stated in the representation [2]. When a representation is declarative and explicit, it is naturally static. This is not to say that the representation cannot change. Indeed, the encoding may be the end result of a process, but the encoding does not represent the process, only the result.

Interpreted

In order to encode something, one must first determine what is to be represented by the encoding. The resulting encoding is an interpretation of the thing to be encoded [4]. In other words, whenever an attempt to assign meaning is made, interpretation occurs. A well-designed representation acknowledges this truism and allows one to make interpretations explicit.

Hierarchical

Many representation schemes have tended toward one of two extremes: encoding scores as a collection of notes or as a single entity without further division. However, there are many musically important structures that fall between these two extremes. The existence of multiple levels of structure implies the need for hierarchical representation.

There are several benefits in representing musical notation hierarchically. In a hierarchical representation, individual components are isolated, making it possible to limit interactions between components and to specify the scope of operators that act on them. In addition, this kind of structure allows any component of a score to be treated in exactly the same manner as any other, regardless of its size or position in the hierarchy. In other words, a hierarchical data structure is object-oriented. However, unnecessary complexity is not introduced by using a hierarchical structure instead of the list structure commonly used to represent music. A list structure can always be represented hierarchically if desired, i.e. by a tree having only 1 level [1].

Formal

A music representation should be as formal as possible. That is, the ability to prove the correctness of the encoding should not depend on knowledge outside the formal definition [2]. While it cannot model semantics, at least a DTD formally declares the syntax of the representation. A representation based on a formal grammar like that embodied in a DTD can describe a broad range of music. In fact, any music that can be segmented can be described by a grammar. Grammars are used extensively in a variety of disciplines. Therefore, a great deal of software, including parsers and compilers, has been developed around them [7].

Instead of attempting to accommodate archaic and incorrect practice, the MEI should continue efforts to modernize and standardize notational practices like those contained in Read [6].

Flexible

While a music representation should have a standardizing effect on the corpus for which it designed, it should not

strive to eliminate all variation. It must remain flexible enough to accommodate minor variations in the source material. In addition, much of the encoding should be made optional so that the encoder is not required to mark up things with which he is not concerned [4]. However, increased flexibility must be carefully weighed against an inevitable corresponding decrease in standardization.

Extensible

No representation can guarantee that it can be used for all future artifacts or anticipate all of its own future uses, especially if the representation is the result of a solitary effort. Therefore, extensibility is required. In addition, it may be necessary to extend the representation in order to resolve ambiguity already latent in the scheme [4].

Other principles

Additional desirable qualities -- unique, mnemonic, consistent, non-cryptic, non-context-dependent, and idiomatic -- have been described [4]. However, most of these qualities come for free with the adoption of XML as a basis for the representation. Therefore, they will not be discussed at length; however, these qualities will figure prominently in the evaluation of the DTD once it is completed.

4 DTD IMPLEMENTATION

It is not enough to simply identify desirable characteristics without putting them into practice. This section describes how the MEI DTD attempts to incorporate the design principles enumerated above.

Why not use an XML schema?

The choice to create an XML DTD instead of a schema is based on several factors. First, DTDs are a stable part of XML technology and are widely supported in XML software. Second, competing schema proposals make it difficult to predict when schemas will achieve the same widespread support. Third, and most importantly, because it does not include support for external entities, a schema is difficult to extend without completely rewriting it. The lack of external entities makes a schema, as well as markup instances, difficult to modularize. The sharing of markup fragments has valuable potential, in the creation of an album of musical compositions, for example. In short, the usefulness and longevity of schema are severely limited.

Elements versus Attributes

The decision regarding when to use elements and when to use attributes is more than a stylistic one. Each construct has advantages [3]. Elements are more useful when the encoded data requires structure, when it may have more than one value at a time, or when the data should make sense with the markup removed. Attributes, however, can be constrained by type and value and their values may be defaulted.

Rather than choosing to use one or the other exclusively, a moderate approach that takes advantage of the strengths of each is the best course. A mix of elements and attributes can help to illuminate the distinction between data that is structural, i.e. an object such as a note, and data that is a property or characteristic of the object, i.e. the pitch of the note. This kind of separation can prove valuable for later processing, but cannot be achieved using an all-element approach.

There are additional advantages to using attributes where possible. Because attribute values can be defaulted, using them instead of elements can eliminate unnecessary redundancy in the data and reduce the size of the encoded file, an important consideration in data transport. Attributes also make it easier to enforce one-to-one relationships, i.e. a single note can only have one pitch designation. In addition, since attributes may be constrained by value, their use encourages the creation of standardized markup.

Furthermore, attributes offer several processing advantages. Since attributes never contain structure, their use can make it possible to have the same element hierarchy to some arbitrary level in the hierarchy, leading to a tree that is more balanced and, therefore, to more efficient processing by down-stream applications such as parsers. Attributes are also easier to access by XML applications because they are immediately available within the context of the element. Since accessing attributes does not require iterative and recursive processing, unlike embedded elements, using them may result in faster processing.

"Milestone" elements

Since music notation contains many examples of multiple hierarchies, i.e. a beam that begins in one measure and ends in the next, it is tempting to use processing instructions or "milestone" elements with empty content models to represent them:

```
<measure>
      <note />
      <?beambegin ?>
      <note />
</measure>
<measure>
      <note />
      <?beamend ?>
      <note />
</measure>
```

Example 1

There are significant disadvantages to this "pseudo-container" approach, however [10]. First, XML applications do not usually support processing instructions. Even if processing instructions were supported, however, it would be impossible to know exactly where the beam begins and ends. Does it begin with the first or second note in the first measure? We could *assume* that the beambegin

instruction precedes the first note lying under the beam, but this is only an assumption. In other words, this approach violates the principle that an encoding be explicit. Second, both constructs contradict the efficient elements-for-objects design described above. Last, the use of "milestone" elements is prone to error. It would be very difficult for a processing application to recover if the beamend construct were mistakenly omitted.

As seen in the example below, an element with a required participant list provides a better, that is, more XML-idiomatic, solution.

```
<measure>
      <note id="n1" />
      <beam corresp="n2 n3" />
      <note id="n2" />
</measure>
<measure>
      <note id="n3" />
      <note id="n4" />
</measure>
```

Example 2

Flexibility

For the purpose of this discussion flexibility is defined as the ability to create restricted arrangements of the data to fit a specific purpose.

The adoption of an exchange DTD which is large and loosely structured does not preclude the creation of customized DTD subsets which are smaller, more tightly structured, and easier to learn and implement. In fact, for the DTD to be effective it must actually encourage this kind of use.

The MEI DTD attempts to achieve flexibility via several techniques. First, sub-element order is not prescribed in many element content models. Of course, attributes are never ordered. Second, many attributes have implied values, meaning they can be left out of an encoding when not needed. Next, many elements may be classed via their type attribute according to their role, effectively allowing them to have user-defined semantics. In addition, both the element's content model and attribute list may be restricted via the m.[element name] and a.[element name] parameter entities. Furthermore, an element may be "switched off" through the use of a marked section. Finally, even the name of the element may be changed via the n.[element name] parameter entity.

```
<!ENTITY % n.beam "beam">
<!ENTITY %n.beam; "INCLUDE">
<![%beam;[
<!ENTITY % m.x.beam "">
<!ENTITY % a.x.beam "">
<!ENTITY % m.beam
"((%n.note;|%n.chord;|%n.pad;|%n.rest;)+
 %m.x.beam;))">
<!ENTITY % a.beam
"%a.common;
meiform       CDATA          #FIXED "beam"
breaksec      CDATA          #IMPLIED
rend          %beam.rend;    #IMPLIED
with          %beam.with;    #IMPLIED
%a.x.beam;">
<!ELEMENT %n.beam; %m.beam;>
<!ATTLIST %n.beam; %a.beam;>
]]>
```

Example 3

Extensibility

As mentioned earlier, extensibility must be a primary concern for any representation that hopes to have a shelf life beyond the completion of the design phase. The MEI DTD employs parameter entities that allow users to add extensions. In addition to showing how an element can be restricted, the DTD fragment above illustrates how an element's content model and its attribute list may be extended. The m.x.[element name] and a.x.[element name] parameter entities may be used to add new sub-elements and attributes. Attribute value lists may also be extended with parameter entities.

Utilization of existing standards

Because progress toward an encoding standard for music notation is much more feasible when not locked into constant re-invention of past wheels, large parts of the design of the MEI DTD are drawn from existing standards.

On the largest scale, the MEI is modeled upon the TEI. At lower levels, the Acoustical Society of America (ASA) system is used to record pitch information, performance-specific data is encoded using elements which have similar names and functions as those in the Musical Instrument Digital Interface (MIDI) standard, most of the mark up for text is designed to be familiar to users of HTML, and TEI header and Dublin Core elements form the basis of the meta-data components. Of course, the Unicode standard underlies the character encoding model for XML, obviating the need to re-invent special character encoding schemes. Finally, while it is not a formal standard, a well-known, authoritative source [6] has been used as the basis for the grammar for music notation parts of the MEI.

5 FURTHER EVALUATION

Up to this point, the design of the MEI DTD has been a solitary effort. Of course, much more evaluation, perhaps against the DTD analysis principles identified by Megginson [5], will be required before it can become an industry standard.

The latest version of the DTD and accompanying markup examples are publicly available [8]. Comments are welcomed.

REFERENCES

1. Buxton, William, et al. The Use of Hierarchy and Instance in a Data Structure for Computer Music in Foundations of Computer Music. Curtis Roads and John Strawn, eds. Cambridge, MA: MIT Press, 1985.
2. Desain, Peter and Henkjan Honing. Issues in the Representation of Time and Structure in Music in Music, Mind and Machine: Studies in Computer Music, Music Cognition and Artificial Intelligence. Amsterdam: Thesis Publishers, 1992.
3. Graves, Mark. Designing XML Databases. Upper Saddle River, NJ: Prentice Hall PTR, 2002.
4. Huron, David. Design Principles in Computer-based Music Representation in Computer Representations and Models of Music. Alan Marsden and Anthony Pople, eds. New York: Academic Press, 1992.
5. Megginson, David. Structuring XML Documents. Upper Saddle River, NJ: Prentice Hall PTR, 1998.
6. Read, Gardner. Music Notation: A Manual of Modern Practice. 2nd ed. New York: Taplinger, 1979.
7. Roads, Curtis. Grammars as Representations for Music in Foundations of Computer Music. Curtis Roads and John Stawn, eds. Cambridge, MA: MIT Press, 1985.
8. Roland, Perry. MEI WWW site: http://www.people.virginia.edu/~pdr4h/mei.
9. Selfridge-Field, Eleanor. Beyond MIDI: The Handbook of Musical Codes. Cambridge, MA: MIT Press, 1997.
10. St. Laurent, Simon. XML Elements of Style. New York: McGraw-Hill, 2000.
11. Wolff, Anthony B. Problems of Representation in Musical Computing. *Computers and the Humanities* 11 (1977), 3-11.

Demo Session

Actos: A Peer-to-Peer Application for the Retrieval of Encoded Music

Gustavo C. S. Frederico
School of Information Technology and Engineering
University of Ottawa
800 King Edward
Ottawa, ON, K1N 6N5, Canada
+1 613 562-5738
gfred006@site.uottawa.ca

ABSTRACT
This article describes Actos, a Peer-to-Peer application developed in Java that uses the JXTA open platform. Actos uses XML-based ChordML for the symbolic representation of music including chords, lyrics and metainformation. It employs XML-based ChordQL to perform chord sequence queries.

Keywords
Peer-to-Peer, music application of XML, music delivery

DESCRIPTION
Actos is a peer-to-peer (P2P) application that enables the retrieval of encoded music. It uses JXTA, a set of open and generic P2P protocols originally developed by SUN Microsystems.

It is very difficult today for the indexing of the existing content on the Internet to keep up to the pace of content creation. The mechanisms in place today, such as search engines and web crawlers, do not facilitate the real-time access to resources. P2P applications enable not only the retrieval resources in central repositories, but also in distributed peers. Also, the representation of music in open XML formats enables the proliferation of heterogeneous applications capable of manipulating the common formats, augments the user audience and, therefore, increases availability of resources. P2P applications suit well for such large user base and dynamic creation of resources.

Notorious P2P applications include Napster, that provides music file sharing, and Gnutella, that provides generic file sharing. However, the JXTA network platform brings a number of benefits in its support to Actos: it is an open specification independent of Operating System, programming language and networking platform. JXTA is divided into three main layers: core, services and application. Actos makes large use of JXTA's distributed file sharing services in the services layer. Also, the use of JXTA will enable Actos to collaborate with other future P2P applications for music retrieval that choose to use the same protocol.

Currently, Actos allows users to store music files encoded in ChordML. ChordML is an XML application that represents the chords, the lyrics and some metainformation of a music, such as title, author and key. The notation represents well jazz improvisations, popular music and contemporary religious music. It appeals to users and applications that do not need a "complete" notational representation.

User searches are performed against ChordML files, that live in the distributed network. There are two types of searches: text-based and chord sequences. Text-based searches match the lyrics, authors and titles. Chord sequence searches are performed using the ChordQL language. ChordQL, also an XML language, allows the users to express queries of chord sequences using the key names (A7, F#/A#, etc) or grades relative to the key, represented in roman numbers (I, ii, IV64, etc). Internally, Actos' chord query processor transforms the query to Regular Expressions using XSLT and applies it to each music instance returning either a "match" or "no match" answer. Besides performing exact chord matches, queries can contain attributes that instruct Actos' chord query processor to disregard chord inversions, major/minor modes or other chord suffixes in the notation.

Actos allows users to view ChordML with a web browser. For that, it employs an XSLT stylesheet to transform ChordML into HTML. Other output formats such as PDF or HTML with chord images can be used in the future. Other enhancements to Actos should also allow other music representation formats and query processors to be dynamically incorporated. Using open Java interfaces, other music and query formats could be "plugged in" on-the-fly.

0-7695-1864-8/02 $17.00 * 2002 IEEE

A case study: Prepared-Piano Notation in XML

Maurizio Longari
LIM-DSI
Università degli Studi di Milano
via Comelico, 39
20135 Milano, Italia
+39 02 5835 6296
longari@dsi.unimi.it

ABSTRACT
In this paper we want to show how easy it is to use the XML language to extend music notation with new features, and how to utilize them for the prepared-piano notation.

Keywords
Prepared-Piano, XML, Music Notation

1 INTRODUCTION

Research and experimentation of new sounds is one of the most interesting challenges of the century. Before the advent of computer music, many composers tried to find out new ways to play conventional instruments. Such practice required the codification of information not contemplated by the Western Common Notation (WCN). This led composers to define and use new proprietary, ad-hoc symbols in their scores. The XML language can help encode custom symbols and give them the meaning intended. As a case study, we want deal with the problem of encoding the symbols needed by the not well-known practice of piano preparation. We will study this notation with the help of Master Pino Devita (Prepared-Piano composer and performer [1]). In the next paragraphs, the process of piano preparation and execution is briefly described, followed by the presentation of a first draft to encode this information in XML syntax.

2 PIANO PREPARATION

A performance for prepared-piano consists of two phases: the preparation of the piano, and the actual performance with the proper tools.

The tools themselves can be subdivided in *preparation tools* and *performance tools*. A preparation tool is any tool applied to the body of the piano: nails, sticky tape, thin layers of plastic, copper or aluminum. These tools are applied to the strings of the piano to cover a range of notes defined by the composer. Performance tools are custom instruments, such as little plexiglass hammers, marimba hammers and aluminum sticks in several shapes.

The performance tools can be classified in three groups: Static, Dynamic and Mixed. For Static prepared-piano performance, only the first phase, namely, application of tools, is carried out and thereafter the piano is played as usual, letting it produce new sounds. In Dynamic prepared-piano performance, the first phase is optional and the piano is played directly on the strings with some performance tools. The Mixed modality combines Static and Dynamic, i.e. the piano is prepared and then played with both performance tools and with the keyboard.

3 PREPARED-PIANO NOTATION

The notation consists of a description of the preparation and of the symbols representing the various tools. These symbols are utilized in the score as if they where symbols of the WCN. In fig.1 we see an example of symbols. The problem of defining custom symbols is well addressed in the XML environment, since it is easy to extend the encoded notation by adding new symbols that have new meanings. In fig.2 and fig.4 we see an example of formalization of the information needed in the preparation phase. The shape of the new symbols can be easily modeled by using SVG [3] (Scalable Vector Graphics). In fig.3, the MusicXML [2] encoding of the third measure of the score is shown. An element is added to the score (pp_ref) for inclusion (reference) of the new symbol.

REFERENCES

[1] http://www.preparedpiano.org

[2] http://www.recordare.com/xml.html

[3] http://www.w3.org/TR/SVG/
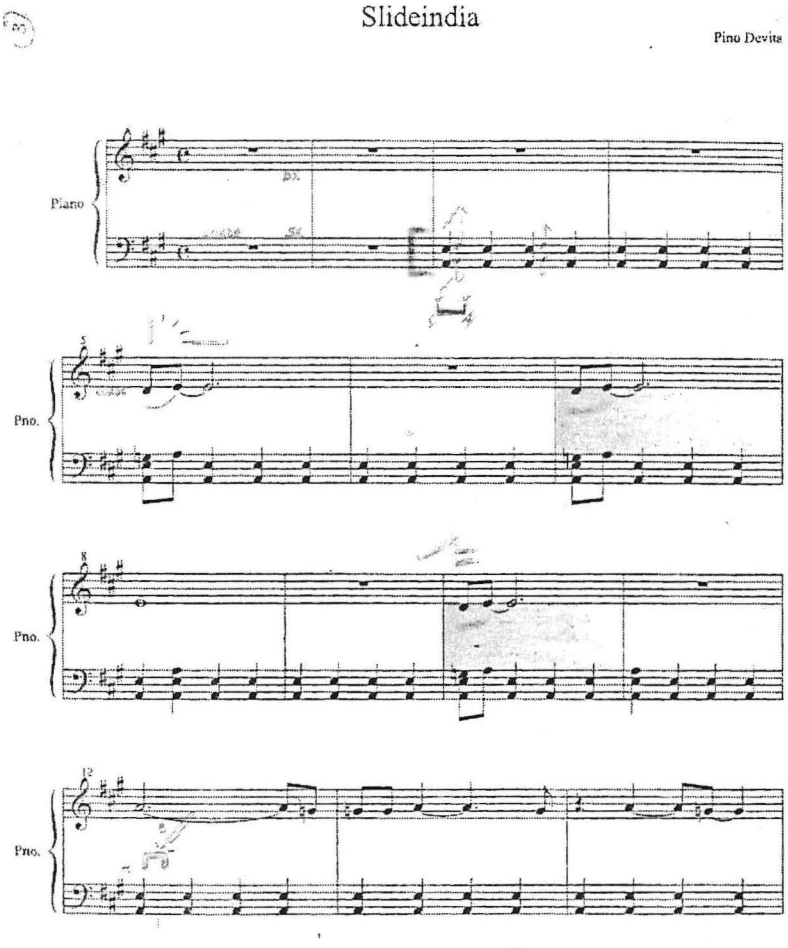


Figure 1: Example of a score for a Dynamic prepared piano performance.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Import of SVG Elements -->
<!ENTITY % svg PUBLIC "//PP" "svg.dtd">
%svg;
<!ELEMENT pp (preparation)>
<!ELEMENT preparation (#PCDATA | pp_symbol)*>
<!ATTLIST preparation
    type (Dynamic | Static | Mixed) "Static"
>
<!ELEMENT pp_symbol (#PCDATA | svg)*>
<!ATTLIST pp_symbol
    id ID #REQUIRED
    toolname CDATA #REQUIRED
    shape IDREF #REQUIRED
>
```

Figure 2: DTD for prepared piano information (pp.dtd)

```
<measure number="3">
    <note>
        <rest/>
        <duration>8</duration>
        <voice>1</voice>
        <staff>1</staff>
    </note>
    <backup>
        <duration>8</duration>
    </backup>
    <note>
        <pitch>
            <step>A</step>
            <octave>2</octave>
        </pitch>
        <duration>2</duration>
        <voice>2</voice>
        <type>quarter</type>
        <stem>up</stem>
        <staff>2</staff>
    </note>
    <note>
        <chord/>
        <pp_ref id="A" loc="below"/>
        <pitch>
            <step>E</step>
            <octave>3</octave>
        </pitch>
        <duration>2</duration>
        <voice>2</voice>
        <type>quarter</type>
        <stem>up</stem>
        <staff>2</staff>
    </note>
...
</measure>
```

Figure 3: Example of measure 3 in MusicXML extended for Prepared

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pp SYSTEM "pp.dtd">
<pp>
    <preparation type="Dynamic">
        <pp_symbol id="A"
            toolname="AluminumStick_IV"
            shape="Shape_AluminumStick_IV">
            This tool is utilized directly onto the piano strings
            playing the I and the relative V.
            Sometimes other tools can be stroked upon
            this tool to obtain a brilliant
            percussive sound.
            <svg id="Shape_AluminumStick_IV">
            </svg>
        </pp_symbol>
    </preparation>
</pp>
```

Figure 4: Example of definition of piano preparation elements.

Background Papers

Learning our Lessons from SMDL

Donald Sloan

Music Department
Ashland University
Ashland, OH 44805 (USA)
+1 419-289-5113
email dsloan@ashland.edu

Abstract.
The information standards HyTime and SMDL (ISO/IEC 10744 and ISO/IEC CD 10743, respectively) were created to handle representation and manipulation of hypermedia documents. HyTime was designed to handle any combination of media, while SMDL was designed as an application of HyTime specific to music information. In part, SMDL was intended to help those working with databases of musical information. While both HyTime and SMDL are based on document type definitions (DTD) of SGML, they never achieved the popularity of other markup languages such as HTML and XML. As researchers work towards making musical tools out of XML, it may be useful to review both the technical aspects of SMDL and the reasons why it never became a widely-used standard.

Keywords
SGML, HyTime, SMDL, music representation, markup

Introduction.
The main body of this paper was presented by this author at the X Colloquium on Music Informatics held in Milan in 1993. It is reproduced here to give the readers a sense of what HyTime and SMDL are, particularly from a technical point of view. Following this presentation is a discussion of what can be learned from this work.

About HyTime and SMDL
From 1986 to 1992, the X3V1.8M committee of the American National Standards Institute (ANSI) undertook a public standards process for the representation of static and dynamic information in hypertext and multimedia documents. This process has produced two related standards. Hypermedia/Time-based Structuring Language (HyTime) was approved by the International Standards Organization (ISO), of which ANSI is the United States representative, and assigned the number ISO/IEC 10744:1992. Standard Music Description Language (SMDL) was approved in a draft form, that bears the number ISO/IEC CD 10743, but was never fully implemented.

Both of these standards rely on an existing standard, Standard Generalized Markup Language, or SGML (ISO/IEC 8879:1986), a markup language which allows documents to be tagged in such a way as to allow representation of virtually any document architecture, by the representation of semantic and syntactic information. SGML is used today by many publishers world-wide, both in the public and private sectors. HyTime is an application of SGML; that is, HyTime relies on certain functions and capabilities outlined in the SGML standard. A HyTime user could design a Document Type Definition (DTD) based on SGML conventions, plus those parts of HyTime that are needed for that type of document. Many different instances of a given document type can use the same DTD. With HyTime, the objects (called elements in SGML) are defined in the DTD, while their actual attributes can vary from instance to instance. In a similar manner, SMDL is an application of HyTime. It uses a DTD that relies on HyTime's facilities, plus those unique to SMDL.
It is important to note that both HyTime and SMDL were designed as enabling standards. They place no restrictions on the content of what is represented, nor do they enforce a single kind of document architecture. By following the standards, one may gain those things that a standard should confer: a representation scheme that is public, a means for expressing the structure of information, and a set of tools for performing useful operations on the information, such as data queries and hypertext links. The standards do not constrain document styles; for example, two different publishers may use different formats for chapter headings. The tagging of a certain datum as a chapter heading is, however, an important piece of information, regardless of what a user may choose to do with it. This principle guided

the developers of HyTime and SMDL; a proper level of abstraction and an appropriate set of tools will allow any author to retain both the data content and the document structure in a useful way. These standards do not force an author to use all of the facilities of the standards. One can include a non-SGML document, and label it as such; one would lose much of the power of the standards, but at least the choice is up to the user.

HyTime and SMDL are intended for use by virtually the entire community of information processing, from paper to electronic publishers, from hypertext documents to multimedia documents, from authors to end users. It is hoped that developers will create a set of software tools similar to what now exists for SGML, so that the intricacies of HyTime and SMDL will be hidden behind an interface that allows a wide range of users to transparently author and read HyTime and SMDL documents. The HyTime standard can be obtained from national standards organizations, as well as from the ISO/IEC Copyright Office, Case Postale 56, CH-1211 Génève 20, Switzerland. Annex B of the HyTime standard lists additional sources of information. SMDL is still in draft form.

Why Two Standards?

Although this project began with only one standard in mind, it became clear that the requirements of the music community were far more detailed than those of the hypermedia community in general. HyTime contains a highly abstract approach to information architecture, consisting of several modules representing these functions: 1) a way to measure the position and extent of objects, 2) a way to address objects that cannot be given unique identifiers in SGML, 3) a way to link objects, whether or not in the same document, 4) a way to schedule events so that not only the proper sequence, but also the proper relationship between events is maintained, and 5) a way to render a particular instance of such a schedule. The scheduling and rendering modules owe a particular debt to the music world; it was determined that the scheduling model for musical events was at least as sophisticated as that for any other medium, and thus any model that could represent musical events was robust enough for virtually any schedule. The content of SMDL, therefore, contains only those items unique to music representation. The time model is already present from SMDL's inheritance of HyTime facilities, meaning that an SMDL document will invoke HyTime elements to represent durations. Those HyTime modules that are not applicable to music representation may be left out; there are different levels of conformance depending on user needs. Thus, if a user does not need hypertext linking facilities, this need not be included. This feature of modular architecture will make it easier for those using HyTime and SMDL for music applications.

HyTime'sTime Model.

HyTime uses its scheduling module to represent sequences of events, durations, and other temporal relationships between objects. In HyTime, a work may be represented by a number of coordinate axes, with each axis representing a temporal or spatial dimension. Each event has a position and extent on an axis. For temporal events, this corresponds to start time and duration. Thus, it may be possible to show the staging of an opera with a temporal axis showing the flow of the music, and two spatial axes showing where each character is on stage at any given moment of the music.

These axes together make up what is called a finite coordinate space, or FCS. The measurement system for each axis is defined by the DTD, and remains fixed for that work. An axis may use a standard measurement unit (SMU) such as meters, seconds, etc., or a virtual measurement unit, with its temporal or spatial meaning defined by the user. The latter system is what would be useful to represent music that is written in Common Western Notation. When musicians think of half notes and quarter notes, these do not have an absolute duration in seconds until the time of performance; even then, no two performances would be the same temporally. Thus, a logical representation in virtual units is what is required to properly capture the musical information.

While it is beyond the scope of this paper to give a tutorial extensive enough to teach fluency in HyTime scheduling, a certain amount of this can be shown in the following way: a potential DTD for an SMDL document could contain the following information:

```
<!DOCTYPE work SYSTEM "smdl.dtd" [
<?HyTime VERSION "ISO/IEC 10744:1992" HYQCNT=32 >
<?HyTime MODULE base
      desctxt dvlist lextype refctl >
<?HyTime MODULE measure
      axismdu dimref fcsmdu HyFunk HyOp markfun >
<?HyTime MODULE sched
      grpdex >
<?HyTime MODULE rend
      modify patch profun project >
<!NOTATION virtime PUBLIC -- virtual time --
           "+//ISO/IEC 10744//NOTATION Virtual Time Unit//EN" >
<!ENTITY tactvtu -- number of vtus per tactus (beat) -- "80640" >

<!ENTITY % av.wxdm  -- dimension of wfaxis -- "4294967295">
<!ENTITY % av.wxfm  -- mdu def for workfcs -- "'SIsecond 1 1000'" >
<!ENTITY % av.wxbg  -- basegran of wfaxis -- "msec" >
<!ENTITY % av.wxgh  --  gran2hmu of wfaxis -- "'1 1'" >
<!ENTITY % av.wxpg  --  pls2gran of wfaxis -- "'1 1'" >

<!ENTITY % av.cxdm  --  dimension of mustime axis --  "4294967295">
<!ENTITY % av.cxbg  --  basegran of mustime axis; --  "vtu" >
<!ENTITY % av.cxgh  --  gran2hmu of mustime axis --  "'1 1'" >
<!ENTITY % av.cxpg  --  pls2gran of mustime axis -- "'1 &tactvtu;'" >
<!ENTITY % av.cxfm  --  mdu def for mustime axis --   "'virtime 1 1'" >
```

The first line of code gives a name to the DTD. The next group of lines names those HyTime modules, and within each module, those functions which will be supported by this DTD. The notation is declared as virtual time, and the units of measurement virtual time units, or vtus. Each beat or tactus is given a total of 80640 vtus, which ensures that equal division of the beat into 2, 3, 4, 5, 6, 7, 8, 9, etc. units can be achieved with integer totals in each case. This number may change according to user needs; for example, there are certain Chopin works in which a beat is divided into 17 notes. In such a case, the number of vtus per tactus could be a multiple of 17 as well. This would guarantee an integer number of vtus for each note, avoiding the problems inherent in floating-point calculations. One may always arrive at a safe 'tactvtu' number by taking a common multiple of the various divisions of the beat in a given piece. The least common multiple would be most economical, of course, but since this DTD may serve several different pieces, it is probably safer to choose a larger mulitple that will not have to be adjusted from piece to piece.

The next five entities define a real time axis, upon which a performance of the musical work could be represented. Notice that the measurements are in seconds and milliseconds, real time units. The last five entities define a virtual time axis, upon which the temporal information inherent in the score would reside. This axis is defined in terms of virtual time units, or vtus. The user may declare in another part of the document what the tactus represents. In this way, a single DTD may be flexible enough to handle several different types of pieces, or even a single piece in which the tactus changes.

These lines of code could be used potentially as a typical SMDL DTD, thus obviating the need to redefine all of the HyTime terms for each work of music represented. There are other parts to this typical DTD that refer to resource tables and other SMDL devices; these have been omitted in this example in order to isolate those things relating directly to the time model, but would ordinarily be included in a typical SMDL DTD.

Since this DTD allows for an axis in virtual time, to capture the logical information, and an axis in real time, to capture a potential performance, there must be a method for mapping from one to the other. HyTime has elements in the rendition module called batons and wands to create this conversion. A baton is a schedule of projectors which can map virtual units to real units, with extensive facilities for determining how to reconcile one to the other. A wand is a schedule of modifiers which act upon HyTime objects. An example of a modifier would be a filter, either sonic or visual. The modifying abilities of these scopes could be defined by the user; for example, those interested in representing electronic music may find it useful to define scopes in terms of the formulas used to manipulate

waveforms, rather than have to name a specific pitch for a sound that is being processed in such a way as to make pitch difficult to define. HyTime does not provide a standard set of tools to describe sound manipulation; rather, it has a place in the standard for those who wish to describe their sound manipulation methods in a representation of their choosing.

From DARMS to SMDL...

The Digital Alternate Representation of Musical Scores, or DARMS, is one of several data content notations currently in use in the music world for the representation of music information. There exist musical databases encoded in DARMS, as well as other data content notations, such as SCORE, MUSTRAN, the CCARH code, MIDI and others. The inclusion of MIDI in this list is an illustration that not all musical information resides in scores; there is a need for capturing potential performances as well. Since humans use a musical score as a blueprint for performances, some people have believed that a data content notation that represents a score represents a performance as well. Any programmer can tell you that there is a good deal of interpretation that a machine must accomplish in order to render a score in actual sound. DARMS is one of those codes that captures the appearance of the musical symbols on a score, but requires an intelligence to give sonic definition to these symbols.

SMDL is being designed to cover both the visual aspects of music representation, such as scores, and the auditory aspects of music representation, such as gestural information for performances. There exist several domains for the information in an SMDL work. The set of logical information, that is, the temporal relationships, pitch information, and the like, is represented in a domain called the cantus. The purely visual phenomena such as how the musical information appears in a score, is in the visual domain. The purely audible phenomena, such as how the music is manifest in a certain performance, is in the gestural domain. Finally, manipulations that rely on the work's musical information, but are not properly part of the piece of music reside in an analytic domain. It is the cantus that contains that which uniquely defines the work; one could imagine taking the information in the cantus and creating several different-looking scores (e.g., two representations of Hebrew cantilation, one using the original tropes, and another in Western notation), or two different performances. And yet, in spite of the different ways of representing it, there remains only a single work of music in the cantus. This cantus represents the abstract information of a piece of music.

Thus, a DARMS-encoded score could be represented in SMDL both in the cantus, which would contain pitch names, articulations and relative note proportions, and in the visual domain, which would contain such purely visual phenomena as clefs, articulation symbols, beaming information, etc. For each symbol in DARMS (or any other existing coding scheme, for that matter), there will be a corresponding way to capture all of the musical information in SMDL.

The latter point is crucial if SMDL is to be useful as a standard. SMDL should be able to represent all that we regard as musical information. In certain cases, this representation may be nothing more than an ability to specify the extent of a chunk of data, and a label for what it is. For example, rather than convert or translate a DARMS database into SMDL data, one can include it in an SMDL document as an uninterpreted section of data called "DARMS code." This will not allow many operations that SMDL could otherwise perform on musical data, but at least would provide a standard way to exchange data with proper clues as to what the data content notation is. Similarly, a bitstream of sound samples may remain undifferentiated in an SMDL document, but bear the label "digital sound samples," so that a device that could potentially use this information will know what it is.

...And Back Again.

Since SMDL can represent at least as much information as would be in a DARMS-encoded score, conversion from SMDL to DARMS would be a matter of using only that information which could be captured by DARMS. If one used only the information present in the cantus of an SMDL work, then there is no guarantee that the resulting DARMS translation would be exactly like the original that was translated to SMDL in the first place. This is because the cantus contains information about the exact pitch, but not how it is represented on a page. Thus, in order to get from an SMDL work back to the DARMS code with which one started, one would have to use both the cantus and a visual domain together. Fortunately, SMDL has an element in the visual domain in which any datum in that domain can be referenced to its corresponding logical information in the cantus. Thus, one can be sure that an event in the cantus and an event in the visual domain refer to the same musical object, and can be used together to translate to the corresponding DARMS code. The link direction of such a reference is from the visual domain to the cantus, not the other way around, since there are potentially limitless ways to represent the logical information visually, but each visual symbol should refer uniquely to its logical information.

Why is it important to be able to translate from DARMS to SMDL, and vice versa? If one wanted to be able to share databases of musical information, without a standard, it would be necessary to create a pair of translators for each pairwise combination of codes. Thus, if there are ten codes commonly in use, a full translating capability would require 90 different translators. With a standard, one would only need a translator from each code to the standard, and back

again. The same ten codes could be fully shared with only 20 different translators. As representation standards proliferate, the need for translators would grow geometrically without a standard, but only arithmetically with a standard. Such a standard need not crowd out all other methods of representation; rather, it can be used only to the extent that it is valuable to do so. It has been suggested that using an already-existing code would save the trouble of developing a standard such as SMDL. Yet, it has not been demonstrated that any existing code can capture all of the information necessary for both musical score and musical performance.

What can be learned from SMDL?

Unfortunately, development of SMDL has not happened for a variety of reasons. The primary reason is the two-tiered nature of the standards. Since SMDL was designed as a DTD of HyTime, there necessarily had to be a waiting period where software tools and a kind of debugging of HyTime took place. By that time, those who were involved with the original project were heavily involved in trying to make HyTime commercially viable. Many parties in the music community were waiting to see what was developed in SMDL, but no one stepped forward to fill in the gaps and write the software necessary to entice people into adopting SMDL. Several software companies who had their own music information format (such as SCORE or ENIGMA) wanted their proprietary format adopted as a general standard. Thus SMDL has suffered not so much from technical deficiencies, but rather for other reasons.

Should the music community now be urged to reclaim SMDL and develop the appropriate tools to make it viable? That is still possible at this point, although there is no apparent catalyst that would get this process restarted. Instead, it may be up to those developing XML music applications to use the development ideas of HyTime/SMDL for a new set of tools. Since XML is a markup language, it can use many of the ideas of the earlier standards.

With that in mind, this author would like to suggest certain features of HyTime/SMDL that should be considered in any attempt to write an XML standard for music representation.

1) There should be a separation between the abstract musical information and the possible instances of its use. This is already obvious regarding performances, which are all unique, but still is not universally recognized in scores, where the actual layout is an application of the musical information, but is not the abstract information itself. The cantus in SMDL created this separation.

2) The time model of HyTime is sufficiently rich to capture any representation of time-based information. Both real-time and virtual-time representations should be available. In addition, there should be a mechanism whereby formulae may be used to represent how time unfolds in a piece of music.

3) It is difficult to be all things to all users. Should a general application show a bias towards Western Music Notation, or should it be equally usable for any kind of music and its notation? The most abstract forms of such an application can be generalized for all significant musical information, as HyTime/SMDL has shown. The specific applications can be written using this abstract shell, so that concepts such as tuning systems, normative scales and other notational issues can hook onto the meta-structure.

4) There needs to be tools to translate any music information from one format into a general standard, and vice versa. See the discussion of DARMS and SMDL above. In order for this to happen, the standard needs to be able to represent any musical information.

5) Any standard that is developed within the academic community should be focused on that community's needs. If the result becomes widely used, it may eventually find its way into the commercial world, but these commercial pressures can be harmful during the development of a standard..

Notation Interchange File Format NIFF 6b

1 INTRODUCTION

NIFF is a file format designed to allow the interchange of music notation data between and among music notation editing and publishing programs and music scanning programs. Its design is a result of combined input from many commercial music software developers, music publishers, and experienced music software users.

Background.

The lack of an accepted standard format for music notation has for years been a source of great frustration for computer musicians, engravers and publishers. Numerous attempts have been made in the past to create a standard format. The effort resulting in NIFF is different for several reasons:

1) Many of the major forces in the commercial music software industry - and several of the largest music publishers - have shown a remarkable willingness to cooperate in the design of NIFF.

2) Commercial music scanning programs are a recent addition to the software market. They require a common language with notation programs to avoid the enormous loss of detail that occurs when translating through MIDI files.

Music notation is a complex language. Like a natural language, it is always changing, it is at times ambiguous and/or redundant, and can be used in many different ways by different people for different purposes. Because of these qualities, it is impossible to create a perfect computer model for music notation.

The original sponsors of the NIFF project recognized these limitations, and set a more reasonable goal: to create a practical, useable format in a short time frame. NIFF project participants have agreed that a solid, workable solution, even if it excludes some unusual situations, is preferable to no solution.

The NIFF planners considered adopting as the standard an existing published format such as DARMS or Score, but decided against it. The designers of DARMS had somewhat different goals from NIFF's. Score, although extremely comprehensive and appropriate in scope, was somewhat unwieldy due to its age and development history. Another possibility considered was the ISO/ANSI standard HyTime and its associated music standard known as SMDL, but SMDL was not yet complete at the time the NIFF project got underway. [As of the first date of publication of this specification, a European Community music library project underway will result in a bridge between SMDL and NIFF.]

A general strategy was chosen as follows: model NIFF's feature set on Score's, organize the data as systematically as possible, and use the most current file format conventions. During the development of the format, additional more specific goals have been established. Some useful functional features of DARMS have been incorporated as well.

The NIFF structure can accommodate full music publishing systems, simpler music display systems, logical definition languages like DARMS, and music scanning programs. Even sequencers can be writers of NIFF files, since the most rudimentary NIFF file has little more notation information than a MIDI file.

NIFF allows representation of the most common situations occurring in conventional music notation, while making provision for software developers to define their own extensions to handle the more unusual situations. It allows inclusion of Encapsulated PostScript (EPS) files and fonts to allow interchange of features not otherwise defined in the format.

Extensibility, Flexibility, and Compactness

Extensibility is an important goal. Even the complete NIFF specification cannot be an exhaustive catalog of music notation features. A high priority has thus been given to structuring the data in ways that minimize the pain of future enhancements.

NIFF is also designed be flexible, to accommodate the differences between the programs and users it will serve.

An effort has been made to keep files as compact as possible, since NIFF files will likely be transmitted electronically over low-bandwidth lines.

Logical, Graphical and Performance (MIDI) Data

Researchers Severo Ornstein and John Maxwell found that the computer representation of music notation has three distinct components only partially related to each other: logical, graphical, and performance (MIDI) information[5]. NIFF designers have found it useful to further subdivide the graphical information into page layout and non-page layout information.

The only information that NIFF absolutely requires is the logical kind. NIFF is structured as a page-ordered format, but can accomodate writing programs without access to page layout information and systems like DARMS, which allows even non-page-layout graphical information such as stem direction to be absent.

When complete graphical information is present in a NIFF file, the reading program can decide between 1) observing the page layout and other positioning information, 2) ignoring graphical information in favor of its own defaults, or 3) some intelligent combination. When any graphical information is absent from a NIFF file, the reading program is expected to provide its own intelligent defaults.

It is recommended that the user of the NIFF writing and reading programs be given as much control as possible. The user should decide the level of detail stored in the file by the writing program, and the degree of freedom used in interpretation by the reading program.

NIFF allows thorough linking of MIDI data and notation.

Illegal Notation Elements
There is no such thing as an "illegal notation element" in NIFF. That is, there is no requirement for the data in a NIFF file to "make sense." A reading program should expect occasionally to find data that is not compatible with its own range of acceptable usage. When an unknown or unacceptable element is encountered, the program should either ignore it or do the best it can with it, in order to create a valid file in its own internal format.

RIFF
The NIFF format follows the design rules of the Resource Interchange File Format (RIFF) structure from Microsoft. In RIFF files, related data items are grouped into "chunks", and related chunks can be combined into "lists." Chunks and lists include their own length within their structure. RIFF files are designed to be upwardly compatible, that is, amenable to future enhancements.

A RIFF file and each of its lists and chunks can be variable in length. A logical definition is required for each RIFF format, to identify which elements are required or optional at each level and the order in which the elements may appear

In NIFF, an additional type of data item known as a "tag" has been defined as an integral part of the format. Tags are used for adding optional elements to the required part of a chunk. Although not part of the standard RIFF design, tags can be described within RIFF's logical definition language.

RIFF is a binary format, but it is possible to describe a complete RIFF file with an ASCII representation. A structured notation system for representing RIFF files in ASCII is presented in the Microsoft Windows Multimedia Programmer's Reference[4].

More details on the format of chunks, lists and tags is presented later in Chapter 1. Detailed rules on reading and writing RIFF files are published in the RIFF documentation[4].

Platform Independence
The same NIFF format definition can be used by software running on any type of machine. RIFF rules allow for separate formats for Intel (IBM compatible) and Motorola (Macintosh) machines, with different integer byte orders for the two machines. However, to make file translations across platforms easier for users, only the Motorola format will be used for NIFF, regardless of the machine used. The byte order is always from most significant to least significant, for both 16-bit and 32-bit integers, on all machines.

The first 4 bytes of a RIFF file indicate the byte ordering. 'R' 'I' 'F' 'X' means Motorola byte ordering, so all NIFF files will begin with these four characters. NIFF programs might therefore be required to translate integers from and to the correct byte order for their machine before reading and writing NIFF files.

More on Flexibility
NIFF is highly structured at its highest levels according to the rules of RIFF. However, writing programs have extremely wide latitude in their choice of what to include in any particular file, and how exactly to organize the data. A minimal NIFF file has little more information than a MIDI file. On the other end of the spectrum, an immense amount of detail can be recorded about the graphical arrangement of complex orchestral scores with custom shapes.

NIFF software developers will need to be creative when facing this range of possibilities. As with the TIFF (Tagged Image File Format) format, there will probably not be any two programs that make use of NIFF in exactly the same way.

2 LOGICAL STRUCTURES

There are some terms commonly found in discussions of music notation which most musicians intuitively understand. This section defines the specific usage of these terms in NIFF.

- Score. Each NIFF file contains one score. A score could hold anything from a one bar music example, to a song, a piano sonata movement, or a movement of a symphony in full score. A work in more than one movement would normally be stored with each movement in its own NIFF file.

- Part. A stream of musical events and associated symbols, text and graphics which can be extracted and printed on a part score for an individual performer. A part may contain music to be played sequentially on different related instruments by the same performer (e.g. oboe/English horn). The maximum number of staves used for display is defined for each part.

- Voice. A rhythmically independent stream of musical events within a part, and its associated symbols, text and graphics. Voices can appear and disappear at any time within a part.

- System. The visual framework on a page, where symbols representing simultaneous events in the various parts are more or less vertically aligned. All parts of the score are logically present in every system although some may be "hidden."

- Staff. An arrangement of parallel horizontal lines serving as the locale for displaying musical symbols. It is a sort of vessel within a system into which musical symbols can be "poured." Displayed on it are music symbols, text and graphics belonging to one or more parts.

- Time-Slice. The mechanism by which a specific point in time is identified in the score. Musical symbols representing simultaneous events.are logically grouped within the same time-slice. The music symbols in a NIFF file are physically grouped by page and staff, so symbols belonging to a common logical time-slice may be physically separated in the file. Horizontal placement can optionally be supplied for a time-slice.

There are two types of time-slice: measure start and event. The measure-start time-slice identifies the start time of a measure with respect to the start of the score. The event time-slice identifes the start time of an event such as a note or rest with respect to the start of the measure.

Examples.

The NIFF logical structures were designed to handle situations like the following:

Example 1. In a Mahler symphony score there are three trumpet parts. In one system the trumpets appear on three separate staves (labelled "Tpt. 1", "Tpt. 2", Tpt. 3"), because they are playing a complex canon. In another system they appear all together on one staff, called "Tpts. 1,2,3," because they are playing homophonic music. They are written as chords, with 3 notes on one stem.

In the logical view, the notes played by the trumpets belong to three separate parts.

In the physical view of the canonic system, each trumpet part is assigned its own staff. Each staff is labelled with its own name.

In the physical view of the homophonic system, the three parts are combined together onto a single staff, labelled "Tpts. 1, 2, 3." The simultaneous notes of each chord played by the three trumpet parts appear together with a single stem within each time-slice. Each note indicates the part to which it belongs.

Example 2. Consider a score with divisi writing where the first violin part is temporarily divided into groups. The first violin part is split onto two separate staves in one system, and three separate staves in another system. There are a variety of ways this could be represented in NIFF. The choice depends on the desired result when the parts are to be extracted and printed for individual players:

a) If the first violin part score is to show all the divisi parts, the first violin should be defined as a single part with three voices and a maximum number of staves of three. The notes of this part would be distributed among the one, two or three active voices, each voice presented on its own staff.

b) If separate part scores are to be printed for different first violin players, more than one first violin part should be defined in the NIFF file. All the musical symbols that are to be printed on a particular part score would have that part number indicated. If the same passage is to be printed on two different part scores, both part numbers should be indicated for each symbol within the passage.

3 PHYSICAL STRUCTURES

A NIFF file is divided into two sections - the Setup Section, containing general information that applies to the whole score, and the Data Section, containing the music symbols and layout information. The Data Section is discussed in some detail here. The structures in the Setup Section are described later under Setup Section Structures. The topic of the relationships between individual music symbols in the Data Section is treated in Chapter 2, Symbol Relationships.

Hierarchy

The music symbols in the Data Section are stored in page-ordered format. The format has a hierarchical structure, where pages contain systems, systems contain staves, and staves contain time-slices and individual music symbols. Within each level, data are supplied generally in a left to right, top to bottom order, except as discussed in Chapter 2, Symbol Relationships. Text and graphics can be included at any level of the hierarchy, according to the type of object to which they logically belong.

Lists and Chunks

Pages, systems, and staves are all represented by lists composed of a header chunk followed by other component chunks that belong at the same hierarchical level. To save space, the time-slice has not been defined as a list chunk with a header and component chunks. Instead, the Time-Slice chunk functions as a sort of header to the group of symbol chunks which follow it. Every symbol within a Staff list, except for text or graphics immediately following the Staff Header chunk, is logically associated with the Time-Slice chunk that most closely precedes it.

A Staff list may contain music symbols belonging to more than one part. When more than one part is present on the staff, the part must be uniquely identified for every symbol. When only one part is present in a particular Staff list, its Part ID is identified in the Staff Header chunk instead of on the individual symbols.

Simulated Part Ordering, and Spacing By Part
A special case of the page-ordered data structure is known as "simulated part ordering." In this structure, there is only one system on one "very wide" page that extends from the start to the end of the score, with each part appearing on its own staff or staves. This special type of file contains a single Page list whose Page Header chunk has zero Width and Height tags.

The simulated part ordering structure is designed to accommodate non-page-oriented programs and programs which offer the user the option to strip page layout data when writing NIFF files. The music symbols of a one-staff part would all be stored contiguously in a single Staff list. Multiple-staff parts such as piano or organ would be split apart into two or more Staff lists.

Another special type of file structure is "spacing by part." In this scheme used by some notation programs, the symbols of each part are assigned horizontal placement values independently of the other parts, as though the file contained a set of part scores instead of one ensemble score. This could be used in a file with page ordering or simulated part ordering. A file recorded with this scheme is identified by the presence of the Spacing By Part tag on the NIFF Information chunk in the Setup Section. Reading programs unable to make use of this spacing should ignore all horizontal placement values, using its own spacing defaults instead.

Notes and Stems
A musical note is normally composed of at least two chunks - a Stem chunk and a Notehead chunk. It can be viewed as a degenerate case of a chord. A full chord is represented by a Stem chunk and several Notehead chunks, each of which represents a single notehead. There is no requirement that a Stem be followed by any Noteheads.

4 MEASUREMENT UNITS AND COORDINATE SYSTEM.

Goals
NIFF's measurement system is designed:

1) to allow enough flexibility to accommodate page-oriented notation programs, non-page-oriented notation programs and scanning programs,

2) to allow a fine enough resolution to describe high-quality printed music,

3) to use integers rather than floating point numbers, for the purpose of avoiding rounding errors and incompatibility among platforms, and

4) to describe each size and position measurement in a way that is semantically useful.

In addition, an attempt has been made to limit rounding errors during translation between the reading and writing programs' measurement units. For this reason, each writing program has its choice of units for absolute measurements.

Absolute Units and Staff Steps
NIFF uses two different measurement systems - absolute units and staff steps.

Absolute units are the writing program's own choice of units, declared in the Setup Section NIFF Information chunk. The unit choice is expressed in two field values: the standard unit (inches, centimeters, or points), and the number of absolute units per standard unit. For example, if the resolution used by the writing program were 4000 dots per inch, the writing program would choose a standard unit of "inches" and the number of absolute units per standard unit would be 4000.

The placement of a symbol whose meaning depends on its vertical position on a staff line or space is given as a staff step. The origin of the staff step system is the bottom line, which is given the value zero. Step numbers increase by one for each successive line or space in an upward direction from the origin. Negative numbers are used for the spaces and ledger lines below the origin.

Measurements and symbol placement are discussed in detail in the Symbol Relationships section below.

Page Coordinate System
The size of a page is given as its height and width in absolute units. The origin of the page's coordinate system is the top left corner of the page (as in the lower right quadrant of a Cartesian coordinate system). Measurements are increasingly positive to the right and towards the bottom of the page.

The origin of a staff is the left end of its top line. The origin of a system is the staff origin of its top staff.

Font size
The size of text fonts is given in twips. (One twip = 1/20 of a point, or 1/1440 of an inch.) This allows for scaling of fonts between point sizes, for those who can use this feature (e.g. a 210 twip font = a 10 1/2 point font).

The size of music fonts is described in two different ways: font size (in twips) and space height (in absolute units). Since there is no agreement on the meaning of font size between different music fonts, the space height is intended

to provide a clue about the intended size of the symbols on a standardized scale. It is equal to the vertical distance in absolute units between two adjacent staff lines of a staff on which the music font symbols would appear of normal size.

5 TIMING REPRESENTATION

Goals

Two goals in NIFF's timing representation are 1) to describe the time values in a semantically useful way, and 2) to provide enough information for the reading program to produce a MIDI playback of the score.

Two kinds of playback are: 1) logically precise playback derived from the notation, and 2) nuanced rubato of a recorded performance.

For each playback event, there are two pieces of information to be described: 1) start time, when the event is to occur, and 2) duration, how long it lasts.

Duration

Durations in music notation are inherently rational numbers (fractions) which describe the relationship between a specific unit of time (the note value) and a standard unit of time (traditionally the whole note). A half note is 1/2 of a whole note; a quarter note is 1/4 of a whole note. Infinitely smaller note types can be invented by increasing the denominator to any power of 2.

More interestingly, the duration of each note in a tuplet sometimes can be precisely represented only by a fraction, such as half note triplets, where each note's duration can be accurately represented as 1/3. When three half notes occur in the time normally allotted for 2, each note takes up 2/3 the time of a normal half note (1/3 = 2/3 * 1/2). Similarly, when 3 quarter notes are stretched out to the time of 4 (notated as 3:4 over a bracket, in tuplet notation), this duration can also be represented as 1/3: each note takes up 4/3 of the time of a normal quarter note (1/3 = 4/3 * 1/4).

A fraction consisting of two integers (a numerator and a denominator) is therefore a natural way to describe durations.

Tuplets

The duration field in a Notehead chunk is the same whether or not the note is included in a tuplet. Additional information (a series of tuplet chunks) is to be stored in the file for each tuplet. The first tuplet chunk contains a transformation ratio to be applied to each note in the tuplet, and graphical information about the appearance of the tuplet such as the presence of numbers and/or brackets.

The tuplet transformation ratio is in the form of 4 integers: a, b, c, and d, where a b-type notes occur in the time of c d-type notes. In conventional usage of tuplets, b and d will be the same, but many musicians use tuplets in an "unconventional" way, and may choose to represent a tuplet as, for example, three half notes in the time of four quarter notes (a, b, c, d = 3, 2, 4, 4).

For example, in the case of half note triplets the transformation ratio in the tuplet chunk is 3:2. When applied to the half note duration on the note chunk (1/2), this would result in an effective duration of 1/3 (1/3= 2/3 * 1/2). The effective duration need not be stored in the file, since it can be calculated by the reading program from values in the note and tuplet chunks.

A tuplet is normally a multi-node symbol, represented by a series of Tuplet node chunks, each one associated with one Stem chunk. All notes on a tupletized stem must contain the same duration. See Chapter 2 for information about multi-node symbols.

Nested tuplets are represented by an additional tuplet associated with a subset of notes already included in a tuplet. The transformation ratios are applied cumulatively to each affected note.

More than one voice or part can be included in a tuplet. For example, when all horns shown on the same staff are playing in rhythmic unison, a whole collection of chords may be tupletized. However, within each voice (or part, if there is one voice per part), the sum of the durations of the notes must equal a b-type notes.

Logical Start Time

An explicit start time is required because simultaneous events are not always vertically aligned - for example, a chord with seconds. Ambiguity can result for a variety of other reasons as well.

The start time of a note or rest is stored on the event Time-slice chunk associated with (most recently preceding) the note or rest symbol. The start time is represented by a fraction which is the sum of the durations of all events that have occurred within the voice since the previous measure-start time-slice.

Here is an example, showing the start times and durations for 4 quarter notes in the first measure of a score in 4/4 time.

```
Time-slice, type=event, start-time=0/4
Stem
Notehead, duration=1/4
Time-slice, type=event, start-time=1/4
Stem
Notehead, duration=1/4
Time-slice, type=event, start-time=2/4
Stem
Notehead, duration=1/4
Time-slice, type=event, start-time=3/4
Stem
Notehead, duration=1/4
```

The start time of a measure is stored on a measure-start Time-slice chunk. It is given as the cumulative duration of all events since the start of the piece. The effective start time of each event in a measure is therefore the sum of the

measure start time (from the measure-start Time-slice chunk) and the event start time (from the event Time-slice chunk). In the example above, the start time on the measure-start Time-slices of the first and second measures, respectively, would be 0/4 and 4/4.

The reference back to the start of the piece permits the correspondence between events in different parts to be determined even when the parts are in different meters.

Note concerning pickup bars: NIFF has no standard "correct" way to represent such (incomplete preliminary) bars. In particular, scanning programs may not have the intelligence to distinguish a genuinely incomplete bar from a pickup measure. Therefore the writing program has the choice of representing the pickup either as simply a "normal" incomplete bar, or else of representing it "correctly" (in musical terms) as the final portion of a preceding bar. As long as the barline time slice and the following measure time slice contain the same time, it should make little difference on which beats the two pickups are placed. Implementations should be prepared to deal with either of these possible representations of time in a pickup measure.

Logical Gaps

An additional reason start time is required is that voices are not always logically complete from start to finish. If a voice were logically complete, the start time of each event within the voice would always be the exact time that the previous event in that voice finished. Logical incompleteness (gaps) can occur, however, for various reasons: 1) Piano music often allows temporary voices to appear and disappear with no rests to fill the time gaps. 2) A non-metrical cadenza-like section can appear in one voice or part, leaving all other voices or parts with a logical gap during that time. 3) Scanning programs can easily miss one or more notes in the middle of a sequence. 4) Inactive parts might be hidden for large sections of a score.

Logical gaps can be filled in with invisible rests and placeholding measure-start time-slices. Or, logical gaps can be handled by adding extra value to the start time of the event following the gap to compensate for the missing time. Either technique ensures that the event following a logical gap in one voice will not occur until its proper time among the events in all other voices and parts.

Performance Information

For performance playback, both start time and duration are measured in MIDI ticks, which denote some fixed amount of time using an integer. MIDI ticks are well suited to performance information because of their high resolution. The relationship between the performance and logical values is described by the field MIDI ticks per quarter note, stored in the NIFF Information chunk in the Setup Section.

The performance start time and duration are given as offsets (+/-) from the logical start time and duration of the event.

Lengthy or complex MIDI information that cannot be represented with the simple MIDI Performance tag can be supplied with the MIDI Data Stream chunk (see below under MIDI Integration).

Additional Comments

The Notehead and Stem chunks and associated chunks and tags contain graphical features such as notehead shape, number of dots and number of flags that in printed music describe a note's duration. However, the reading program need not concern itself with the timing implications of these details. The duration field stored on the Notehead chunk should be used for the timing value, regardless of the graphical features. For example, a note with duration of 3/16 could appear visually as a dotted eighth note with either a flag or a beam. In practice, any combination of a note's graphical features, or even an invisible note on a zero length stem, could be used to represent any time value.

The performance playback values may be affected by other chunks and tags associated with the Notehead or Stem chunks, however. For example, an Accidental chunk will affect the pitch, an Articulation chunk or Silent tag might affect the performance duration, and a Grace Note tag would likely affect the start time.

An event's start time and duration are always present (each composed of a numerator and denominator component); the performance start time and duration are optional values that can be supplied on the MIDI Performance tag. If not present, it is assumed that the performance values are equivalent to the logical values.

NIFF does not require logical adherence to time signatures.

Barlines, Clefs, Time Signatures, Key Signatures

A barline signals the end of a measure rather than the start of a measure. On the last measure of a system it is necessary to store the Barline chunk before the measure-start time-slice, which belongs in a new Staff list and a new System list. So, for consistency and logical rigor, the Barline graphical symbol chunk is to be stored between the last event in the measure and the measure-start time-slice of the next measure. A special event time-slice chunk should be stored for this purpose, to uniquely identify the sequential position of the barline. Its start time can be the same, in essence, as the start time of the next measure-start time-slice, except given in relation to the start of its measure rather than the start of the score. The only time non-unique time-slices are allowed within a Staff list are when an event time-slice duplicates a measure-start time-slice in this way.

Chunks representing changes in clef, key signature and time-signature, when present at a measure boundary, should be stored in the same way, following the Barline chunk. Horizontal placement can be supplied to identify the individual placement of each symbol within this special time-slice, though most reading programs would have spacing algorithms to handle them adequately without it. When changes in clef, key signature or time signature occur in the middle of a measure, they should be stored with the time-slice which represents the following event. The horizontal placement of a symbol chunk in this situation, if specified, should have a negative value.

Chunks representing the clefs, key signatures and time signatures that appear at the start of the staff do not need to be preceded by an event-time-slice. They are stored immediately after the new measure-start time-slice. In the case where the new system starts in the middle of a measure, these chunks should be stored with the time-slice which represents the following event. The horizontal placement of a symbol chunk in this situation, if specified, should have a negative value.

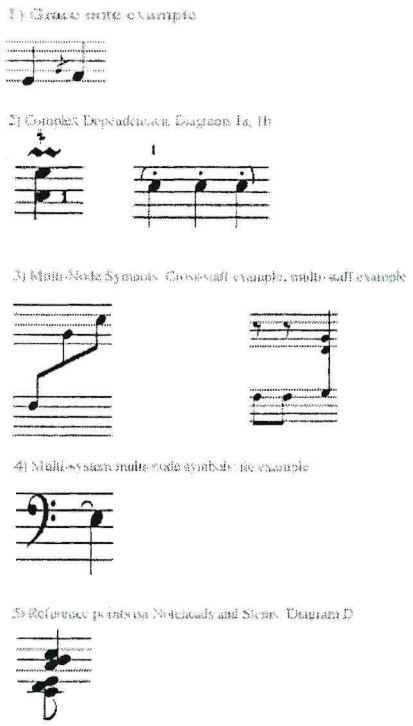
Grace Notes

A grace note is a note whose time value is not counted in the general rhythm; its time value must be subtracted from that of the preceding or following normal note during performance. A grace note is always associated with the event time-slice of the normal note following it. It is indicated by the addition of a Grace Note tag to a Notehead or Stem chunk. The Grace Note tag gives the note's logical start time as an offset from thestart time of its time-slice (negative, zero or positive, depending on the chosen interpretation of grace notes and the note's position within a series of grace notes).

For small notes that break the meter but whose time is not subtracted from that of the preceding or following normal note, do not use the Grace Note tag.

The performance start time (in the note's MIDI Performance tag) is given as an offset in MIDI ticks from the grace note's logical start time. Other tags that might be found on a grace note's Notehead or Stem chunk include the Slash tag, Small Size tag, and the Absolute Placement tag (with a negative value) indicating a negative horizontal offset from the time-slice.

In the following example, the grace note's time value is subtracted from the preceding note during performance, as in Chopin or Liszt. However, the duration field on the preceding Notehead chunk is not modified to compensate for the grace note. The start time on the Grace Note tag gives a negative offset of 1/32 from its time slice, which can be used by the reading program during playback.



```
Time-slice, type=event,start-time=0/4
Stem
Notehead, shape=filled, staff step=0, duration=1/4
Time-slice, type=event, start-time=1/4
Stem, Number of Flags=1, Slashed Stem, Small Size, Grace Note=-
1/32
Notehead, shape=filled,staff step=2, duration=1/32
Stem
Notehead, shape=filled, staff step=1, duration=1/4
```


ACKNOWLEDGEMENTS

This article is extracted from the NIFF Official Web Site [12]. We would to thank Prof Alan Belkin, NIFF Coordinator, for the permission to use these material.

BIBLIOGRAPHY

1. Read, Gardner, "Music Notation - A Manual of Modern Practice," Second Edition, New York: Taplinger Publishing Company, 1969 (1979).

2. Ross, Ted, "The Art of Music Engraving and Processing," Miami Beach: Hansen Books, 1970.

3. Byrd, Donald, "Music Notation by Computer" , PhD dissertation , Indiana University, 1984. Available from UMI, 300 N. Zeeb Road, Ann Arbor, MI 48106. Telephone 1-800-521-3042. Order number DA8506091.

4. Microsoft Press, "Microsoft Windows Multimedia Programmer's Reference," Redmond, WA, 1991. Telephone: 1-800-MSPRESS, ISBN#1-55615-389-9. Contains the full description of Microsoft's RIFF standard.

5. Ornstein, Severo M. and Maxwell, John Turner Maxwell III, "Mockingbird: A Composer's Amanuensis," Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA, 94304. CSL-83-2. January, 1983. [P83-00002].

6. Apel, Willi and Daniel, Ralph, "The Harvard Brief Dictionary of Music", New York: Amsco Music Publishing Company, 1960.

7. Heussenstamm, George, "The Norton Manual of Music Notation," New York: The Norton Company.

8. Roemer, Clinton, "The Art of Music Copying", Sherman Oaks, CA: Roerick Music Co, 1985. (out of print)

9. Vinci, Albert C., "Fundamentals of Traditional Music Notation," Kent State University Press, 1989.

10. G. Schirmer, Inc. "The G. Schirmer Manual of Style and Usage," New York: The G. Schirmer Publications Department, 1990.

11. Yergeau, Francois, "UTF-8, a transformation format of ISO 10646," RFC 2279, Internet Engineering Task Force, January 1998.

12.http://www.musique.umontreal.ca/personnel/Belkin/N/NIFF6b.html

Music Information Description by Mark-up Languages within DB-Web Applications

Goffredo Haus Maurizio Longari
LIM-DSI
Università degli Studi di Milano
via Comelico, 39
20135 Milano, Italia
+39 02 5835 6222, +39 02 5835 6296
haus@dsi.unimi.it, longari@dsi.unimi.it

Abstract

We are working on several projects addressing digitalization and restoration of legacy musical archives; We are dealing with several sources of musical information (audio, notation, performance, video, images, etc...). On one hand, these sources are similar to those of usual multimedia databases. On the other hand, some specific feature of legacy musical archives call for ad-hoc solutions.

In legacy musical databases many relations are implied by the framework (for instance, recordings of parts of the same piece), but not made explicit in the content. To address this specific feature, we propose a Three-level database schema where the bottom level (called Source Level) describes the source material; the mid level (called Logical Level) describes explicit and implied relationships among bottom-level data. Finally, the top level (called Catalog Level) represents the common catalog data attached to pieces (composer, conductor, data of recording...).

To support this methodology, we need a language to describe musical content. In this article we survey several mark-up languages (such as SMDL and XML derived languages) and discuss their features w. r. t. our proposed layered database schema.

Keywords

Symbolic music representation, multimedia database, XML, SMDL

1. Introduction

Symbolic musical representation has long been a topic of study without an accepted standard having accepted yet.

In our experience with important Musical Archives (Teatro alla Scala and Bolshoi Theatre [11]) the symbolic representation is of crucial importance. In fact we have to deal with different sources of material representing a unique piece of music and to their relations thereof.

Other projects having similar data source like VARIATIONS project [7] at Indiana University and Online Music Recognition and Searching (OMRAS) project [22] addresses the need to represent symbolic information.

Furthermore, since there is a demand to support web protocols, it is necessary to represent the symbolic information in a format that allows the simple integration with web technologies.

We present a three-level database schema that addresses the topic of representation. Then, several symbolic musical representation proposals are presented and evaluated as candidates for a representation standard.

2. Three-level database schema

We propose the following structure:

Catalogue Level: information in tabular form
Logical Level: information in symbolic notation
Source Level: contains the source files of music pieces (audio, notation, performance, video, images, etc...)

The Source Level is the grounding level of the information stored in the database. Since the dimension of each item is significantly big, this layer needs particular handling techniques. Each file may contain either a complete or only a part of the piece. No explicit relation among single files is stored at this level although several source files can take part to the representation of a single piece.

The correlation information must be drawn out from the content of each source file and represented in a more

comprehensive and easy-to-handle way. Therefore, we abstract the information contained in the different representations of a musical piece. This must be done at different levels: form the structure of a complete piece (e.g. an Opera) to the representation of a single event (as an audio or graphical phenomena) and simultaneously from emotional to analytical information. Moreover, it is necessary to represent the relationships that can exist among the various abstract and non-abstract elements [6].

This information is stored in the Logical Level. The contents and format that this representation must capture is the topic of this article.

In the Catalog Level the information attached to pieces of music is represented in tabular form. Catalog data can be stored in this layer together besides content indexing information extracted from the Logical Level by means of harmonic and melodic algorithmic segmentation.

In the next sections we will analyse some musical representation formats in order to verify whether they can capture and describe the properties needed for the representation in the Logical Level. Furthermore, we mind that a useful symbolic representation must be a suitable format for exchanging and delivery over the Web.

3. SMDL representation

3.1. The draft

The Standard Music Description Language (SMDL [21]) was created in 1984 from an idea of Charles Goldfarb. The purpose was to standardize musical information by means of the capabilities offered by SGML format. The idea was to create an environment in which the several aspects of music representation can be considered and used.

SMDL is an instance of HyTime (Hypermedia Time/Based Description Language [9][19]) and it inherits the functionality of this standard [20][25]. It divides the aspects of music representation in four domains: Logical (cantus), Visual, Gestual and Analytical.

Logical domain represents the intent of the composer when he writes the musical piece. Visual domain incorporates the graphic information and score notations. The Gestual domain contains performance and executive information. Analytical domain embraces all music background information.

Example 1. SMDL representation of a note and a rest.

```
<ces id=ces4>
<pitched exspec=exlist1>
<nompitch><gampitch><pitchnm>c</pitched>
<rest exspec=exlist1></rest>
</ces>
```

Logical domain (called *cantus*) codes musical symbols in SGML syntax. Others domains use the hyperlink capabilities of HyTime to relate to other representation of specific musical information [19].

SGML syntax of SMDL is defined in the standard draft ISO 10743 [21]. The structure of information is developed over a system of time dependent axis called FCS (Finite Coordinate Space, an architectural form of HyTime) that is composed of cantus events (notes, rests, symbols). The other domains can refer to this representation to link the content of music information.

3.2. Developed tools

We have developed a software that extracts the information of pitches, duration and parts from the MIDI and NIFF files [10]. We started from a NIFF to SMDL translator made by Steve Mounce of Bradford University [13]. This software is composed of two stages that execute the steps of translation. The first stage extracts from a NIFF file the essential information for the translation and writes them into another file into an intermediate SGML format using the tools provided by the NIFF SDK toolkit [3]. The second stage translates the intermediate file into SMDL syntax by means of the HyMinder System (the only software-tool available to handle the HyTime Standard [26]).

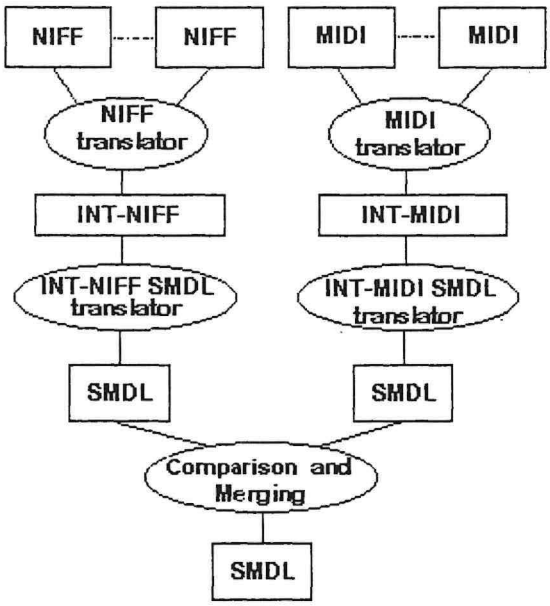


Figure1. Translation from NIFF & MIDI to SMDL

Starting from this two stages we have developed a second branch of translation that takes MIDI files (in both format0 and format1), translates them into another intermediate SGML format (INT-MIDI) and by means of HyMinder system translates it into SMDL syntax. A third

stage merges the two SMDL files obtaining a single SMDL file with hyperlinks to the source files.

4. XML - Extensible Markup Language

"The Extensible Markup Language (XML [2]) is a subset of SGML ... Its goal is to enable generic SGML to be served, received and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML."

XML has been developed by the W3C (World Wide Web Consortium) in 1996 and its design goals was:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

Defining an XML document has the same difficulty as to write the document itself. The document structure can be created while it is compiled, paying attention to some syntactical rules. For example a poem can be encoded as in example 2.

We must note that the logical structure of an XML document is a hierarchical structure with a system of hyperlinks by means of which is possible to define cross-element relations.

Example 2. XML representation of a poem.

```
<poem style="Romantic">
  <author>William Blake</author>
  <title>Never Seek To Tell Thy Love</title>

  <paragraph>Never seek to tell thy love
  Love that never told can be;
  For the gentle wind does move
  Silently, invisibly. </paragraph>
  ...
  <paragraph>...
  O, was no deny.</paragraph>
</poem>
```

An XML document is said to be *Well-Formed* if it respects the hierarchy structure, if all the tags are open and closed correctly, if the nesting of the elements are well-posed and some other rules are respected.

4.1. Document Type Definition (DTD)

XML is a meta-language derived from SGML, it defines the rules to describe the structure of a class of documents. These rules are written in the document standard definition [2]. An instance of this meta-language is called Document Type Definition (DTD) that defines the structure of the instance documents.

A DTD is mainly composed of definition of Entities, Elements and Element's Attributes. Entities can be SYSTEM Entities (reference to objects resident externally of a DTD) or Parameter Entities that define reusable objects inside the DTD. Elements are the components of the structure of the hierarchy. For each element a series of attributes declared as Element's Attributes can be defined. In example 3 we show the DTD that describe the poem of example 2.

If the XML file is well-formed and it respects the structure defined in the DTD it is said to be a *Valid* Document.

Example 3. Poem's DTD.

```
<Entity % style.type "Romantic | Classic |
  Neo-classic">
<!ELEMENT poem (author | title | paragraph+)>
<!ATTLIST poem style (%style.type;) #REQUIRED>
<!ELEMENT author (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT paragraph (#PCDATA)>
```

4.2. XML Schema definition

The schema definition of an XML document is another way to define the structure. It is now a W3C recommendation and it is proposed for approval [8].

It has many new features that gain the power of XML to represent structures, such as:

Data Types: it is possible to define simple, complex and list data types. The Schema makes available some simple and list data types that can be used to build complex types;

Namespaces: (definition form the draft) "A schema can be viewed as a collection (vocabulary) of type definitions and element declarations whose names belong to a particular namespace called a target namespace. The target namespace enables us to distinguish between definitions and declarations from different vocabularies".

It is also possible to define element's properties such as uniqueness, key elements and abstract data types.

5. Available technology for XML

Since XML has been approved by the W3C a broad range of software handling the standard has been developed. Software tools allowing the verification of well formed-ness and validity vs. a DTD are called parsers. There are several implementations of these kind of tools that make available the source code in Java, C++ and other programming languages.

Editors are a more complex kind of software allowing validation, modification and structure visualization.

The most important application of XML is in the area of databases. In this environment the standard is used as a real exchanging format: because of its hierarchy structure we can design a correspondence between a database schema and a DTD or Schema definition. In this way it is possible to represent the queries and especially their results as XML document. The resulting document contains not only the data but also the structure. Moreover any XML parser or validator can read it. Software implementing XML Schema is in early stage of development.

Good performances are relevant properties in databases. The weakness of this framework is that for every query it is necessary to make a translation between the database schema and XML format. XML Native databases implements the XML format as a model of data storage providing direct input and output of XML documents.

5.1. Extensible Style Language (XSL)

XSL [1] is an XML application that allows translation and formatting of XML documents. Actually two languages compose XSL: XSL Transformations (XSLT) and XSL Formatting. The XSLT language translates an XML document into another XML document. XSLT is also designed to be used independently of XSL. XSL Formatting language is designed to render XML document producing presentation of the content data. An example of use of XSL is the rendering of XML documents produced by a query in a database to an HTML file displayable directly on a browser.

6. XML for music representation

In the past few years several proposal have been made to represent the symbolic musical information by means of XML standard. We will discuss the most complete and relevant.

6.1. MusicXML

The definition of the proposal [16] is under development by Michael Good of Recordare, and it is in an early review state.

6.1.1. The structure. Conceptually the structure of the document is composed of individual score files (*Score* Element) collected together within the structure defined in the *Opus* element. The Opus element allows direct and indirect arbitrary nesting of other Opus and Score elements.

This content information is based on the MuseData and Humdrum [24] formats that explicitly represent the symbolic musical information in two dimensions (time/measure and part/instrument). Since XML is a hierarchical representation, the structure of each Score element can be represented in one of the two conceptual hierarchies: Timewise and Partwise. The type of hierarchy is declared in each Score:

Partwise: within each part the sequence of the measures composing the part is coded;

Timewise: for each measure of the score the relative measures for each part are coded;

Within each measure or within each part the symbols of the musical events of the score are coded. An XSLT program has been developed that allows an automatic translation between the two orderings. The definition of the DTD is split-up in several files, each one containing a modular part of the definition:

Common definitions: elements and entities that are shared among multiple components like date formats, spacing units, position coordinates format, embellishment structures and others;

Score identification: elements describing meta-data about the score;

Attributes: element definition of key and time signature, clefs, transpositions and other part's attribute elements;

Note: elements defining a consistent amount of property of the note, from the graphical representation to the logical aggregation between different parts, from articulations to performance information (technical information to play the note on particular instruments, in this draft are coded the techniques for strings and keyboards instruments) and associated lyric information;

Barline: element defining the several types of barlines and their functionality like repeats, multiple ending etc...

Directions and general purpose elements: contains elements describing several sign not directly related to the notes (e.g. segno, pedal, others), harmony structures (e.g. describing chords), printing and sound elements to specify general parameters associated to the rendering of the piece.

6.1.2. Considerations. The MusicXML proposal is intended as a sufficient interchange format for notation, analysis, retrieval and performance application. The content represented by the format is score-oriented, e.g. the notes are represented as symbolic and graphical objects.

Example 4. MusicXML representation of a note.

```
<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>
```

6.2. MusiXML

The aim of the proposal written by Gerd Castan [4] is to define a standard format in exchanging musical score information.

6.2.2. The structure. The structure is subdivided into three sections: Bibliographic data section, Logical data section and Filter data section. The Bibliographic section is very small and allows only the explicit definition of composer's name descriptions; other information are described as a normal sequence of characters.

The other two sections are based on the idea of writing the minimum amount of redundancy information: as the composer says "...we consider separating content from style." The given example is:

> "Assume we represent musical sheets for a symphony orchestra. To see the problem, we need only the score and the violin 1 part. Using a hierarchical representation, the score looks like this:
> `work -> page -> system -> staff -> measure -> <content>`
> and the violin 1 sheets look like this:
> `work -> page -> system -> staff -> measure -> <content>`
> Both have the same structure. They have different instances of page, system and staff, but they share parts of `<content>`..."

In the Logical section the single instrument parts are coded. In the Filter section the structural information needed by different "views" of the content is coded.

In the Logical section each part is represented as a work-segment that is subdivided into one Master Thread and one or more Threads. In Master Thread the time dependant elements that occur only once per score are coded. In the Thread section are represented sequential music symbols such as notes, bars, chords, beams, slurs, clefs and keys. The note's representation follows the guidelines of the NIFF representation, i.e. each note is a generalization of the chord symbol with only one note-head.

The Filter section is subdivided into extracts and staffs. One extract is a grouping of staffs with bibliographic data, bracket, brace and references to the staffs. A Staff contains references to Threads and some context attributes like start time, clefs, keys, metric and others.

Example 5. MusiXML representation of a note.

```
<chord>
<note name="d" oct="2" dur=":8" beam="b1.2"/>
</chord>
```

6.2.2. Considerations. The structure is incomplete as the composer remarks in the DTD:

> To do:
> * find a better word for time
> * replace start_klammer and end_klammer by an english word
> * define register pipeLen
> * define duration
> * think about how to handle %lasting; attributes
> * replace paragraph + type by scoreParagraph
> * octave shift for clefs

An XML DTD and an XML Schema definition of MusicXML are available. We note that the MusicXML Schema is richer than the MusicXML DTD because XML Schema language provides tools that XML DTD language does not.

6.3. MusiCat

This proposal, defined by Perry Roland [14], is a catalogue oriented encoding format.

6.3.1. The structure. A MusiCat document is organized as a catalogue document and has four main sections: header, body, front and back.

In the header the bibliographic sections and administrative information are coded. The front & back sections contain the usual elements of a document like titles, index and bibliography.

The body element contains the great part of the encoded information:

High-level bibliographic data elements contain elements about the material like physical descriptions, distribution, control access, provenience, physical location (e.g. in shelves) and musical details;

High-level bibliographic analysis elements contain elements about the history, material summary, biographical information, circumstances and methods of

composition, description text notes, information about performances, publications, transcriptions and relation to other works or materials;
Data grouping elements contain elements like addresses, information about analogical and digital descriptions, incipit and grouping of text information.
Data elements are those elements that describe the specific characteristics of the material like parts of addresses, single bibliographic descriptions, conversion methods, channels of a recording, encoding system, file formats, identifiers numbers (such as ISBN, ISSN), some musical information like key, clef and time meter, notation and other.
Formatting elements contains element to describe the structure of text and descriptive elements.

Some description elements require a table structure to be organized, the DTD provide two methods to contain the data, the html tables and CALS tables. Html tables are the default. Finally the *linking* and *inline* elements, used within other elements, are described.

6.3.2. Considerations. As we can notice the topic of the description, as commented in the document, is:

> "To encode new and existing music catalogues (finding aids, discographies, databases, etc.), leading to more widespread electronic storage and transmission of such musical information."

In fact the schema doesn't provide elements to describe musical content, it lets to other notation format the task to represent it. Only an element (<notation>) containing text that describes the musical content is provided (in [23] the Humdrum format is referenced as an example).

6.4. Other proposals

Some other musical representation proposals have been written. As a matter of completion we quote them:

6.4.1. Musical Notation Markup Language (MNML). Developed at IRDU (Internet Research and Development Unit) of National University of Singapore, this proposal is an "attempt to represent the entire musical and lyrical content of a music score". As declared in the description of the specifications: "MNML fully describes the basic melody and lyrics of the piece. It is not the intention at this stage to include capability for defining complex layout and other publishing features, performance annotations, and other embellishments, as it has not been accepted as a standard." [18].

A prototype of MNML editor and search engine has been developed.

6.4.2. Music Markup Language (MML). Developed at University of Pretoria under the direction of Jacques Steyn. The development of the proposal is carried on by student's projects. The aim is to define a markup language that can represent "the structure of music and music related processes". They also have developed some experimental software to graphically render an MML file [12].

6.4.3. MuTaTeD. MuTaTeD stands for (Music Tagging Type Definition). It is a project coordinated by Carola Boehm that "intends to validate the concept of integrating the two existing music representation standards, SMDL and NIFF, and to provide a meta-standard for music mark-up (SMDL DTD)."[17].

6.4.4. MusicML. Developed by Jeroen van Rotterdam, it is a simple draft that can represent only basic musical information. He has developed also a Java applet to parse and visualize MusicML files. [15].

6.4.5. ChordML. A very simple DTD developed by Gustavo Federico that can represent only lyrics and associated chords [5].

7. From SMDL to Music-XML

All SMDL and Music-XML languages are mark-up languages derived from SGML. The main difference between the two kinds of languages resides in the fact that SMDL derives also from HyTime (Hyper-Media Time-Based Description Language) that is a more powerful language then XML. For example SMDL grounds the representation of musical events on the HyTime FCS architectural form. In this model musical events are represented in a time dependent co-ordinate space and the language has the ability to describe projection and modification of events or group of events. Moreover, SMDL inherits from HyTime the powerful linking and addressing systems that, for example, can operate on multiple locations and can define path and graphs of links and references. On the other hand SMDL is still in a draft stage and there are only experimental implementation of the standard.

As we mentioned in section 4, XML is born as a simplified subset of SMGL in order to become a standard of information exchange over the Internet.

XML simplifies the implementation of SGML (loosing some properties) and inherits from HTML the hyperlink capability. This property gives to XML enough power to be used on the Web. Just this property and the sudden growth of implementations have given the language a position of great interest for commercial applications. Implementations of XML to music representation can take advantage of this property for the purpose of web

delivery of music.

Table 1. Comparison of mark-up languages.

	Symbolic notation	Catalogue info.	Extra info.	Software tools
SMDL	high	mid	high	poor
MusiXML	mid	mid	*	high
MusicXML	high	mid	*	high
MusiCat	poor	high	mid	high
MNML	mid	poor	*	high
MML	mid	poor	*	high
MusicML	mid	*	*	high
ChordML	poor	*	*	high

Regarding the content representation, both the languages can represent the same set of musical objects, because mark-up syntax is the same provided by SGML. In fact there is one attempt to rewrite SMDL by means of UML (Unified Markup Language) and to publish it in an XML-compatible form. The evaluation of the SGML-languages depends on the linking and addressing ability. A meta-representation of different pieces of information must be able to link different physical location and formats. The linking system of SMDL is very powerful: for example the multi-location linking system can capture the musical parallelism information. With XML languages it is however possible to represent links to external files also in different formats satisfying the requirements described in the Logical Level of the database.

Each of the different Music-XML representations described earlier capture different aspects of Music representation focusing on different details. The presented proposals are somehow complementary, i.e. information that one language do not represent can be represented by another language. For example MusiCat do not represent the note information that MusicXML and MusiXML describe.

Furthermore XML is becoming a standard to exchange and represent information over the web. A lot of implementations are developed for its use. These two facts pose Music-XML languages at the first place as candidate to our aims.

In table 1 all reviewed languages are qualitatively compared. Parameters of comparison are evaluated individually into four values: high, mid (middle), poor and * (not envisaged).

Symbolic notation parameter indicates the ability to represent score information. Catalogue information parameter points out the quality and quantity of archival attributes. Extra information parameter measures how much external data can be related to the content of mark-upped data. Software tools parameter refers to the possibility of tools implementation. We have intentionally left out MuTaTeD project since no definition document were released.

8. Conclusions and future development

As discussed before the expressiveness in terms of symbolic notation of both the languages is enough. On one hand Music-XML proposals cannot be used in our framework since at this time they are not complete. On the other hand SMDL is a rich language but is still in a draft version not approved by the standardization organizations and it is expected to be reviewed and updated.

The evaluation must also consider the powerfulness of a language in relation to the needs involved in the representation. Supposing that we choose SMDL as the language representing the symbolic information within the Logical Level of the database, is it too expensive to deal with SMDL with regard to the task that we must compute with the musical data?

Music-XML languages can boast about the broad diffusion and implementation of the language that allow an easier development of tools suitable to exchange over the web.

We can conclude that at this stage mark-up languages representing musical information are not sufficiently complete to allow us to choose which format has to be used. In our opinion a presence of an accepted standard that takes into consideration the problems discussed above is needed not only for the representation in our three-level database schema but also for musical information exchanging.

9. Acknowledgements

The authors wish to acknowledge the partial support of this project by the Italian National Research Council, in the frame of the research program "Methodologies, techniques, and computer tools for the preservation, the structural organization, and the intelligent query of musical audio archives stored on heterogeneous magnetic media", Finalized Project "Cultural Heritage", Subproject 3, Topic 3.2, Subtopic 3.2.2, Target 3.2.1.

This work has been made possible by the effort of a number of researchers and graduate students at LIM. A special thank is given to Fiorenzo Galli, General Secretary of Foundation Milano per La Scala, for his fundamental role in designing and supporting the Scala and Bolshoi projects. Thanks are expressed to the projects' sponsors: AEM, Accenture, Banca Commerciale Italiana, Hewlett-Packard, Oracle, ST Microelectronics, TDK, and the Teatro alla Scala Foundation USA. The authors wish to acknowledge A. Provetti, M. Bertoldi for improving the English language quality of the paper.

10. References

1. AA. VV. Extensible Stylesheet Language (XSL) Version 1.0 *W3C Candidate Recommendation* 21 November 2000 http://www.w3.org/TR/xsl/
2. Bray, T. Paoli, J. Sperberg-McQueen, C.M. Maler, E. Exstensible Markup Language (XML) 1.0 (second Edition) http://www.w3.org/TR/REC-xml
3. Butler, T. Grande, C. Faq for the NIFFSDK. site http://mistral.ERE.UMontreal.CA:80/~belkina/N/niffsdk.txt
4. Castan, G. Common music notation and Computers: Index. http://www.s-line.de/homepages/gerd_castan/compmus/index_e.html
5. ChordML Web site http://www.oasis-open.org/cover/chordML-sample.txt
6. Dannenberg, R.B. Music Representation Issues, Techniques, and Systems. *Computer Music Journal*, 17(3), pp. 20-30
7. Dunn, J.W., Mayer, C.A. "VARIATIONS: A Digital Music Library System at Indiana University" *Proceeedings of the Fourth ACM Conference on Digital Libraries*, Berkeley, CA, Aug 1999, pp. 12-19
8. Fallside, D. C. XML Schema Part 0: Primer. http://www.w3.org/TR/xmlschema-0/
9. Goldfarb, C.F. Newcomb, S.R. Kimber, W.E. Newcomb, P.J. Information Processing – Hypermedia/Time-based Structuring Language (HyTime) - 2nd edition. *ISO/IEC JTC 1/SC 18 WG8 N1920rev*, May 1997.
10. Haus, G. Longari, M. Coding Music Information within a Multimedia Database by an Integrated Description Environment. *Atti del XII° Colloquio di Informatica Musicale*, AIMI/Università di Udine, Gorizia, 1998
11. Haus, G. Rescuing La Scala's Audio Archives. *IEEE Computer, 31(3), pp.88-89, IEEE CS Press*, Washington, 1998.
12. MML Web site http://www.mmlxml.org
13. Mounce, S.R. NIFF to SMDL translator Documentation. *Software Documentation for WP3*, Project Cantate
14. MusiCat Web site http://icarus.lib.virginia.edu/~pdr4h/musicat/
15. MusicML DTD site link http://www.oasis-open.org/cover/musicML-DTD.txt
16. MusicXML DTD web site http://www.recordare.com/xml.html
17. MuTaTeD http://www.pads.ahds.ac.uk/mutated
18. Narayan, K. Chiam, P. Wei, Y. Kok, L. Tin Wee Y. and T. Delivering Music Notation over the Internet: Syntax, Parser, Viewer and GUI-Input System. *Proc. of Inet'96,* http://irdu.nus.edu.sg/music/
19. Newcomb, S.R. Kipp, N.A. Newcomb, V.T. The HyTime. Hypermedia/Time-based Document Structur-ing Language. *Communication of the ACM 34:11*, November 1991.
20. Newcomb, S.R. Standard Music Description Language complies with hypermedia standard. *COMPUTER*, vol. X N.Y. 24:7, pp. 76-79, July 1991
21. Newcomb, S.R. Standard Music Description Language. ISO/IEC DIS 10743 ftp://ftp.ornl.gov/pub/sgml/WG8/SMDL/10743.ps, June 1995.
22. On Line Music Recognition And Searching (OMRAS) http://www.omras.org
23. Roland, P. Extensible XML4MIR: Markup Language for Music Information Retrieval. *Proc. of Music IR 2000*
24. Selfridge-Field, E. Beyond MIDI The Handook of Musical Codes. *The MIT Press*, Cambridge, Massachusetts, 1997
25. Sloan, D. Aspect of Music Representation in HyTime/SMDL. *Computer Music Journal*, Cambrige, MIT Press 17:4, pp 51-59,Winter 1993
26. Techno Teacher Inc. HyMinder User Guide 0.8.4, http://www.techno.com 1996

Coding Music Information within a Multimedia Database by an integrated description environment*

Goffredo Haus **Maurizio Longari**
LIM Laboratorio di Informatica Musicale
Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano
via Comelico, 39 I-20135 Milano (Italia)
fax +39 2 55006373 e-mail: maurizio@lim.dsi.unimi.it

Abstract

The symbolic representation of musical information involves several aspects of computer science. A piece of music can be represented in several ways such as: digital signals, sets of time-dependent events that a sequencer (or other devices) can translate into an audio performance (e.g. MIDI), sets of graphic images that represent the score (e.g. TIFF, GIF, JPEG), a notational representation of the score that respects its hierarchical structure (e.g. NIFF, DARMS) or a representation of the logical information (for logical information we mean the information common to all the above music aspects) that the composer intends to put in the piece (SMDL). The fact that SMDL is an HyTime application, makes it a powerful environment in which all the other formats can be integrated and organized by means of the hypermedia ability. Since SMDL's cantus domain can represent the logical information of a musical piece, we can extract the logical information from files coded in the formats mentioned so far, translate them into SMDL, and then use the SMDL representation within a database.

The representation of the musical information of a whole piece of music within the database is quite expensive with respect to memory occupation and computing time. Therefore it is necessary to devise methods for reducing the cost. A possible solution to this problem is to automatically extract the significant parts of a piece of music from the SMDL files, and then code these parts into the database instead of SMDL syntax. By means of this approach, most of the queries on pieces of music can be solved directly into the database without accessing the files.

Therefore in this environment SMDL plays two roles: the role of the common descriptor for the other file formats with the ability to organize them as hypermedia objects, and the role of the front-end to the database system.

SMDL's origins

The standard music description language (SMDL [1]) was born in 1984 based on an idea of Charls Goldfarb for standardizing the computer representation of music in an SGML format. The idea was to create an environment in which the several aspects of music representation can be considered and used.

SMDL is an instance of HyTime (Hypermedia Time/Based Description Language [2][3]) and so it inherits the functionality of this standard [6][7]. It divides the aspects of music representation in four domains:

- Logical (cantus)
- Visual
- Gestual
- Analytical

The Logical domain can represent the intent of the author when it thinks the musical piece. The Visual domain incorporates the graphic information and score notations. The Gestual domain incorporates the performance and operational information. The Analytical domain incorporates all of the music background information.

In the Logical domain you can write in an SGML format the music information while the other domains use the hyperlink capabilities of HyTime to point to and integrate the other formats that can represent the particular information [5].

The Logical domain is called *cantus* and the SGML syntax of SMDL is defined in the standard draft ISO 10743 [4]. The structure of information is developed over a system of time dependent axis called FCS (Finite Coordinate Space an architectural form of HyTime), which is composed by *cantus events* (notes, rests, symbols). The other domains can refer to this representation to link the several piece of music information.

SMDL vs musical standard formats

As s prototypal study of relations between SMDL and other formats we consider the well-known formats:

* *The project has been partially supported by the Italian National Research Council in the frame of the "Methodologies, techniques, and computer tools for preservation, the structural organisation, the intelligent query of musical audio archives stored on heterogeneous magnetic media" research, Finalised Project" Cultural Heritage" (Subproject 3, Topic 3.2, Subtopic 3.2.2, Target 3.2.1).*

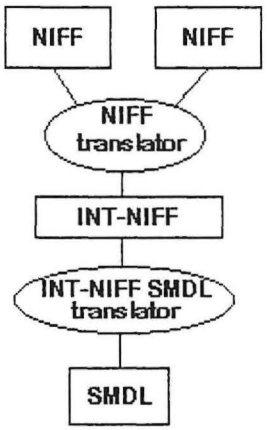


Figure 0: NIFF to SMDL translation

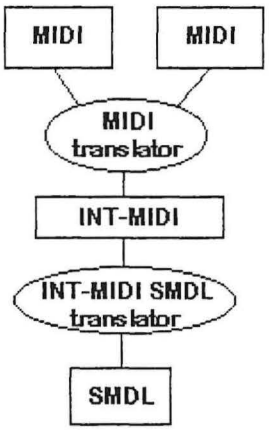


Figure 0: MIDI to SMDL translation

- MIDI: as an instance of Gestual domain;
- AIFF: as another instance of Gestual domain;
- NIFF: as an instance of Visual domain.

The essential information that must be coded in SMDL is the time representation of notes. In MIDI format, notes are coded in several tracks and channels by means of MIDI messages NOTE ON and NOTE OFF, and in this case the extraction of notes, duration and parts is straightforward.
The AIFF format is an example of audio file format containing audio signals samples. Musical information stored in this format can be handled by means of audio segmentation tools. This is one of the topics of our future work. We decided to include this type of information as a compact object that represents a particular instance of the musical piece. The AIFF format is interesting because it is a standard format designed by means of chunks (the IFF format) making the implementation easier. The new version allows the inclusion of several type of compression algorithms (a very useful facilities for files of those dimension).
The NIFF format is a new standard designed to code the notational information of music. It is the result of an effort of the major music software-houses to exchange electronic scores. Its structure is made by chunks in which are coded the objects that compose a score. The score is divided in pages, systems and staffs in which are contained the musical symbols. The way in which they are coded allows an easy extraction of information about notes, duration and parts, allowing the integration NIFF format in an SMDL environment.

Translation and integration in SMDL

We have developed prototype that extracts information of notes, durations and parts from the MIDI and NIFF files and translates them in SMDL format. We did start from a NIFF to SMDL translator made by Steve Mounce of Bradford University [11]. Two steps of translation compose this software. The first step extracts the essential information for the translation and writes it in an intermediate SGML format using the tools provided by the NIFF SDK toolkit. The second step takes the intermediate file and translates it in SMDL syntax by means of the HyMinder System (the only software-tool available to manage the HyTime Standard [10]).
Starting from this two stages we have developed a second branch of translation that takes MIDI files (in both format0 and format1), translates it in another intermediate SGML format (INT-MIDI) and by means of HyMinder system translates it in SMDL syntax.
The two intermediate formats are both written in SGML syntax. HyMinder takes the intermediate files and translates them in its internal class-representation. Then, it take SMDL DTD and writes the information received as SMDL files. The structure of the intermediate files is different because each of them reflects the structure of the respective source file, that is: INT-NIFF file reflects the structure of NIFF file and INT-MIDI file reflects the structure of MIDI file.
The third stage makes a merge of the resulting SMDL files comparing the information of notes. The result is an instance of SMDL cantus with hyperlinks to the source file as instances of Visual and Gestual domains.
Since the software developed is a prototype it works only with ad-hoc source files.

Integration with multimedia database

The problem that arises in the treatment of music files in a database environment is that they aren't directly representable in database tabular forms. This implies the conversion and the extraction of some type of data in order to allow the system to do the queries in an acceptable time [14][15].
The conversion of music files in SMDL permits to accumulate all the information in one file upon which we

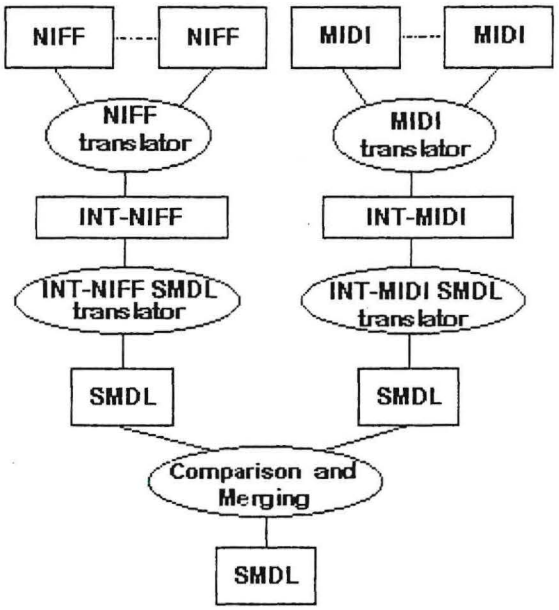


Figure 0: Translation and Merging

can extract the representative information only one time for each piece of music.
The kind of information that we extract from SMDL files is the themes, by means of which we recognize the piece. This is obtained by the implementation of score segmentation algorithms.
Now we'll take a look to a brief description of score segmentation algorithms and to the structure of an example database.

Score segmentation algorithms
The segmentation algorithms are based on the observation that most composition (quite all) have some source parts (*i.e.* brief sequence of notes) to which a great part of the piece can be brought back by means of several *music functions* [17][18][19][20]. These functions can be transposition, repetition, mirror inversion or other transformations. The themes, in most cases, are the melodies of the piece. These parts can be utilized to retrieve a particular piece among other pieces with respect to the human perception.
From this point of view we can use the segmentation algorithms to extract the themes from SMDL files and use them in tabular form as indexes of pieces for intuitive queries in the database. As intuitive queries we mean the queries where the user can input the information (melody of the piece) playing a keyboard or whistling in a microphone [13].

An example of database
The example we have sketched is based on an object relational database management system (Oracle8) and in particular upon some of its tools as "Nested Tables" and large data type. Nested tables are particular objects that allow the treatment of some tables as row of other tables [16].
During the study of musical information in relation with database management we have noticed that there can be different levels of information at which the user could access. For example a query by whistling must access a different kind of information (*i.e.* more detailed) from a catalogue query or from a musicological query (*e.g.* search how many times this sequence of notes is repeated in the piece).
So, we have divided the structure of the database into three levels:

- Database Level: information in tabular form.
- Logical Level: information in SMDL form.
- Code Level: information in source forms (NIFF, MIDI, AIFF, etc..).

In the Database Level we store catalogue information in tabular form and the themes in nested tables. In the tables below we show an example of the score's database schema: the audio database need a separate (but joined) treatment because often the several pieces refer to a "show event".
In Logical Level are contained the SMDL files extracted from source files. As explained before there can be more source files to which a single SMDL file refers.
In the Code Level are contained all source file from which the SMDL files are extracted. They are the actual source material of the multimedia database.

SCORE Table

Field	Description
Id	Score index
Author Name	
Pubblication_Date	Day , Month, Year
Pubblicazione_Place	
Edition	Editor Surname, Date (DD, MM, YYYY)
Type	Opera, Concert, ...
Instruments	Instrument's list as set of character
File_Names	Nested Table referring to code files (FILERIF)
Themes	Nested Table referring to the tables containing source parts (THEMES).

FILERIF Nested Table

Field	Description
Instrument	Score's Instrument name.
File_NIFF	Pointer to NIFF file.
File_SMDL	Pointer to SMDL file.

THEMES Nested Table

Field	Descriprion
Instrument	Scores's Instrument name.
Note_ Table	Pointer to NOTE table.

NOTE Nested Table

Field	Description
Stress	1 character.
Pitch	Number from 0 to 127; 0 is NULL value.
Duration	Integer; fraction.
Dynamic	String of character (max. 4).
Start_time	Integer defining start time (in virtual units).
Stop_time	Integer defining end time (in virtual units).

Conclusions

The discussion above is only a prototypal draft, many things can be developed and implemented:

- in 1997 a new release of HyMinder have been approved by ISO, it's necessary to incorporate this new features [8][9];
- SMDL is still in draft status and since it will change because of the changes in HyTime it's necessary to implement them;
- more complicate musical structures;
- recognition of other file formats (generic MIDI or NIFF file);
- translation and integration of audio files;

Many other improvements can be developed and implemented in order to make the system more powerful and flexible.

This article is intended as a starting study of an application of SMDL as an actual tool to manage the musical information in a database environment.

Acknowledgements

We would thank Steve R. Mounce of Bradford University for his help and for the start up information that he provided.

References

[1] D. Sloan, "Aspect of Music Representation in HyTime/SMDL" Computer Music Journal, Cambrige, MIT Press 17:4, pp 51-59,Winter 1993

[2] S.R. Newcomb, N. A. Kipp, V. T. Newcomb "The HyTime. Hypermedia/Time-based Document Structuring Language". Communication of the ACM 34:11, November 1991.

[3] C.F. Goldfarb, S.R. Newcomb, W.E. Kimber, P.J. Newcomb "Information Processing – Hypermedia/Time-based Structuring Language (HyTime) - 2nd edition", ISO/IEC JTC 1/SC 18 WG8 N1920rev, 9 Maggio 1997.

[4] S.R. Newcomb, Standard Music Description Language ISO/IEC DIS 10743 ftp.techno.com/pub/SMDL/10743.ps, June 1995.

[5] S.R. Newcomb, "Standard Music Description Language complies with hypermedia standard" COMPUTER, vol.X N.Y. 24:7, pp. 76-79, July 1991

[6] "A Brief History of the development of SMDL and HyTime" http://www.techno.com/history.html

[7] S.J. Derose, D.G. Durand, "Making hypermedia work: a user's guide to HyTime" Boston, Kluwer Academic, 1994

[8] E. Kimber, "An Excerpt from Practical Hypermedia: An Introduction to HyTime Property Sets and Groves", http://www.hight.com/IHC96/ek8.htm, 1996.

[9] "The GroveMinder System – Technical IHC 1997 Presentation" http://www.techno.com//gminder3.htm

[10] "HyMinder User Guide 0.8.4", June 1996, Techno Teacher Inc., http://www.techno.com

[11] S.R. Mounce "NIFF to SMDL translator Documentation", Software Documentation for WP3, Project Cantate

[12] Project CANTATE "Computer access to notation and text in music libraries" Project LIB-CANTATE-4-2016, Commission of the European Communities. Work Packages: WP1, WP2, WP3, WP5, Final Report.

[13] Ghias, J. Logan "Query by humming: musical information retrieval in an audio database" Proceedings [of the] ACM Multimedia 95 : San Francisco, California, November 5-9, 1995

[14] B.M. Eaglestone, "Composition tools integration with a music database system" Database and expert systems applcations : 6th International Conference ; proceedings/dexa 1995

[15] B.M. Eaglestone, "Extending the relational database model for computer music research" Computer representations and models in music, 1992

[16] M. Marchesi, "Le basi di dati orientate agli oggetti", Informatica oggi & Unix, May 1992.

[17] F. Lonati, "Metodi, algoritmi e loro implementazione per la segmentazione automatica di partiture musicali" Master Thesis in Computer Science University of Milan 1990

[18] D. Wilson, "The Role of Patterning in Music" Leonardo, Journal of the International Society for thr Arts, Science and Technology, vol. 22, Pergamon Press, Oxford, 1989

[19] G. Haus, "Elementi di informatica musicale" Jackson, Milano, 1984.

[20] F. Lerdhal - R. Jackendoff, "A Generative Theory of Tonal Music" MIT Press, Cambridge, USA 1993.

[21] E. Pollastri, G. Haus, "Metodi e prototipi software per la classificazione e il reperimento di brani audio in archivi musicali basati su tecniche di elaborazione numerica dei segnali", Master Thesis in Electronic and Computer Engineering, Politecnico di Milano, 1998.