A Database Design for Musical Information

W Bradley Rubenstein

Computer Science Division[1]
University of California
Berkeley, California 94720

ABSTRACT

As part of our research into a general purpose data management system for musical information, a major focus has been the development of tools to support a data model for music This paper first outlines the various types of information that fall under the purview of our proposed data manager We consider extensions to the entity-relationship data model to implement the notion of *hierarchical ordering*, commonly found in musical data. We then present examples from our schema for representing musical notation in a database, taking advantage of these extensions

1 Introduction

This paper reports on research, currently in progress, on the development of a database representation for musical information

Several research projects have recently focused on extending the applicability and usefulness of information management techniques and database systems to a variety of application areas In the technical field, these include design data such as is generated by VLSI (Very Large Scale Integration) chip development and other CAD (Computer-Aided Design) processes [SRG83] In the field of artificial intelligence, databases are being applied to the management of "knowledge bases" to support deduction and inference [DeF84] In each of these cases, the data model, which serves as the primary tool for describing the representation of the data, has undergone successive extension and refinement This dissertation is concerned with those extensions and refinements necessary to support applications that manage musical information

To begin, we point out certain features of music that motivate our research into musical information as an interesting data management domain

- Musical representations, such as music notation, have complex, rich semantics

In particular, they must convey more information than simple lists, tables, and spatial representations, which are the mainstay of "traditional" database applications

- The complexity of musical information is easily bounded, and therefore amenable to data management

For example, cases of ambiguity which abound in natural language are more rare (though not unknown) in music Additionally, the syntax and semantics of representations such as common musical notation are already reasonably well defined

- The uses of musical information are, in a sense, limited and well understood

For our purposes, typical examples of operations on musical data are production (e g composition and synthesis), editing, performance, and analysis Intentionality and planning, which complicate artificial intelligence problems, are considered to be outside the domain of this research

1 1 Organization

The remainder of this paper is organized as follows Section 2 introduces our notion of a database back end system for musical applications, the music data manger Section 3 then presents some related research that has supported our work in developing this system.

In section 4, we survey the various types of information which must be maintained in the music database, and existing techniques for representing this information

Section 5 describes our extensions to the entity-relationship data model that allow us to represent musical information One aspect of this model is that the distinction between schema information and musical data is somewhat blurred We consider this phenomenon in section 6

In section 7, we enumerate the various entity types that compose our database schema for common musical notation (which we will define presently) Each of several aspects of this information, including temporal, timbral and graphical attributes, are modeled in our schema We will focus on the temporal aspect, as an example of how the schema is developed.

Finally, we present our conclusions in section 8

2 The Music Data Manager

A *music data manager* (MDM) provides a service to other programs, known as *clients* (figure 1) For example, a music typesetting program would be a client, as would a musical score editor, a compositional tool, or a program which performs musicological analyses of compositions In current applications, these programs each are required to perform their own data management They have incompatible internal representations for the information they manipulate Having a single MDM manage the musical information used by each of these clients provides certain benefits

- The considerable burden, in terms of program complexity, of managing the data is no longer duplicated within every client.
- Any improvements or optimizations in the quality of data management provided by the MDM accrue to all its clients Thus, optimizing one system causes improved performance in many systems
- Because all clients maintain their information in the same way, they can more easily communicate with each other For example, a music analysis program can easily process the output of a composition program, if both have been designed to use the same MDM
- A good data model within the MDM should allow the development of clients that are faster to implement and easier to maintain, because the client need only manipulate a high-level musical information abstraction

© 1987 ACM 0-89791-236-5/87/0005/0479 75¢

[1] The author's current address is Sun Microsystems Inc mail stop A4-40 2550 Garcia Avenue, Mountain View, California 94043 This research was sponsored by the National Science Foundation under grant DMC 8504633

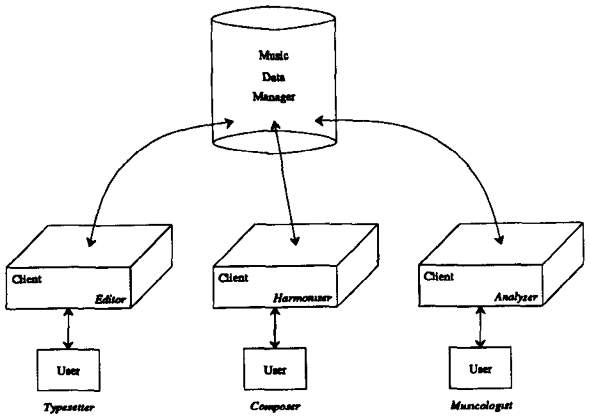


Figure 1 The Music Data Manager and Its Clients

The MDM must handle typical database operations, some standard, such as concurrency control and recovery, and some particular to the musical domain The primary extensions to traditional database systems considered by our research pertain to the modeling of music semantics, and the implementation of structures to support the physical realization of that model This requires that some decisions be made *a priori* as to what type of clients will be served by the MDM The following candidates are considered

Music editors and typesetters These systems usually manipulate a single musical *score* Examples may be found in [Byr84, MaO83, Smi72]

Compositional Tools These systems are generative they produce music, often in both sound and graphic representations See [LoA85] for a survey

Score Libraries These large collections of musical scores, often containing the complete works of a given composer or era, serve as the starting point for most musicological research A typical directory of these libraries is found in [HeS86]

Music Analysis Systems Music analysis involves applying particular operations to musical data Systems that perform various sorts of harmonic analysis, or those that determine melodic structure are examples Examples may be found in [Alp80, Gro84]

3 Research Context

Because of the interdisciplinary nature of this dissertation, several ideas are drawn from distinct bodies of research in both music and computer science These are discussed fully in [Rub87] However, in order to put our research in perspective, we briefly outline the related fields on which it is based

Our starting point for a data model for music is the entity-relationship model [Che76], which in turn is an extension of the relational model [Cod70] We have made use of other extensions to the relational model which are summarized in the RM/T proposal [Cod79] These extensions have been considered for several application domains, such as statistical databases [Sho82], scientific databases [SOW84], pictorial databases [RoL85], and computer-assisted design (CAD) databases [SRG83]

At the core of our proposal is the concept of hierarchy Data models that allow representation of hierarchies have been proposed in [Bro83, LeG78, SmS77] and implemented in systems such as GEM [TsZ84, Zan83] and GAMBIT [BDR85]

Many of the proposals in this paper stem from research on integrating abstract data types into the INGRES relational database system [Fog82, Ong82], A proposal for incorporating user-defined aggregate functions over abstract data types has also proven directly applicable to our music representation problem [Han84]

In the music domain, this research is supported by previous work in the area of music representation These include practical presentations of music notation [Don63, Rea69], as well as more theoretical analysis of notational systems [Wol77] Score representations have been explored in the DARMS system [McL86a, McL86b], and in the composition/editing domain under the SSSP project [BRB78-BPR81] Additionally, the field has seen research in artificial intelligence approaches to music representation [Roa79], and in expert systems [Ash83, Ash85] Another recently proposed score representation [Dan86] incorporates versions and multiple views into its structure, relating to database research in version control, as in [KaL82]

In both the database and music domains, the issue of managing temporal information has received considerable attention (see [BAD82] for a survey) This dissertation makes particular use of research in temporal modeling [And81, ShK86] Time has also been considered specifically in music systems [DeK85, MaM70, Pru84a] These systems are all concerned with the representation of temporal data, such as events and processes that transpire over time, multiple independent time lines, and virtual time

4 Musical Information

The information within the music manager incorporates several different facets of music, which we divide roughly into five categories,

- sound information,
- bibliographic information,
- "meta-musical" information,
- graphical information, and
- conceptual representations

Each of these types of information will be discussed in this section, demonstrating the wide variety of types of information which must be integrated into the musical data manager

4 1 Sound Representations

Obviously, one fundamental type of object which a music information manager needs to represent is the sound of the music itself The simplest representation of sound in a digital computer is merely an array of numbers, the result of *digitizing* the sound [OpS75]

Digital audio devices of professional quality typically use 16-bit integers for each sample, and record 48,000 samples per second of sound. This implies that ten minutes of musical sound can be recorded with acceptable accuracy by storing 57 6 megabytes of data

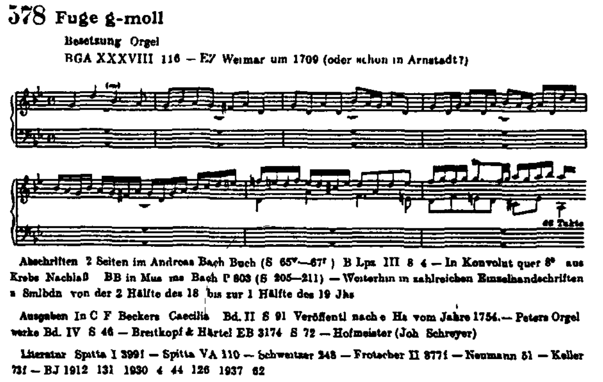
Much research in audio signal processing analyzes methods for reducing this massive storage requirement while still preserving the aurally perceptible properties of the sound From an information theoretic point of view, the digitized sound stream can be compacted in two ways by eliminating redundant information from the sound stream [Wil85], and by eliminating aurally imperceptible information from the sound stream [Kra79]

In contrast to random sound, or speech, music has a much greater burden of structure over and above that detected by these signal processing methods This structure is what differentiates music from sound. For instance, rhythmic structure (e g a "beat") and timbral structure (e g that some sounds are generated by one instrument and some by another) may exist in musical sound Such abstractions remain hidden at this level of representation

4 2 Bibliographical Information

An important use of music databases is as a reference for musicological research Such a reference may provide several types of information One common reference tool is the *thematic index* Such an index is an organization of the works of a particular composer or period, including for each work sufficient musical (i e thematic) material to identify the composition This is often a fragment of the melody or the key voices from the first several measures of the composition Figure 2 shows a typical entry in a thematic index

In addition to the thematic material that identifies the composition, several other pieces of information are provided in a highly compressed format These are the orchestration or setting of the composition (*Besetzung*), when and where it was composed, how many measures (*Takte*) it contains, where copies (*Abschriften*) of the manuscript are located, editions (*Ausgaben*) in which it is printed, and articles written about it (*Literatur*) In the language of data management, these are each bibliographic *attributes* of the composition



578 **Fuge g-moll**

Besetzung Orgel

BGA XXXVIII 116 – EZ Weimar um 1709 (oder schon in Arnstadt?)

Abschriften 2 Seiten im Andreas Bach Buch (S 65ᵛ—67ʳ) B Lpz III 8 4 — In Konvolut quer 8° aus Krebs Nachlaß BB in Mus ms Bach P 803 (S 205—211) — Weiterhin in zahlreichen Einzelhandschriften u Smlbdn von der 2 Hälfte des 18 bis zur 1 Hälfte des 19 Jhs

Ausgaben In C F Beckers Caecilia Bd. II S 91 Veröffentl nach e Hs vom Jahre 1754.— Peters Orgelwerke Bd. IV S 46 — Breitkopf & Härtel EB 3174 S 72 — Hofmeister (Joh Schreyer)

Literatur Spitta I 399f — Spitta VA 110 — Schweitzer 248 — Frotscher II 877f — Neumann 51 — Keller 73f — BJ 1912 131 1930 4 44 126 1937 62

Figure 2 A Thematic Index Entry

Once a bibliographic collection becomes established as definitive for a particular composer or body of music, the identifier created by the bibliographer may be widely understood to refer to a particular piece Thus, the accepted name for the fugue in this example is as "BWV 578 " "BWV" identifies the index (*Bach Werke Verzeichnis*), and "578" identifies the composition In this particular index, compositions are ordered chronologically

4.3 Meta-musical Information

Many of the "meanings" of musical information can be described either declaratively or procedurally For example, consider the treble clef symbol The meaning of this graphical icon might be described thus

> All subsequent notes on the same staff as the treble clef have a mapping from staff degree to scale pitch which is "*E*very *G*ood *B*oy *D*oes *F*ine" (to use a favorite grade-school mnemonic)

This meaning can be interpreted declaratively, whereby all subsequent notes have the "treble clef" pitch interpretation, or procedurally, whereby the treble clef means that subsequent note heads are to be performed (or "mapped to pitches") in a particular way In the first case, an icon determines a property of a passage In the second case, the icon tells how to interpret the subsequent notes

A more vivid example is provided by a musical *accidental* such as the sharp sign (#) A group of sharps placed at the beginning of a section of music composed in a particular style constitute a *key signature* A key signature consisting of three sharps carries a declarative meaning, stating a fact about the *tonality* of the musical passage

> The piece is in the key of A major (or f# minor)

It also carries the procedural meaning

> Perform all notes notated as F, C, or G one semitone higher than written

Much of the information contained in the music database may be derived procedurally from other declarative data in the database Suppose that the database contains, as part of a score representation, a note object. An attribute of this note would be the staff on which the note lies Another attribute would be the performance pitch of the note However, the performance pitch of a note depends procedurally (as in the above two examples) on other elements on the same staff line, such as clefs and key signatures In fact, there are other pieces of information, such as stylistic information about a composition, which govern the interpretation of performance pitch from graphical criteria. These rules constitute "meta-musical" information, and are part of the musical data to be maintained with the score

4 4 Common Musical Notation

In the case that the "listener" of a piece of music is a person (as opposed to a recording device), raw audio information is in general not sufficient for the recipient to fully understand the performance For example, the following operations, related to the transcription of sounds into scores, are difficult for human experts to execute[2] Given an audio representation (e g a recording) of a piece of music

- Determine the rhythmic structure of a composition that contains multiple independent voices
- Determine what pitches are being played, in the face of complex harmonic structures
- Determine what instruments (even assuming they are familiar to the listener) are performing which musical events

A useful written notation for music conveys the above information clearly from composer to performer, along with additional information which is similarly obscured in the audio representation

Music, like natural language, has many written forms which developed slowly over time along different paths within different cultures Although there is no universal written musical form, there is a reasonably well defined language of music notation which has been codified for Western tonal music used from about the 17th century to the present. We will refer to this as *common musical notation* (CMN)

As a "language" of musical notation, CMN has its grammatical rules These may be found in standard textbooks [Don63,Rea69] More exacting notators, such as engravers who print music, require more detailed graphical information such as is presented in [Ros70]

4 5 Other Graphical Notations

The various symbols of CMN have developed slowly over time into a reasonably stable set. However, musicians, as artists, occasionally develop their own notational extensions to CMN to better express their musical intentions (several interesting examples are collected in [Kar72]) Other types of notations are specific to particular instruments (e g lute tablature), or intended to replace CMN (e g equitone notation)

One form of notation which has received prominent attention in computer applications is the *piano roll* notation, so named because it looks similar to the rolls of punched paper used in player pianos The piano roll is essentially a map of the state of a musical keyboard against time Unlike actual player piano rolls, we typically see time progressing to the left along the x-axis, and pitch (usually quantized by semitones) increasing upward along the y-axis Figure 3 shows an example This figure shows the piano roll representation of the fragment of the Bach fugue shown in figure 2 Each note is represented by a black rectangle The entrances of the fugue, which are normally hidden in a piano roll notation, have been shaded in grey They are clearly distinguished in the CMN score by a change in note stem direction

Some existing systems have been developed to edit and display piano rolls [BSR79,Pru84b] The popularity of piano roll notation is explained by the ease of translation between note event streams (as generated by a variety of electronic music keyboard products) and piano rolls

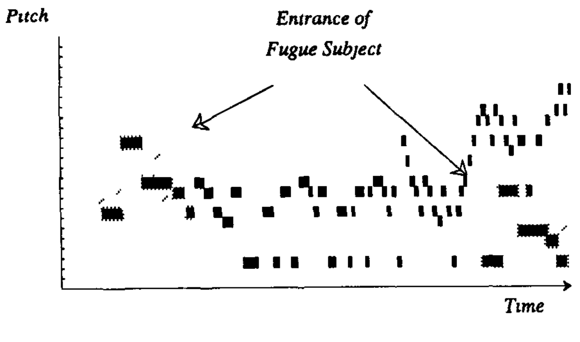


Figure 3 A Piano Roll

[2] The difficulty of these operations is one of accurately interpreting perceptual data The problems associated with this interpretation are common knowledge among those who have experience in music transcription For this reason, we may speak of music transcribers as ' experts "

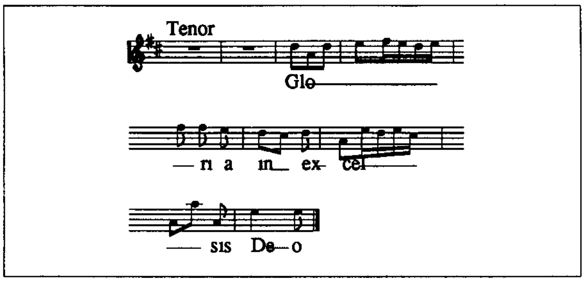
4.6 Encodings for Representations

Before discussing how these various representations are to be encoded for the data manager, let us consider the various levels at which such an encoding may be done.

In the sound domain, music may be organized into event streams, as with industry standard MIDI (Musical Information Data Interchange) event lists [Jun83]. More abstractly, it may be represented by various programming language specifications, as in the CMusic system [Moo85]. In the graphical domain, the lowest level of encoding is simply digitized (raster) graphics. This can be abstracted into its constituent graphical shapes, icons and linears, and described using a graphical definition language such as PostScript [Ado85]. Finally, a CMN score constitutes an abstract representation of the graphical aspect of a piece of music.

Several methods have been developed to represent graphical scores in a form amenable to information storage and retrieval. Such systems include DARMS (Digital Alternate Representation of Musical Scores) [Eri77, ErW83], a general purpose encoding language whose goal is to objectively represent any score material notated using CMN. MUSTRAN [Wen77] is similar to DARMS, although its focus is on ethnomusicological material. Smith's system, SCORE [Smi72, Smi73] (now known as MS), is oriented toward producing very high quality graphical output. This system has interactive score editing tools that give the user very fine control over the music typesetting process.

As an example of these graphical languages, figure 4 shows a small piece of music, along with its DARMS encoding. This system was intended to encode musical scores onto punch cards (the project was started by Stefan Bauer-Mengelberg in the 1960's). It generally utilizes one letter codes for each attribute of an object found on the score. Numbers are used typically to indicate vertical position. 21 (or 1 for short) is the bottom line, 22 is the bottom space, and so forth. The other abbreviations are summarized in figure 4(c).



(a) A Fragment of Music

```
I4 'G 'K2# 00@¢TENOR$ R2W /
          (7,@¢GLO-$ 4 7) / (8 (9 8 7 8)) /
9E 9,@RI-$ 8,@A$ / (7,@IN_$ 6) 7,@EX-$ /
          (4D,@CEL-$ (8 7 8 6̄)) /
(4D 31) 4,@SIS$ / 8Q,@¢DE-$ E,@O$ //
```

(b) Its DARMS Encoding

Abbreviation	Meaning
I4	Instrument (or voice) definition #4
'G	G (treble) clef
'K	Key signature ('K2# two sharps)
00	Annotation above the staff
R	Rest (two whole rests)
@*text*$	Literal string
¢	Capitalize next letter
(*notes*)	Beam grouping
W	Whole duration
Q	Quarter duration
E	Eighth duration
D	Stems down
/	Bar line

(c) Abbreviation Key for the DARMS Encoding

Figure 4. DARMS Encoding (from [Eri77])

DARMS has a very flexible input protocol, allowing information to be entered from the page in a variety of orders (a measure at a time, whole lines at a time, etc.). Also, redundant information can often be suppressed, so that repeated note durations or pitches can be rapidly entered. Programs have been written to convert this "user DARMS" into "canonical DARMS" (the programs have been whimsically named "canonizers"). A canonical DARMS encoding presents the score information in a consistent order, and explicitly includes all repeated information [ErW83, McL86b]. Systems to generate a graphical CMN score from a DARMS encoding have also been designed [Gom77].

5 Adding Hierarchical Ordering to the Entity-Relationship Model

We use the entity-relationship data model [Che76] as the basis for describing musical structures. Each structure is represented in the database by an *entity*. In order to represent the relationships among these structures, we introduce the concept of *hierarchical ordering* as a tool for data modeling. We use three complementary representations for describing hierarchical ordering:

- *Instance graphs* as a pictorial representation of hierarchically ordered data,
- A *data definition language* (DDL) for hierarchical ordering,
- *Hierarchical ordering graphs* (HO graphs) to represent hierarchical ordering at the database schema level.

5.1 The Entity-Relationship Model

As a basis for the discussion which follows, we briefly review the entity-relationship model. The domain to be modeled is represented by a variety of *entity types*. In the musical score domain, examples of entity types include compositions, measures, chords, notes, staves, and so on.

The actual objects within the domain are represented by *entity instances*. Every entity instance of a given type has a set of *attributes* associated with it. Within every entity instance of a particular type, each attribute is assigned a distinct *value*. For example, according to the definition

define entity COMPOSITION (title = **string**)

every composition is defined to have a title, and the value of that title is typically different for each composition.

Relationships between entities take two forms: "*m* to *n*" relationships, and "1 to *n*" relationships. In the (admittedly unusual) case where multiple people might compose a single composition, and a single person might compose many compositions, the COMPOSER relationship between the PERSON entity and the COMPOSITION entity is an "*m* to *n*" relationship. Associated with each composition is a single date that determines when a composition was written, though many compositions might be composed on the same date. Thus the COMPOSITION_DATE relationship between the DATE entity and the COMPOSITION entity is a "1 to *n*" relationship.

These entities and relationships may be expressed as follows:

define entity DATE
    (day = **integer**, month = **integer**, year = **integer**)
define entity COMPOSITION
    (title = **string**, composition_date = DATE)
define entity PERSON
    (name = **string**)
define relationship COMPOSER
    (person = PERSON, composition = COMPOSITION)

Note that the "1 to *n*" relationship for composition date is represented implicitly as an attribute of the COMPOSITION entity. Chen introduces a pictorial notation for representing entities and relationships. The diagram for the above example is given in figure 5. In this type of representation, entity types are shown in rectangular boxes, and relationships are shown in diamond-shaped boxes. Lines are drawn from relationships to the entities which they reference. The type of the relationship (*m* to *n* or 1 to *n*) is indicated on these lines.

5.2 Ordering

Neither the relational model, nor the entity-relationship model incorporates any concept of ordering among elements stored in the database. Actual relational database systems, on the other hand, usually implement some form of ordering among data records. This is typically provided by

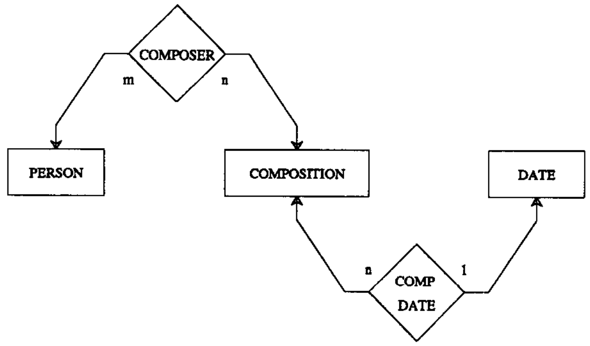


Figure 5 An Entity-Relationship Graph

allowing the database designer to designate *key* attributes for a relation, allowing the system to sort the data records so that they are ordered by ascending (or descending) key value

This use of ordering may be seen purely as a performance optimization in relational databases An important relational operation is to select data records that have a particular key value (or range of key values) This may be efficiently performed on relations that are sorted, because the desired records are all stored together, rather than being randomly distributed throughout the relation [3]

In contrast to this, we are interested in modeling a domain where an important attribute of the data is the participation of entities in various orderings For example, a musical score consists of an ordered set of measures of music, and the fact that one measure follows another measure is a concept which must be modeled by the database definition

5 3 Instance Graphs

In its most general form, *hierarchical ordering* occurs when a group of database objects (of one or more types) forms an ordered set associated with a distinct parent object For instance, a particular set of notes aggregate to form a given chord An *instance graph*, such as the one in figure 6, shows this relationship pictorially This graph, in its entirety, could represent, for example, a four note chord It consists of a parent, y, and an ordered set of children, $\{ u , v , w , x \}$ The ordering among the children is indicated here by arrows from one child to the next one in the ordering Such edges of the graph are called S-edges, as they indicate a relationship among siblings Each child also has a relationship with its parent, indicated in the example by P-edges Notice that each child has an ordinal position under its parent For example, we may speak of the node w in this figure as the third child of the parent labeled y [4]

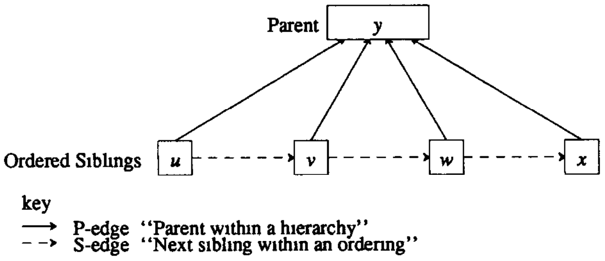

Parent
y
Ordered Siblings
u
v
w
x
key
→ P-edge "Parent within a hierarchy"
--> S-edge "Next sibling within an ordering"

Figure 6 A Simple Instance Graph

5 4 Defining Hierarchical Ordering in a Schema

In an aggregation hierarchy [SmS77], the number and type of elements in the aggregation are fixed by the schema For example, a piano is an aggregation of one keyboard, a fixed number of strings, a sounding mechanism, and a bench A piano bench in turn is an aggregation of four legs and a cushion For entities in the musical score, this characterization is insufficient Specifically

- The number of objects in an aggregation is typically not fixed For instance, under the aggregation of notes into chords, different chords typically have different numbers of notes
- The objects in an aggregation are ordered For instance, given two measures in a score, one must be prior to the other

These characteristics distinguish hierarchical ordering from aggregation hierarchies The **define ordering** statement is included in our DDL to model hierarchical ordering [5]

The syntax for the **define ordering** statement is

```
define_ordering_statement
    define ordering [ order_name ] ( child_entity { , child_entity } )
        [ under parent_entity ]

child_entity
    entity_name

parent_entity
    entity_name
```

One such statement defines a single instance of hierarchical ordering "Order_name" is the name of the ordering This is followed by one or more child entity names whose instances will participate in the ordering The **under** clause specifies the relation from which parent entities are taken, determining the type of the entity instance under which each ordering will be grouped A schema containing musical notes ordered within chords would be specified as

define entity CHORD (*chord attributes*)
define entity NOTE (*note attributes*)

define ordering note_in_chord (NOTE) **under** CHORD

In this simple example, the ordering is named "note_in_chord " It consists of a single child type, NOTE, under the parent type, CHORD This schema definition would allow queries to retrieve, for instance, "the third note in chord x " The query language extensions to accomplish this are described below

The semantics of various forms of the **define ordering** statement, as when the order name is missing, or when there are multiple child types, will be the focus of the next section

5 5 Types of Hierarchical Ordering

It will generally be more convenient to present ordering definitions in pictorial form We therefore make use of the *hierarchical ordering graph* (HO graph), an example of which is shown in figure 7 This graph represents a single ordering In general, each edge in the HO graph corresponds to one **define ordering** statement Hierarchical ordering may take several forms

Multiple Levels of Hierarchy An object that is a parent in one ordering may be a child in another This type of ordering is quite common in music For example, we can order a set of notes (say, from high pitch to low pitch) under a parent chord A set of chords can then be organized temporally into a measure of music The statements to represent such a schema are

define ordering (NOTE) **under** CHORD
define ordering (CHORD) **under** MEASURE

[3] Of course this is only true if the desired selection is compatible with the choice of sort key For instance, a relation sorted on composition title cannot efficiently support a selection based on composer name

[4] The pointers in an instance graph should not be misinterpreted as an indication of the physical implementation of these objects within the data manager They merely serve to indicate graphically the ordering and hierarchy among objects

[5] The syntax for our data definition language (DDL) is presented in detail in [Rub87] We use standard BNF [Bac59] for syntax descriptions Clauses in square brackets are optional and clauses in braces may be repeated zero or more times

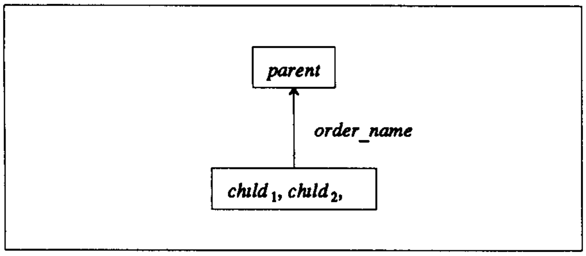


Figure 7 An HO graph for a Single Ordering

Multiple Orderings Under a Parent This type of ordering schema occurs in a musical score, where both instrumental parts and staves are ordered on the score page within an instrument (e g the portion of a score system dedicated to the violin instrument may contain three violin parts, notated on two staves) This could be represented by the following statements

define ordering (PART) **under** INSTRUMENT
define ordering (STAFF) **under** INSTRUMENT

Inhomogeneous Orderings The set of siblings in a particular ordering may not be of homogeneous type In a score, a musical *voice* consists of an ordered sequence of *chords* and *rests*, intermixed (this is a simplified view, for the purpose of this example)

define ordering (CHORD, REST) **under** VOICE

Every rest and chord, by our definition, has some voice as parent. The element at a particular position of the ordering, say, "the second object under voice *V*," must be either a chord or a rest. Of course, it can't be both, since there is only one "second object." This differs from the previous case, where a parent covered two child types under *different* orderings, then it made sense to speak of "the second part for the violin instrument" as well as "the second staff for the violin instrument."

Multiple Parents Another possible configuration is for an entity to have multiple parents For example, a note has a chord as parent, under the ordering named "ordered set of notes per chord." A note also has a staff as parent, under the ordering "next note per staff"

define ordering (NOTE) **under** CHORD
define ordering (NOTE) **under** STAFF

We can see that these two orderings are independent. A chord may lie on multiple staves, so two notes that are members of the same "per chord" ordering are not necessarily members of the same "per staff" ordering

Recursive Ordering Suppose that the parent in an ordering is of the same type as one of the children In that case, the ordering is recursive An example from music would be found in the grouping of chords under beams A *beam groups* consists of an ordered set of smaller beam groups intermixed with chords This would be defined as follows

define ordering (BEAM_GROUP, CHORD) **under** BEAM_GROUP

The HO graph for this ordering is shown in figure8(a) Figure 8(b) contains a fragment of musical notation with several layers of beam groups The six chords in this fragment are labeled c_1 to c_6 The instance graph for the chords and beam groups is shown in figure 8(c) Every object in this instance graph is either a group (labeled g_i) or a chord (labeled c_i)

Certain restrictions on recursive ordering are necessary, to prevent the occurrence of instance graphs that are malformed One difficulty arises if the P-edges for a given ordering form a cycle Because this would mean that an instance is "part of" itself, such cycles in the instance graph are disallowed Similarly, cycles among the S-edges of a given ordering are not permitted, because they result in the situation where an object is "before itself" in the ordering

5 6 Manipulation of Ordered Entities

We use QUEL [Rel84] as a basis for our data manipulation language Three new operators are added to QUEL to support hierarchical

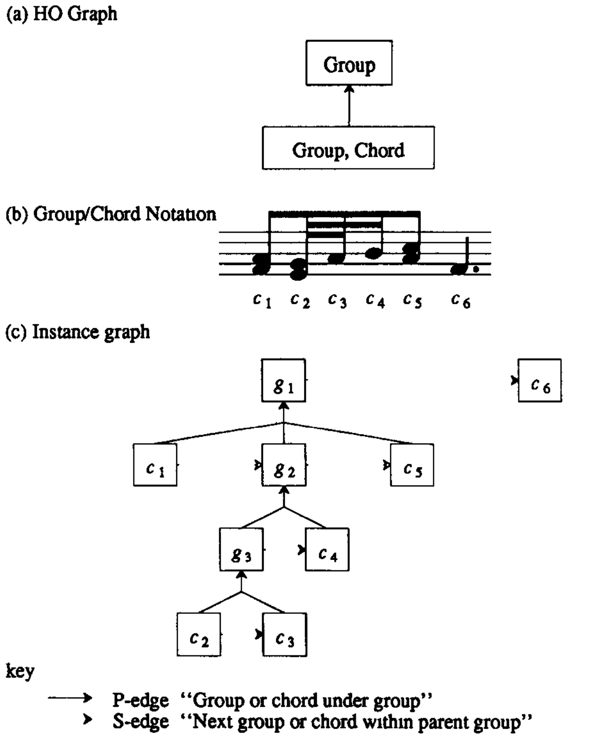


Figure 8 An Example of Recursive Hierarchical Ordering

ordering **before, after,** and **under** Unlike other QUEL operators, the ordering functions operate on entities (represented by range variables in QUEL), rather than on attribute values In this way they are similar to the entity equivalence operator, **is**, introduced in the GEM extensions to QUEL [Zan83] The following example, using the is operator, finds all the composers of "The Star Spangled Banner"

Schema definition
  define entity PERSON (name = **string**,)
  define entity COMPOSITION (title = **string**,)

  define relationship COMPOSER
    (composer = PERSON, composition = COMPOSITION)

Query[6]
  retrieve (PERSON name)
    where COMPOSITION title = "The Star Spangled Banner"
    and COMPOSER composition **is** COMPOSITION
    and COMPOSER composer **is** PERSON

Unlike other operators, the is operator takes entities (i e range variables) rather than attribute values as operands

The ordering operators each take two range variables and an optional ordering name as operands The syntax for a qualification using the "before" operator is representative

before_clause
  range_variable **before** range_variable [**in** order_name]

The **after** and **under** operators have similar syntax For the **before** and after operators, the types of both range variables are taken from the child types of the ordering indicated by "order_name " For the **under** clause, the type of the first range variable is taken from the children of the ordering, and the type of the second is the parent type in the ordering The

[6] As in GEM and later versions of INGRES, a range variable with the same name as its entity type is implicitly declared for each entity type

clause,

```
a before b in order_name
```

evaluates to "true" if *a* and *b* both have the same parent with respect to the hierarchical ordering indicated by *order_name*, and *a* is before *b* in that ordering. If *a* and *b* have different parents, then they are not comparable, and the **before** clause evaluates to "false".

Given these definitions of NOTE and CHORD,

```
define entity CHORD (name = integer, other chord attributes )
define entity NOTE (name = integer, other note attributes )

define ordering note_in_chord (NOTE) under CHORD

range of n1, n2 is NOTE
range of c1 is CHORD
```

here are examples of the use of the ordering operators.

Given a note *n*, retrieve the notes prior to *n* in its chord.

```
retrieve (n1.name)
    where n1 before n2 in note_in_chord
    and n2.name = n
```

Retrieve the notes that follow note *n*.

```
retrieve (n1.name)
    where n1 after n2 in note_in_chord
    and n2.name = n
```

Retrieve the notes under chord *c*.

```
retrieve (n1.name)
    where n1 under c1 in note_in_chord
    and c1.name = c
```

Retrieve the parent chord of note *n*.

```
retrieve (c1.name)
    where n1 under c1 in note_in_chord
    and n1.name = n
```

6 Blurring the Schema/Data Distinction

An important feature of the musical schema is that many entity types (as opposed to entity instances) have attributes associated with them. These attributes are similar to "class variables" in object-oriented systems. Although they are supported in object-oriented languages such as Smalltalk [GoR83], they are not typically available in relational or entity-relationship database systems.

6.1 Storing the Schema Definition as Ordered Entities

Where the schema itself is maintained as data in the database, as in the INGRES system [Rel84], these class variables may be simulated by referring to parts of the schema within the data. We may actually use our data definition language to define a meta-database: a database that models our definitions of entities, relationships, attributes and orderings.

The meta-definition for a schema is therefore

```
define entity ENTITY
    (entity_name = string)
define entity RELATIONSHIP
    (relationship_name = string)
define entity ATTRIBUTE
    (attribute_name = string, attribute_type = string)
define entity ORDERING
    (order_name = string, order_parent = ENTITY)

define ordering entity_attributes
    (ATTRIBUTE) under ENTITY
define ordering relationship_attributes
    (ATTRIBUTE) under RELATIONSHIP
define relationship order_child
    (child = ENTITY, ordering = ORDERING)
```

The HO graph for this meta-schema is shown in figure 9. It shows the many to many ("*n* to *m*") relationship between child entities and and orderings, and the "1 to *n*" relationship between parent entities and orderings (this latter relationship is implicitly specified in the ORDERING entity in the above definition). The hierarchical ordering of ATTRIBUTE under ENTITY and RELATIONSHIP is also shown.

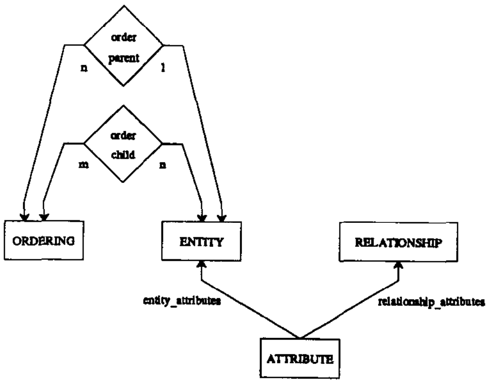


Figure 9. A Hierarchically Ordered Meta-Schema

A schema definition may be stored in such a database. Each **define entity** statement generates an instance of an ENTITY entity and several instances of ATTRIBUTE entities (one per attribute). Similarly, each **define relationship** statement generates an RELATIONSHIP instance and several ATTRIBUTE instances. Each **define ordering** statement generates one ORDERING entity, a single parent relationship, and one or more child relationships.

6.2 Referencing the Schema

A database modeled as in the previous section contains two levels of information.

(1) Entity types reflecting the data model, and containing the schema definition (e.g. ENTITY and ATTRIBUTE), and

(2) Entity types that implement the schema definition, and contain data instances (e.g. CHORD and NOTE).

We find it useful to insert an additional level of information into the database, resulting in three layers.

(1) Entity types determining the data model, and containing the schema definition,

(2) Entity types reflecting the particular data domain, and containing domain-specific attributes of entities referenced by the schema definition.

(3) Entity types that implement the schema definition, and contain data instances.

As an example of this middle level, we will discuss a property of many musical entity types: their graphical representations. Consider the statement to define the STEM entity in the CMN database. This gives the location and size of stems associated with particular chords. Its definition is

```
define entity STEM
    (xpos = integer,
    ypos = integer,
    length = integer,
    direction = integer)
```

The STEM entity is catalogued in the ENTITY relation. Its four attributes are catalogued in the ATTRIBUTE relation. Associated with the STEM entity is additional information that describes how stems are drawn.

Figure 10 shows the two "schema" entity types, ATTRIBUTE and ENTITY, and a new "GraphDef" entity type. Each GraphDef entity contains the graphical definition (e.g. PostScript function) to draw a particular object. Additionally, two relationships are shown. The "GParmUse" relationship identifies which attributes serve as parameters to a graphical function, and the "GDefUse" relationship associates graphical definitions with entity types. In contrast to the ENTITY and ATTRIBUTE relations, which are used for schema definitions in any domain, these three new relations, GraphDef, GParmUse, and GDefUse, are application specific. But

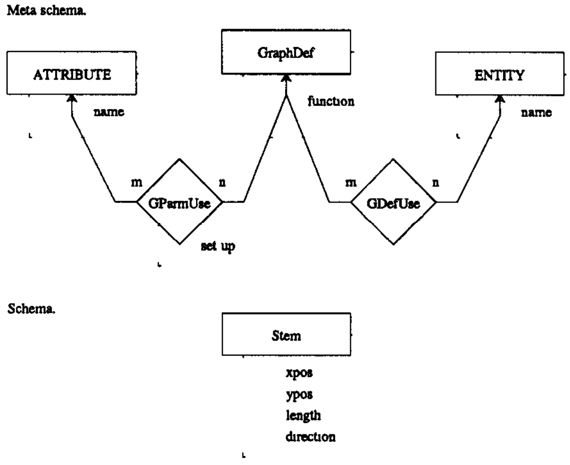


Figure 10 Schema for Graphical Definitions

instead of modeling instance data, like other application specific relations (such as STEM), these relations model information specific to the domain of musical notation (e g that a stem is to be drawn in a particular way)

The STEM entity, as defined in the ENTITY relation, is associated in GDefUse with the graphical definition that draws a stem Associated with each stem are particular parameters, xpos and ypos (defining the endpoint of the stem), the length of the stem, and its direction (either up or down) These attributes, as defined in the ATTRIBUTE relation, are associated with the graphical definition by instances in the GParmUse relationship Each such instance also contains the PostScript fragment that sets up the particular attribute value for the given procedure

The process for drawing a particular stem instance is as follows

(1) Find the stem instance in the STEM relation

(2) Find the graphical definition for the STEM entity type via the GDefUse relationship

(3) For each parameter of this definition, found via GParmUse, get its value from the stem relation and execute the set up code as given in GParmUse

(4) Then execute the graphical definition given in GraphDef

The client program must actually move back and forth between using the schema definition (in ENTITY, ATTRIBUTE, and GraphDef) and the musical data itself (in STEM) in order to provide this type of functionality By making this schema definition accessible as data, the client program may freely modify such attributes as the printing function for a graphical object

7 A Database Schema for Common Musical Notation

In order to allow a user to refer to meaningful units of musical information, we must first determine what those units are This section analyzes in detail a subset of the entities that compose CMN, and their interrelationships

7 1 CMN Entities

In many data management domains, there are only a handful of entities For example, the standard company database contains employees, jobs, departments, parts, suppliers, and orders CMN has, even at first glance, many more entities than this These entities are summarized in figure 11, and will be discussed in the following sections

7 1 1 Aspects of CMN

Each musical entity contains various attributes For example, attributes of a "note" entity are its position, shape, size, start time, parent chord, and so on Musical entities in the CMN score have several aspects and subaspects, as shown in figure 12 These may be thought of as different views on the musical schema. Roughly, the temporal aspect pertains to *when* musical events are performed The timbral aspect refers to *how* they are performed (e g by what instrument, at what pitch, how loudly, etc) This aspect itself admits a finer characterization, into pitch, articulation, and dynamic (i e volume) subaspects of the data. The graphical aspect is concerned with how musical events are notated graphically A subaspect within the graphical aspect of the score is concerned with textual material, including a variety of score annotations, as well as the lyrics (or *libretti*) associated with melodies

Entity type	Description
Score	The unit of musical composition
Movement	A temporal subsection of the score
Measure	A temporal subsection of the movement
Sync	Sets of simultaneous events
Group	A group of contiguous chords and rests in a voice
Chord	A set of notes in one voice at one sync
Event	An atomic unit of sound, one or more notes
Note	An atomic unit of music, a pitch in a chord
Rest	A "chord" containing no notes
MIDI	A MIDI note event.
MIDI control	A MIDI control event at a point in time
Orchestra	A Set of Instruments performing a Score
Section	A family of instruments
Instrument	The unit of timbral definition
Part	Music assigned to an individual performer
Voice	The unit of homophony
Text	In vocal music, a line of text associated with the notes
Syllable	The piece of text associated with a single note
Page	One graphical page of the score
System	One line of the score on a page
Staff	A division of the system, associated with an instrument
Degree	A division of the staff (line and space)
Graphical Definitions	All the graphical icons and linears
Instrument Definitions	Instrument patches and specifications
Other graphical attributes	Accents, Accidentals, Annotations, Arpeggii, Barlines, Beams, Clefs, Duration dots, Fingerings, Flags, Hairpins, Key signatures, Meter signatures, Note heads, Rests, Slurs, Staff lines, Stems, Ties, Letters, etc

Figure 11 The Entities of a CMN Schema

The utility of this notion of aspect may be suggested by example A musical note, as it appears on a score page, possess attributes associated with each of these aspects

The temporal aspect of a musical entity refers to those attributes and relationships which model the entity's placement in time A note has attributes related to the time at which it is performed in the course of a composition

Because CMN groups musical events by "instrument," we can speak of the timbral aspect of certain entities A note has a timbral aspect that refers to the instrument that "performs" it

A note may have several attributes reflecting its pitch aspect. These include such things as its staff degree, associated accidentals, and relations to key signatures and clefs There is also a notion of performance pitch (either MIDI key codes or frequency information) that is indirectly associated with notes

A note inherits various articulative attributes These reflect roughly how the note is performed They include modal attributes such as *staccato* (shortened or clipped) or *marcato* (marked or stressed) Also, a note may

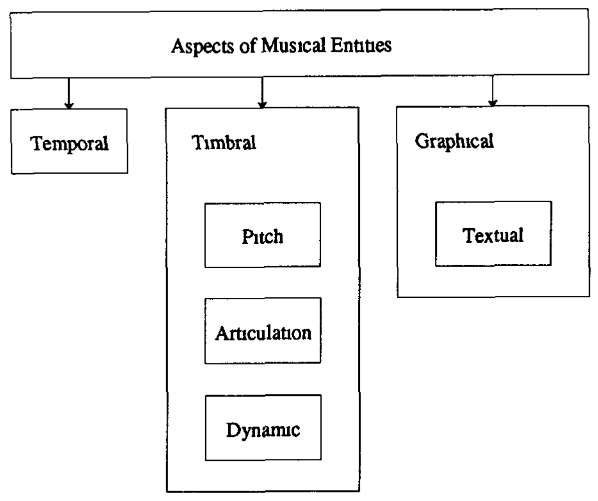


Figure 12 Aspects of Musical Entities

have inherited various performance attributes, such as when a violin note is played *pizzicato* (plucked) or *arco* (bowed)

Another attribute which a note must inherit is its dynamic value, which indicates how loudly it is to be played In the graphical score, these are given as annotations such as *forte* (loud) or *pianissimo* (very soft) Such attributes are not typically assigned directly to a note, but rather are inherited by the note from the context in which it lies

Finally, since CMN is a graphical notation, musical entities have a graphical aspect, relating to their representation on the written page For a note, this includes its various graphical components, such as the note head, stem, associated accidentals, flags, dots, accents, and so on Each of these has a shape or size and location on the page These are all graphical attributes A subclass of graphical objects on the score page may be considered to be textual objects Although individual note entities do not have a textual aspect, there are a variety of textual annotations associated with pages, systems, staves, syncs and individual chords

We use two strategies to organize the representation of musical entities First, we arrange the entities into groups by the aspects in which their attributes participate, then we define a HO graph for the entities in each group Towards the top of each graph will be abstract structures that give form to the music At the bottom of the graph will be the low level objects that make up the physical attributes of the music Not every entity has attributes in every aspect (MIDI events, for example, have no graphical aspect in CMN) Many entities, as we have seen for notes, appear in the graphs for several aspects

As an example of the definition of these we will only consider the HO graph for the temporal aspect. The complete set of graphs is given in [Rub87]

7 2 The Temporal Aspect

Before discussing the entities involved in the temporal aspect of a CMN score, we must define certain uses of the word "time" in music Specifically, we distinguish between "performance time" and "score time "

The location in time at which a musical event is actually initiated, and how long it lasts, are recorded in performance time The units of performance time are seconds Score time, on the other hand, is measured in rhythmic units Musical structures in CMN, such as notes, chords and measures, may fall into a more or less regular rhythmic structure whose unit is the *beat*

The duration of a beat, however, is consistently distorted in performance This distortion may be noted in the score, by directives such as *accelerando* to speed up a passage or *ritardando* to slow down Alternatively, they may be inherent in the style of the music, as in the *rubato* associated with certain musical styles Thus the mapping between the location of events in score time, and their location in performance time, may be arbitrarily complex When an orchestra performs, it is the role of the conductor to establish this relationship between score time and performance time

We now consider the HO graph for temporal attributes The relationships among the temporal aspects of musical entities are shown in figure 13 To review the elements of the graph, each box contains one or more entity types The solid arrows refer to hierarchical ordering of child types under a parent type, while the dotted arrows indicate hierarchical ordering under entities not shown in this graph (they appear under other aspects) The indirect relationship indicated by the dotted lines arises because we are not considering the HO graph for the entire entity set at one time

A musical *score* is the compositional unit of the database Its temporal attribute is the duration of the composition This duration is the sum of the durations of its constituent *movements* A movement is a somewhat arbitrary (though widely used) unit of performance These movements are further subdivided in time, into *measures* Measures determine rhythmic divisions of a passage Where a musical passage has a rhythmic pulse (i e a *beat*), each measure consists of an integral number of such pulses

The various musical events within a passage (such as notes) are typically aligned on these pulses Each such point of alignment constitutes a *sync* This term is taken from the Mockingbird system [MaO83] A sync has, as a temporal attribute, the point in score time at which it occurs This can be specified as a number of beats (units of score time) from the start of the measure in which the sync occurs Figure 14 shows how a measure is divided into syncs The notes within a sync are grouped into *chords* (by voice, as we shall see in the timbral definition) The start times of notes and chords are inherited from their parent syncs

In addition to the grouping of chords into syncs into measures, particular musical voices may be independently organized into melodic *groups* Groups have a variety of semantic functions in music As shown in figure 15, these include phrasing (e g notes covered by a slur) and timing (e g beams and tuplets) A group has a the temporal attribute, "duration," which is a function of the duration of its constituent *chords* and *rests*

Rests, like chords, have temporal location and duration, although they result in no performance (MIDI) information

An *event*, from the temporal point of view, determines the placement in time of each atomic unit of sound It has a unique start and end time, and is performed by a specific voice An event is thus a unit of performance A *note*, on the other hand, is the notated unit of music These two are not necessarily the same, as, for example, when two notes are tied together The *Tie* is a musical construct that binds multiple note entities under a single event entity

At the bottom of the graph appears the *MIDI* entity We assume a MIDI model [Jun83], where individual musical "events" have particular starting and ending times For scores that use CMusic style note lists, these can easily be extrapolated from the MIDI event information MIDI events constitute performance information, and so their temporal parameters are given in performance time (i e seconds) There are MIDI commands to control note events, as well as control information such as the actuation of a control switch other than a keyboard key (e g the *sostenuto* pedal of a piano)

8 Conclusion

Musical information seems to provide a good domain for the exploration of a wide variety of data management issues It consists of several different types of data, including sound, graphics, text, and conceptual abstractions Each of these data types has its own peculiarities of representation and manipulation

In the course of developing an entity-relationship database schema for common musical notation, additional tools to adequately represent ordering and hierarchy were needed We have formally extended the entity-relationship model with the concept of hierarchical ordering to fill this need

This semantic relationship occurs when an entity type (the parent) "consists of" an ordered aggregation of instances of another (the child) A collection of such parent-child relationships may form hierarchies We

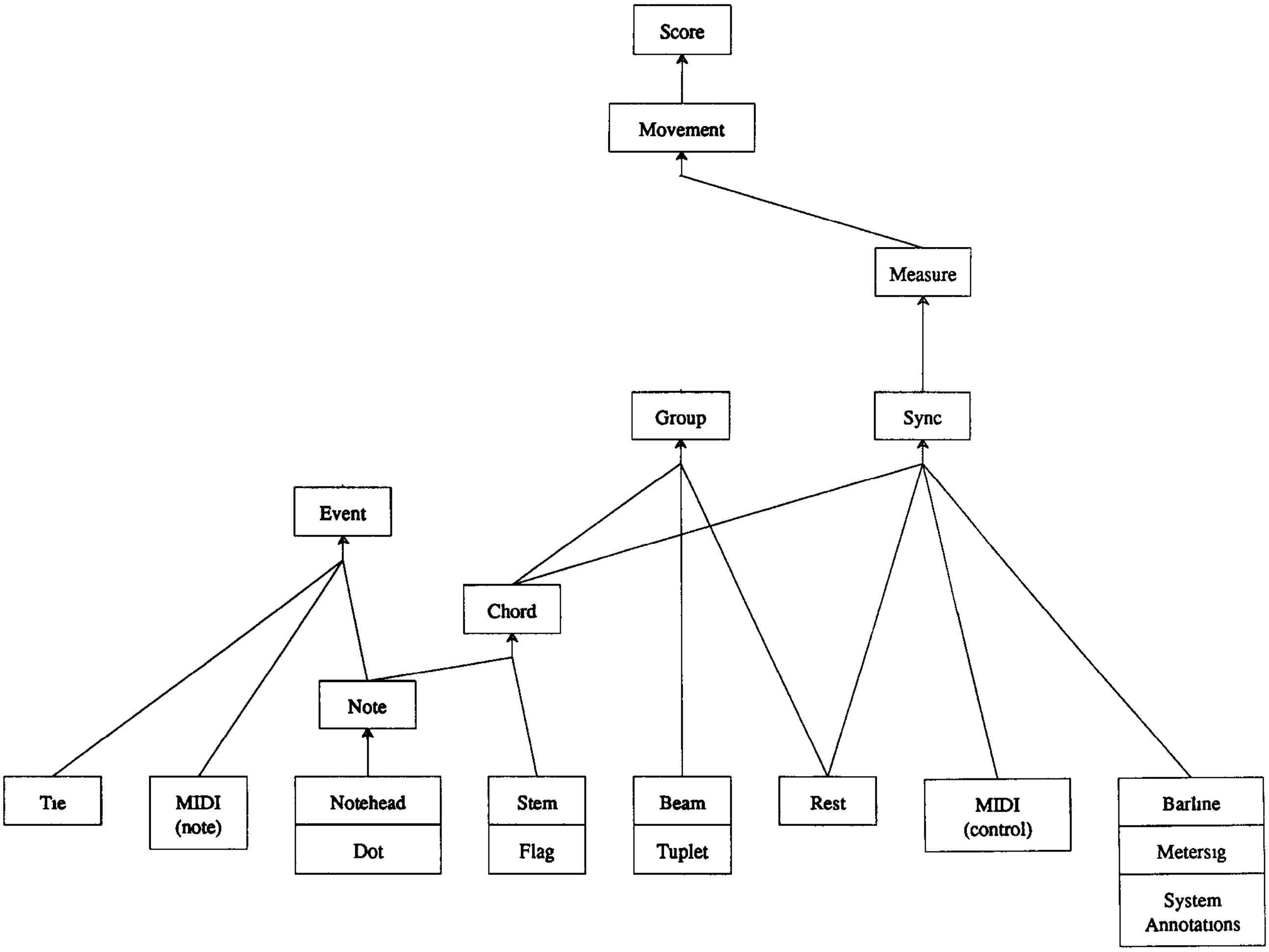
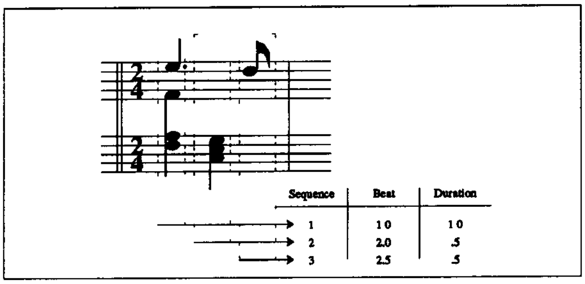


Figure 13 Temporal Relationships in the CMN Schema



Sequence	Beat	Duration
1	1 0	1 0
2	2.0	.5
3	2.5	.5

Figure 14 Dividing a Score into Syncs

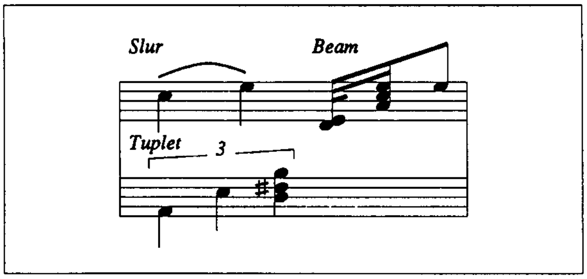


Figure 15 Examples of Chord Groups

have developed a graph model for these relationships, which we find to be a useful tool for modeling many different aspects of musical information As an example of its utility, we have presented a portion of our schema that models those entities within the musical score which exhibiting temporal attributes Because of the high level of interest with which database research has focused on maintaining temporal information (for one example), we hope that these modeling tools may find broader applicability outside the music domain

References

[Ado85] Adobe Systems, *PostScript Language Reference Manual*, Addison-Wesley, Reading, MA, 1985

[Alp80] Alphonce, B , "Music Analysis by Computer", *Computer Music Journal 4*, 2 (Summer 1980), 26-35

[And81] Anderson, T L, *The Database Semantics of Time*, Ph D Dissertation, University of Washington, 1981

[Ash83] Ashley, R , "Production Systems Three Applications in Music", *Proceedings of the International Computer Music Conference*, Rochester, NY, 1983, 160-172

[Ash85] Ashley, R D, "KSM An Essay in Knowledge Representation in Music", *Proceedings of the International Computer Music Conference*, Burnaby, British Columbia, 1985, 383-390

[Bac59] Backus, J , "The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference", *Proceedings of the International Conference on Information Processing*, 1959, 125-132

[BAD82] Bolour, A , Anderson, T , Dekeyser, L and Wong, H , "The Role of Time in Information Processing A Survey", *ACM SIGMOD Record 12*, 3 (1982), 27-50

[BDR85] Braegger, R , Dudler, A , Rebsamen, J and Zehnder, C , "Gambit An Interactive Database Design Tool for Data Structures, Integrity Constraints, and Transactions", *IEEE Transactions on Software Engineering SE-11*, 7 (July 1985), 574-583

[Bro83] Brodie, M L , "Association A Database Abstraction for Semantic Modeling", in *Entity-Relationship Approach to Information Modeling and Analysis*, P Chen (editor), Elsevier Science Publishers, Amsterdam, 1983, 577-602

[BRB78] Buxton, W , Reeves, W , Baeker, R and Mezei, L , "The Use of Hierarchy and Instance in a Data Structure for Computer Music", *Computer Music Journal 2*, 4 (Winter 1978), 10-20

[BSR79] Buxton, W , Sniderman, R , Reeves, W , Patel, S and Baeker, R , "The Evolution of the SSSP Score Editing Tools", *Computer Music Journal 3*, 4 (1979), 14-26

[BPR81] Buxton, W , Patel, S , Reeves, W and Baecker, R , "Scope in Interactive Score Editors", *Computer Music Journal 5*, 3 (Fall 1981), 50-56

[Byr84] Byrd, D , *Music Notation By Computer*, Ph D Dissertation, Department of Computer Science, Indiana University, 1984

[Che76] Chen, P , "The Entity-Relationship Model Toward a Unified View of Data", *ACM Transactions on Database Systems 1*, 1 (March 1976), 9-36

[Cod70] Codd, E , "A Relational Model of Data for Large Shared Data Banks", *Communications of the ACM 13*, 6 (June 1970), 377-387

[Cod79] Codd, E F, "Extending the Database Relational Model to Capture More Meaning", *ACM Transactions on Database Systems 4*, 4 (December 1979), 397-434

[Dan86] Dannenberg, R , "A Structure for Representing, Displaying and Editing Music", *Proceedings of the International Computer Music Conference*, The Hague, Netherlands, 1986, 153-160

[DeK85] Decker, S L and Kendall, G S , "A Unified Approach to the Editing of Time-Ordered Events", *Proceedings of the International Computer Music Conference*, Burnaby, British Columbia, 1985, 69-78

[DeF84] Deering, M and Faletti, J , "Database Support for Storage of AI Reasoning Knowledge", *Proceedings of the First International Workshop on Expert Data Base Systems*, Kiawah, SC, October 1984

[Don63] Donato, A , *Preparing Music Manuscript*, Prentice-Hall, Englewood Cliffs, NJ, 1963

[Eri77] Erickson, R , *DARMS A Reference Manual*, (no listed publisher), 1977

[ErW83] Erickson, R and Wolff, A , "The DARMS Project Implementation of an Artificial Language for the Representation of Music", *Trends in Linguistics 19* (1983)

[Fog82] Fogg, D , "Implementation of Domain Abstraction in the Relational Database System INGRES", Masters Report, Department of Electrical Engineering and Computer Science, University of California Berkeley, Berkeley, CA, November 1982

[GoR83] Goldberg, A and Robson, D , *Smalltalk-80 The language and its Implementation*, Addison-Wesley, Reading, MA, 1983

[Gom77] Gomberg, D A., "A Computer-Oriented System for Music Printing", *Computers and the Humanities 11* (1977), 63-80 This article is based on the author's D Sc dissertation from Washington University (1975) of the same title

[Gro84] Gross, D , "Computer Applications to Music Theory A Retrospective", *Computer Music Journal 8*, 4 (Winter 1984), 35-42

[Han84] Hanson, E , "User-Defined Aggregates in the Relational Database System INGRES", Masters Report, Computer Science Division, University of California Berkeley, Berkeley, CA, December 1984

[HeS86] Hewlett, W and Selfridge-Field, E , *Directory of Computer Assisted Research in Musicology*, Center for Computer Assisted Research in the Humanities, Menlo Park, CA, June 1986

[Jun83] Junglieb, S , "MIDI Hardware Fundamentals", *Polyphony 8*, 4 (1983), 34-38

[Kar72] Karkoschka, E , *Notation in New Music*, Praeger Publishers, New York, 1972 trans R Koenig

[KaL82] Katz, R and Lehman, T , "Storage Structures for Versions and Alternatives", Computer Sciences Technical Report #479, University of Wisconsin, Madison, July 1982

[Kra79] Krasner, M A , *Digital Encoding of Speech and Audio Signals Based on the Perceptual Requirements of the Auditory System*, Ph D Dissertation, Lincoln Laboratory, Massachusetts Institute of Technology, Lexington, MA, June 1979

[LeG78] Lee, R M and Gerritsen, R , "Extended Semantics for Generalization Hierarchies", *Proceedings of the ACM-SIGMOD International Conference on the Management of Data*, Austin, TX, May 1978, 18-25

[LoA85] Loy, G and Abbott, C , "Programming Languages for Computer Music Synthesis, Performance, and Composition", *ACM Computing Surveys 17*, 2 (June 1985), 235-265

[MaM70] Mathews, M and Moore, F , "GROOVE – A Program to compose, store and edit functions of time", *Communications of the ACM 13*, 12 (December 1970), 715-721

[MaO83] Maxwell, J T and Ornstein, S M , "Mockingbird A Composer's Amanuensis", Technical Report CSL-83-2, Xerox Palo Alto Research Center, Palo Alto, CA, January 1983

[McL86a] McLean, B , *A Database System for Score-Processing Applications in Musical Computing*, Ph.D Dissertation, State University of New York, Binghamton, 1986 In preparation

[McL86b] McLean, B , "The DARMS Cube The Design of a Data Structure for Score Processing Applications", *Symposium on Computers and Music Research*, Oxford, July 1986

[Moo85] Moore, F R , "The Cmusic Sound Synthesis Program", Computer Audio Research Laboratory Technical Report, University of California, San Diego, La Jolla, CA, 1985

[Ong82] Ong, J , "The Design and Implementation of Abstract Data Types in the Relational Database System INGRES", Masters Report, Department of Electrical Engineering and Computer

Science, University of California Berkeley, Berkeley, CA, September 1982

[OpS75] Oppenheim, A and Schafer, R , *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1975

[Pru84a] Prusinkiewicz, P , "Time Management in Interactive Score Editing", *Proceedings of the International Computer Music Conference*, Paris, 1984, 275-280

[Pru84b] Prusinkiewicz, P , "INTERSCORE – An interactive score editor for microcomputers", *Proceedings of the Fourth Symposium on Small Computers in the Arts*, Philadelphia, PA, 1984, 58-64

[Rea69] Read, G , *Music Notation*, Allyn and Bacon, Boston, 1969

[Rel84] Relational Technology Incorporated, *INGRES Reference Manual, Version 2 1*, Relational Technology Incorporated, Alameda, CA, July 1984

[Roa79] Roads, C , "Grammars as Representations for Music", *Computer Music Journal 3*, 1 (March 1979), 48-56

[Ros70] Ross, T , *The Art of Music Engraving and Processing*, Hansen Books, Miami, 1970

[RoL85] Roussopoulos, N and Leifker, D , "Direct Spatial Search on Pictorial Databases Using Packed R-trees", *Proceedings of the ACM-SIGMOD International Conference on the Management of Data 14*, 4 (December 1985), 17-31

[Rub87] Rubenstein, W , *Data Management of Musical Information*, Ph D Dissertation, University of California Berkeley, Berkeley, CA, 1987 In preparation

[Sho82] Shoshani, A , "Statistical Databases Characteristics, Problems, and Some Solutions", *Proceedings of the International Conference on Very Large Data Bases*, Mexico City , 1982, 208-222

[SOW84] Shoshani, A , Olken, F and Wong, H , "Characteristics of Scientific Databases", *Proceedings of the International Conference on Very Large Data Bases*, 1984, 147-160

[ShK86] Shoshani, A and Kawagoe, K , "Temporal Data Management", Technical Report LBL-21143, Lawrence Berkeley Laboratory, Berkeley, CA, February 1986

[Smi72] Smith, L , "SCORE – A Musician's Approach to Computer Music", *Journal of the Audio Engineering Society 20*, 1 (January 1972), 7-14

[Smi73] Smith, L , "Editing and Printing Music by Computer", *Journal of Music Theory 17*, 2 (1973), 292-308

[SmS77] Smith, J and Smith, D , "Data Base Abstractions Aggregation and Generalization", *ACM Transactions on Database Systems 2* (June 1977), 105-133 Also available as a Technical Report from the University of Utah Department of Computer Science

[SRG83] Stonebraker, M , Rubenstein, W B and Guttman, A , "Application of Abstract Data Types and Abstract Indices to CAD Databases", *Proceedings of the Engineering Design Applications of ACM-IEEE Data Base Week*, San Jose, CA, May 1983 Also available as Electronic Research Laboratory Memorandum M83/3 from University of California Berkeley

[TsZ84] Tsur, S and Zaniolo, C , "An Implementation of GEM – supporting a semantic data model on a relational back-end", *Proceedings of the ACM-SIGMOD International Conference on the Management of Data*, Boston, MA, June 1984, 286-295

[Wen77] Wenker, J , *An Analytical Study of Anglo-Canadian Folksong*, Ph D Dissertation, Indiana University, 1977

[Wil85] Wilson, T A , "Data Reduction of Musical Signals", *Proceedings of the International Computer Music Conference*, Burnaby, British Columbia, 1985, 25-32

[Wol77] Wolff, A B , "Problems of Representation in Musical Computing", *Computers and the Humanities 11* (1977), 3-12

[Zan83] Zaniolo, C , "The Database Language GEM", *Proceedings of the ACM-SIGMOD International Conference on the Management of Data*, Ann Arbor, MI, May 1983