An Object-Relational Model for Musical Data

Research Report

Eva Toller, Tore Risch
Department of Information Science, Uppsala University, S-751 20 Uppsala, Sweden

Abstract[1]

This paper presents an extended entity-relationship model for structured musical databases. Both high-level items like whole pieces of music, down to atomic musical events like single notes, are modelled. The resulting schema has been implemented in AMOS II, an object-relational multidatabase system, with seven short pieces of music as examples. The model supports analysis and search, rather than sound applications or notational applications. There is an emphasis on harmonics (chords), without excluding monophonic (melodic) and rhythmic analysis. The user aspect is focused more on musical amateurs and performers than on musicologists or people who are buying music on the Web.

Keywords: musical database, musical search, musical analysis, object-relational

1 Introduction

With the introduction of the World Wide Web, the distribution and marketing of music have gained a tremendous potential. The record companies and other distributors of music can offer the customers a wider choice than before and an easier way to search for music to purchase. The distribution of music is also easier, for example with mp3-files. Not only audio files but also *musical scores* (i.e. printed music) can be electronically distributed to professional and amateur performers of music around the world.

With the large amount of music that is offered in this way, the need for efficient search methods also increases. Many music purveyors let the customer search regular databases for “descriptional” musical data like genre, title, artist, record company, year of recording, and so on.

However, there are still not many possibilities offered to do search and analysis among the musical *contents* itself. When the music is included at all, it is often as a *BLOB* (Binary Large OBject); for example, as an mp3-file. An established way to recognize musical pieces is to store the few first *bars* of the music, called an *incipit* or *theme* (used by, for example, *Themefinder*, [12]). This is “better than nothing” but has two serious limitations. Firstly, it does not allow general search and analysis on the whole musical contents. Secondly, its usefulness is mostly limited to classical music. In many genres the *chorus* (or *refrain*) is what is most memorable about a tune, and it is more common than not that the refrain is *not* in the beginning.

The purpose of this report is to define a model where *search* and *analysis* can be made down to the “lowest level”, that is, single tones. Our frame of reference is the standard Western musical notation (score) rather than an audio representation like mp3. Thus, the aim is to be able to represent and analyze all components that can occur in a “standard” musical score, but not on signal processing level.

The main contribution presented here is a *conceptual model* of music that is suited for an object-relational database management system. Observe that it is the conceptual *level* that is presented; the *external* and *internal* views will be different (see section 7 for a discussion of this). For the sake of brevity, the model will sometimes be referred to as *SMORMIA* (short for “structured musical object-relational model in AMOS II”).

Organization

In section 2, some similar projects and systems are referenced. This is related work in a wide meaning since we have been unable to find many other musical database models; to the best of our knowledge, this is the first *object-relational* model. Section 3 describes assumptions and restrictions we have set for this work. In section 4, the object-relation database management system that is used it shortly outlined. Section 5 presents the developed conceptual schema, and section 6 shows some pieces of music that have been implemented using the schema. Section 7 discusses the important differences between the conceptual

[1] This work was partly sponsored by *Framfab*.

level and the internal and external levels, while section 8 gives examples of queries that can be put to a musical database. Finally, conclusions and future work are presented in section 9.

2 Related work

The work we have found that is most closely related to our own is the entity-relationship model for musical data described in [17]. As in our model, the concepts of *recursive hierarchies* and *hierarchical ordering* are important, but in [17] they are also used on a lower level, for groups and subgroups of chords. (See section 5.1 for how the hierarchical concept is used in our model). [17] is furthermore much more concerned with the printing of musical scores.

Apart from entity-relationship models for database systems, there is a large number of codes for musical representation. Many of them are described in [18], where they are divided into three *code categories*: sound applications, notational applications, and analytical applications. Examples of sound applications described there are *MIDI* and *Csound*; examples of notational applications are *DARMS* and *SCORE*.

Our approach belongs to the analytical group but we will also use the well-known MIDI format as a data source. Both “classical” MIDI and several extensions are described in [18].

Some musical codes are part of a system for musical analysis. The *Humdrum Toolkit* [11], [9] contains both a large number of musical formats and operators defined to work on one or several formats. An extensive bibliography of publications related to the Humdrum Toolkit is listed in [10]. The basic format is ***kern* [8].

There are search tools for music on the Web, some with their own musical formats. *MuseData* [6], [14], has a multipurpose representation and also includes a database with several hundreds of classic musical works. *Themefinder* [12] has a couple of thousand incipits (themes) that can be searched (it can be accessed from [14]). A larger music database is the *Essen* database of 10,000 *monophonic* (melodic) transcriptions of folk songs, coded in *EsAC* [18].

More systems and programs are mentioned in [7], and many more codes are described in [18]. However, the Humdrum Toolkit and MuseData are among the few that include representations for *polyphonic* music (i.e., music for several voices or instruments) which is necessary for harmonic analysis (or chord analysis).

There are also markup and interchange languages for music; see for example chapter 30 and 31 in [18], and [15].

Tools for advanced analysis of polyphonic music are not so common as tools and programs for monophonic music. See [22] and [23] for examples of automatic analysis of polyphonic music.

The low-level methods for content-based music search differ. A common approach seems to be methods based on *text-search*, see for example [26]. The *OMRAS* project (Online Music Recognition and Searching, [16]) has defined two different search strategies: one is text-based and the other one is based on *score-matrices* [2], [3].

The work presented in this paper has an emphasis on three aspects: *analysis and search* (as opposed to sound or notation, see [18]), *music performers* (as opposed to musicologists or music buyers) and *harmonic analysis* (as opposed to melodic or rhythmic analysis, see again [18]). These choices will be motivated in section 3. A common factor, however, is that the respective choice has been only moderately explored yet, compared to the others.

3 Restrictions and assumptions

The first and most important restriction is that we do not think that the suggested schema is suitable and/or possible to use for *all* types of musical genres. In concentrating on the musical score, certain kinds of music that have invented other types of representations exactly *because* a score was not suitable, may be impossible to represent. However, we have included some very common representations, like lyrics together with chord names. Incidentally, it is also possible to store *only* the lyrics as long as some type of *meter* (time signature, e.g. 3/4) is assigned to it.

The musical examples in section 6 are all hand-coded. This is of course only feasible for small amounts of data. To obtain the larger data quantities that are necessary for scalability tests, we plan on two strategies. The first is to legally obtain as many MIDI files as possible, and use the Humdrum format **kern as an intermediate format. MIDI files can be converted to **kern files, coded in a comprehensible ASCII format that looks rather similar to a musical score [8]. The second strategy is to use a simple program that generates synthetic pieces of music *en masse*. From an aesthetic point of view, this “music” will not be of good quality, but it will be sufficient for benchmark tests.

Performers, analysis/search, harmonics: motivations

The reasons to concentrate on musical amateurs and performers are the following. Firstly, what efforts that have already been made in this research area are mostly for the benefit of musicologists; for example, to make advanced and intricate analyses. These systems are also mostly handling classical (i.e., mostly old) music. Secondly, music buyers on the Web will require easy input methods (for example, direct input of song), and that is beyond the scope of this work. Note that the “SQLish” syntax used for queries in section 8 may be incomprehensible to most musicians, but at least the musical terms should be clear. However, we hope that more groups than musical performers will benefit. For example, we got a spontaneous remark from a dancer that this could be used to find good music when learning to dance. Important aspects here are rhythm, tempo, breaks, and the *form schema* (the major building blocks for a piece of music, see figure 4). Yet another application is copyright infringement [1].

One reason to concentrate on analysis and search is that there is a natural connection to musical performers, who are often searching for suitable music to perform or arrange. Furthermore, we regard this as the most interesting area from a research perspective. This does not necessarily exclude other applications totally. For example, some features for score layout are very easy to add (like *stem directions* and *clefs*). But no explicit efforts will be made to include such features.

The reason for concentrating on harmonics is that it has been less investigated than other aspects, but also because it introduces *n-dimensionality*; a musical piece with several *parts* (voices or instruments) can be regarded as having a *horizontal* aspect (time) and a *vertical* aspect (the different parts). This may be exploited in unforeseen ways, for example with *image processing* (see suggestion in section 7).

There are several requirements on the database system that will be used for the musical database. For example, it must be able to store and query *vectors* (sequences) in an efficient way. Vectors are central since they (as opposed to *bags*) can store data in a defined order, which is extremely important for all types of music. If an image processing approach is used, the database system must also be able to store vectors of at least two dimensions.

4 The object-relational database: AMOS II

AMOS II (Active Mediators Object System [4]) is a data integration system based on the *wrapper-mediator* approach [25]. AMOS II consists of a mediator database engine that can process and execute queries over data stored locally and in several external data sources, and object-oriented multidatabase views for reconciliation of data and schema heterogeneities among data sources.

The data stored in different types of data sources are translated and integrated using object-oriented mediation primitives, providing the user with a consistent view of data in all the sources. Through its multidatabase facilities, many distributed AMOS II systems can inter-operate in a federation. Since most data reside in the data sources, and to achieve high performance, the core of the system is a main-memory database management system. AMOS II is extensible so that new data types and query operators can be added in some external programming language.

The query language of AMOS II, *AMOSQL* [24], is based on the functional data model DAPLEX [20] and OSQL [13].

AMOS II is well suited to store musical databases because it can handle both distributed and heterogeneous data. The former is important because wide-world search should be a major application. The latter is important because of the large number of different musical representations that can act as data sources.

For an overview of object-relational databases in general, see [21].

5 Conceptual schema

In this section, the most important features of the extended entity-relationship model (SMORMIA) are described. As far as possible, the standard symbols for EER-models are used (see figure 1). To describe all features in detail is beyond the scope of this report. Except or the central concept called *Piece*, other concepts will mostly be described with examples when this is necessary for the context. However, all types and relations in the schema are shown in the figures.

Except for a few derived functions for harmonic relations, the whole schema has been implemented in AMOS II.

5.1 Pieces and Parts

A Piece (see figure 2) is any self-contained musical piece, and can itself contain a number of sub-Pieces (called *sections*) represented in a vector. For example, a song (top-level Piece) can contain a vector [Verse1, Verse2, Chorus, Verse3, Chorus, Verse4] where each element (= section) is itself a Piece. Each section can itself contain its own nested sections (figure 3 illustrates this).

The Piece concept takes care of the “horizontal” aspect of music (i.e., progress in time). To take care of the “vertical” aspect (several instruments or voices sounding at the same time stamp) there is a corresponding concept *Part*. Each Part represents not primarily a single voice or instrument, but rather a single *melody line*. Thus, some instruments (like Piano) may have to be represented by several parallel Parts. This is simple to achieve because the Part is constructed exactly as the Piece in that it may itself contain vectors of Parts to an unspecified depth. See for example “Viola” in the last section of the Mozart Trio (figure 9 in section 6.1) or the Mozart Piano Sonata’s right hand (figure 10 in section 6.2).

Each *leaf-node* level Piece has a vector of Parts associated with it (called *allParts*). The Part vector that can be included in another Part is called *voices*. Finally, each *leaf-node* level Part has the actual musical material associated with it, as a vector of tones and rests[1] called *tones*. In figure 2, the most important relationships of Piece

[1] Other types of entities are also allowed here; see section 5.3.

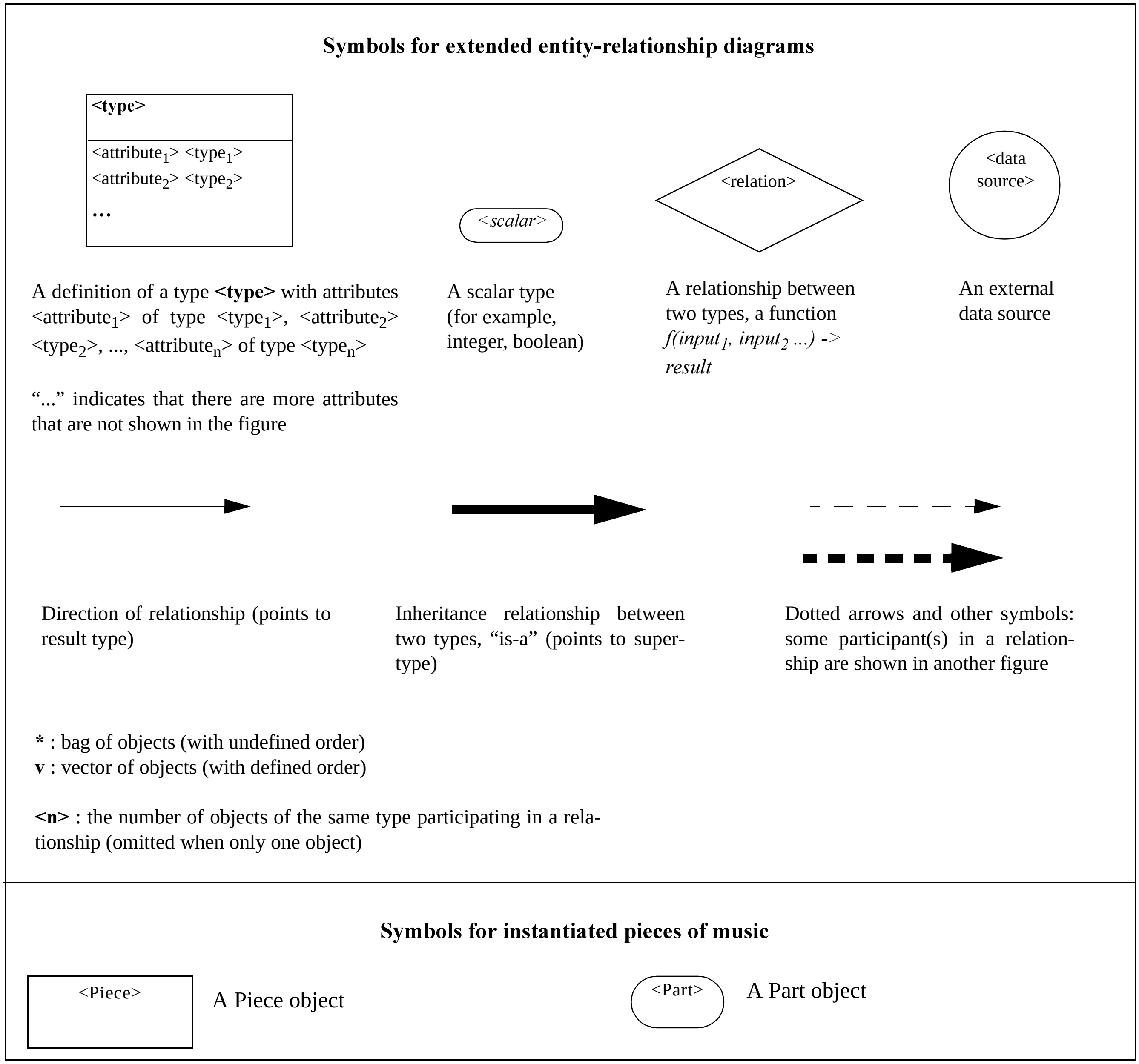
and Part are shown in an entity-relationship diagram, and a musical example is given in figure 3. Note that some properties and relationships are duplicated in Piece and Part. For example, it is possible but not mandatory to assign meter and key signatures to individual Parts as well as to Pieces.

5.2 High-level musical contents of a Piece

Some high-level concepts in SMORMIA are shown in figure 4 (see also figure 2).

Note that a *formSchema* can be attached to a Piece if desired. The formSchema does not have to correspond exactly to the structure of Pieces, although this will often be the case for simple music. An example of a case when the correspondence is not one-to-one is shown in figure 11.

In *Genre*, features specific to certain musical genres can be stored (but see also section 5.7). Properties regarded common to “all” music is stored in *Music* but can be overrided by Genre.


Symbols for extended entity-relationship diagrams
<type>
<attribute$_1$> <type$_1$>
<attribute$_2$> <type$_2$>
...
<scalar>
<relation>
<data source>
Symbols for instantiated pieces of music
<Piece>
<Part>

A definition of a type **<type>** with attributes <attribute$_1$> of type <type$_1$>, <attribute$_2$> <type$_2$>, ..., <attribute$_n$> of type <type$_n$>

“...” indicates that there are more attributes that are not shown in the figure

A scalar type (for example, integer, boolean)

A relationship between two types, a function *f(input$_1$, input$_2$ ...) -> result*

An external data source

Direction of relationship (points to result type)

Inheritance relationship between two types, “is-a” (points to supertype)

Dotted arrows and other symbols: some participant(s) in a relationship are shown in another figure

***** : bag of objects (with undefined order)
v : vector of objects (with defined order)

<n> : the number of objects of the same type participating in a relationship (omitted when only one object)

A Piece object

A Part object

Figure 1: Symbols used in EER diagrams and implementations

5.3 Metric and duration information for a Piece

Types and relations for metric information are shown in figure 5. An important concept is the *DurationUnit*, whose main feature is that it has a duration in time[1]. Both tones and *rests* are subclasses of DurationUnit. However, it is possible to use the DurationUnit directly for instantiation of objects. For example, when you want only *lyrics* but no tones (as in rap and text declamations) this is the natural choice. Each DurationUnit then contains one text syllable[2].

[1] In some cases, the duration is allowed to be zero; see section 6.2.

[2] Sometimes syllables have to be "divided" among several DurationUnits, but this is a relatively trivial problem (see section 6.4).

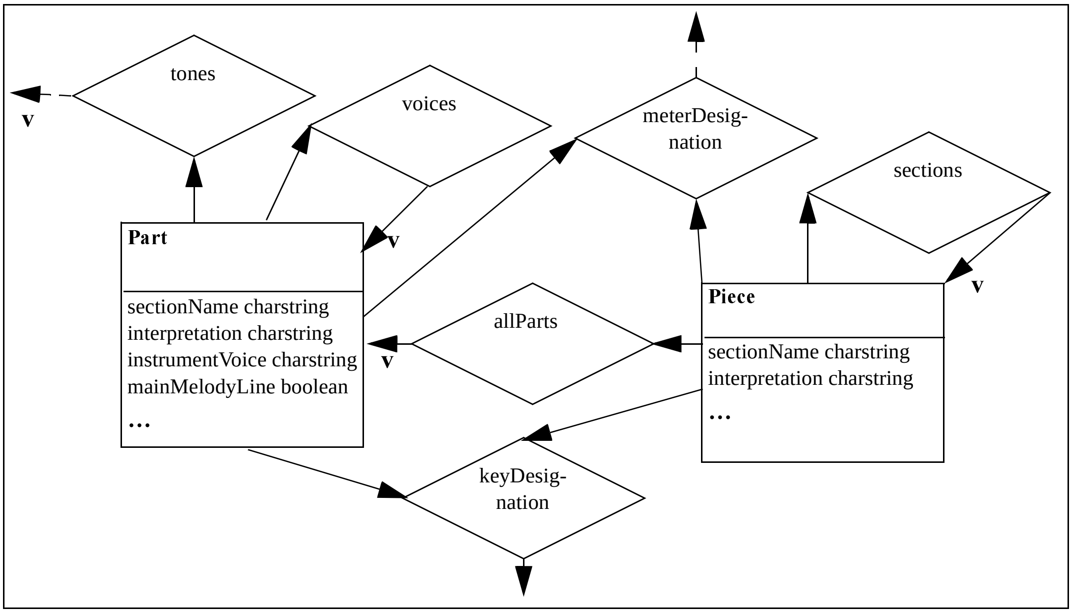


Figure 2: Piece, Part and their relation to each other

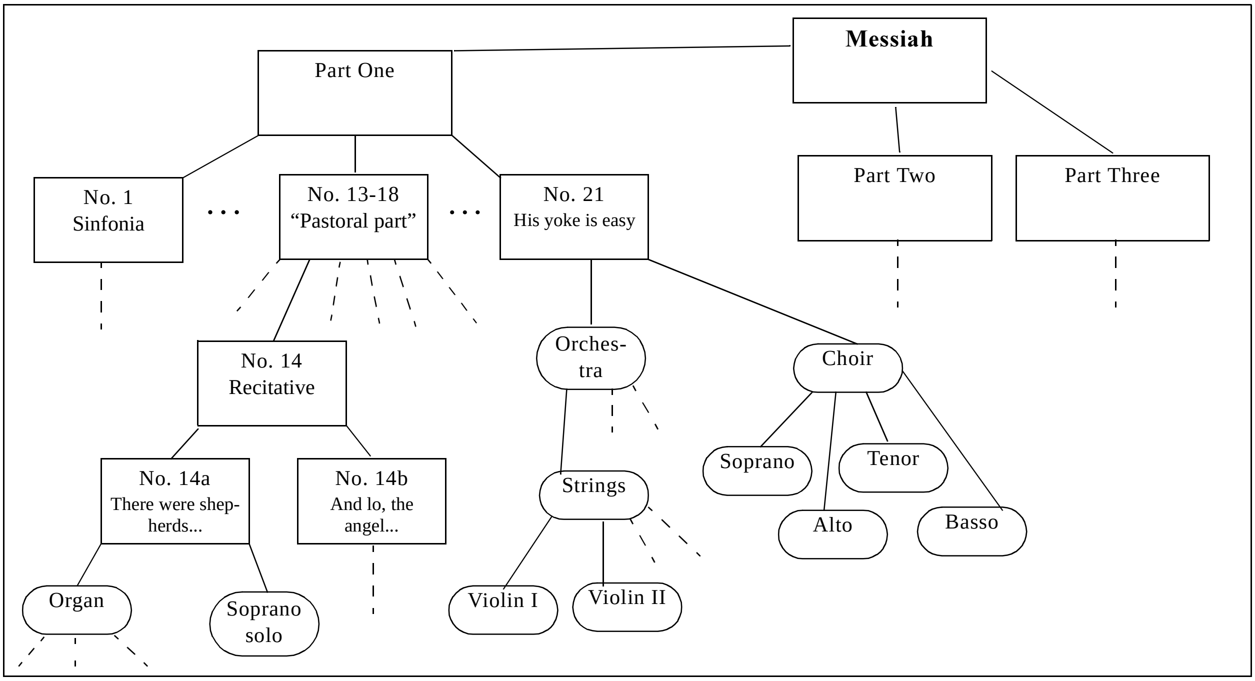


Figure 3: Examples of Pieces and Parts from Handel's Messiah

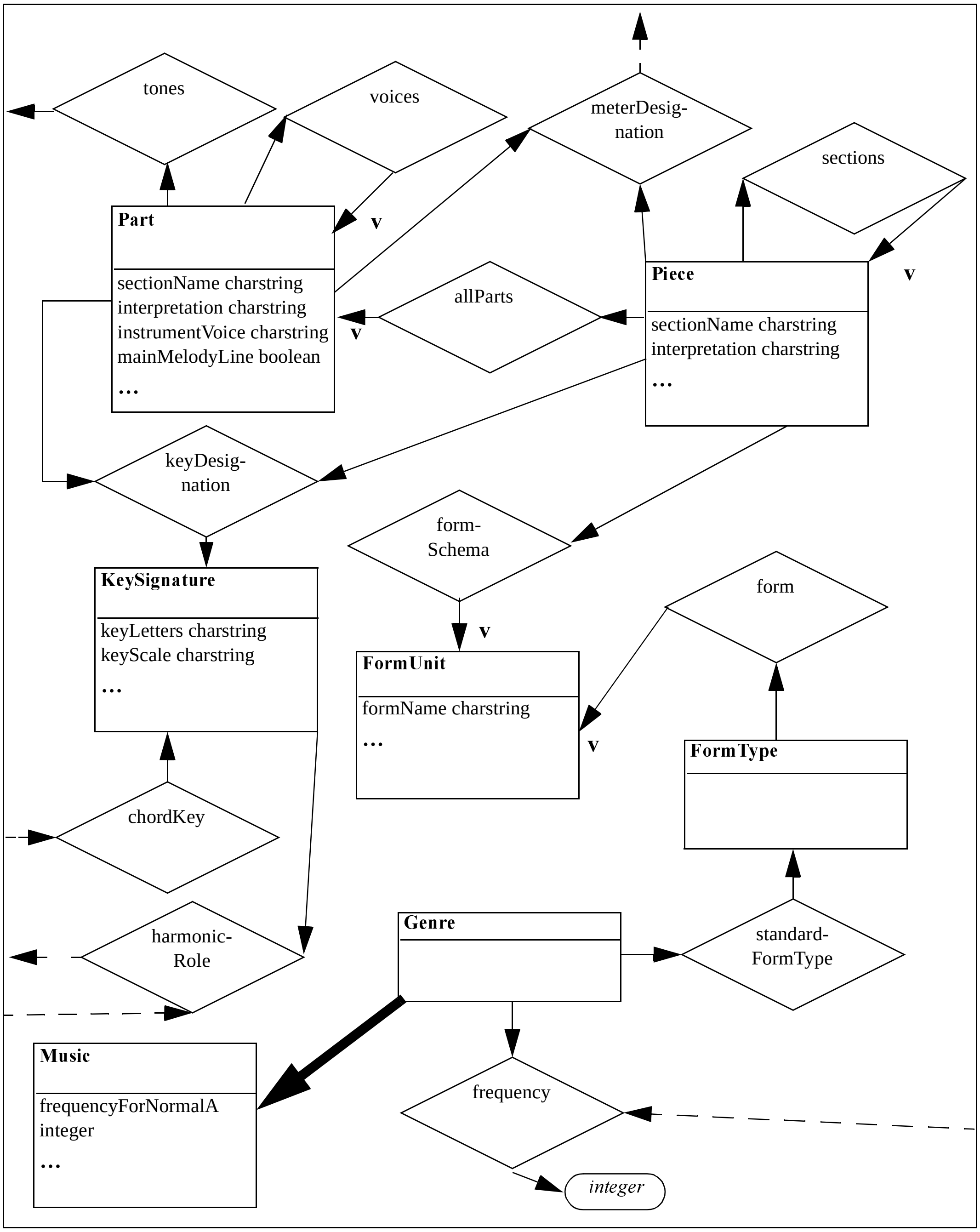


Figure 4: High-level musical contents of a Piece

AbsoluteDuration is the "real" note value (e.g., a quarter note). *RelativeDuration* is introduced so that the user does not have to know the exact meter designation for a Piece (e.g., 3/4 versus 3/8). It can be automatically calculated.

5.4 Tonal information for a Piece

Types and relations for tonal information are shown in figure 6. Observe that it is a *single melody line* that is described here.

AbsoluteTone is the "real" pitch, and *SolfaTone* (the relative position of a tone in a scale) is introduced for key-independence. Like RelativeDuration, SolfaTone can be automatically calculated.

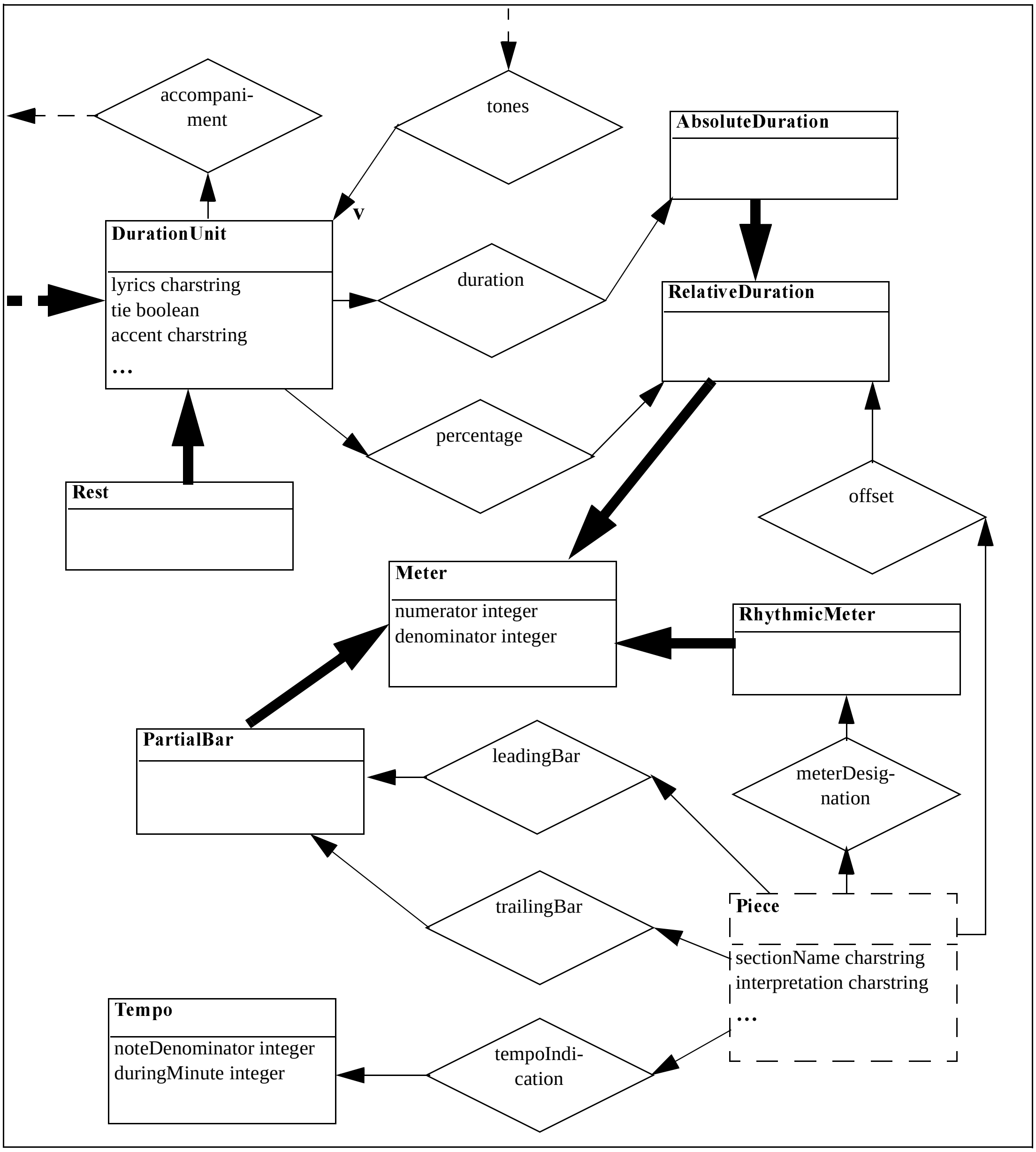


Figure 5: Metric and duration information for a Piece

5.5 Harmonic information for a Piece

Harmonic information is shown in figure 7. Some of this information is predefined, according to classical Western harmonic theory. For example, *TheoreticChord* instantiations are all types of "named" chords: C, Cm, C+, C6, C7, Cdim... but are stored as key-independent. *DefinedChord* are all variants that is allowed of a specific TheoreticChord (for example, a C7 either without the tone C or the tone G). *AbsoluteChord* is the "real" chord, and here *duplicates* of named tones may be included (for example, a Cm chord with two G tones).

To allow other combinations than the traditional chords, *ToneCombination* can contain any possible combination of tones. This may or may not be an AbsoluteChord. Note that an AbsoluteChord can be automatically derived from one time stamp of several parallel Parts.

The user is free to add his/her own TheoreticChords and DefinedChords. For example, if there is a common tone combination C-D-E-F-G that is used, this can be defined as a TheoreticChord named (for example) "DoReMiFaSo".

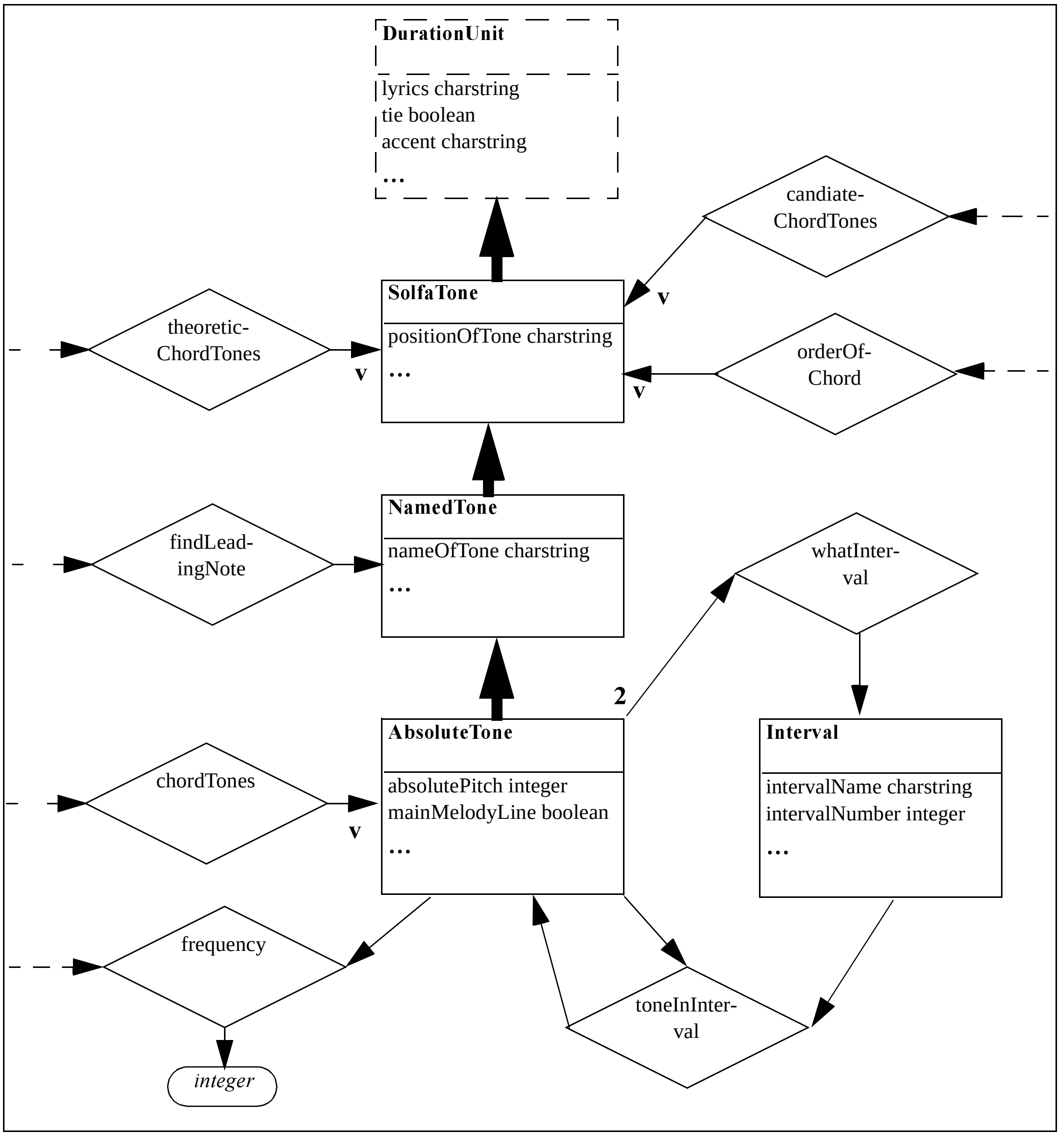


Figure 6: Tonal information for a Piece

accompaniment is chord designations that can be used for any DurationUnit: tones, rests, or lyrics. Thus, it is for example possible to store the very common combination of lyrics + chords, or lyrics + melody + chords.

5.6 Descriptive, audio, and source data for a Piece

Lastly, high-level descriptions and other information is shown in figure 8. This includes composer data, different versions of a Piece, indication of copyright for composers, performed versions of a Piece, data sources for the structured musical contents (for example MIDI files) and audio data (for example, mp3-files).

It is perfectly possible to store *OriginalPieces* that have never been performed, and thus contain no audio data or performed versions (*PerformedPiece*). Likewise, it is possible to store pieces that have no structured musical contents at all, as in the Web sites that let you listen to or load down mp3-files.

Source data (*PieceDataSource*) deserves some more explanation. To obtain large quantities of data for volume tests, MIDI files with **kern as an intermediate format can be used (as mentioned in section 3). Note that several data sources may be required for one Piece. For example, **kern itself does not include representation of lyrics, but another Humdrum format, ***text*, does (see also section 6.4).

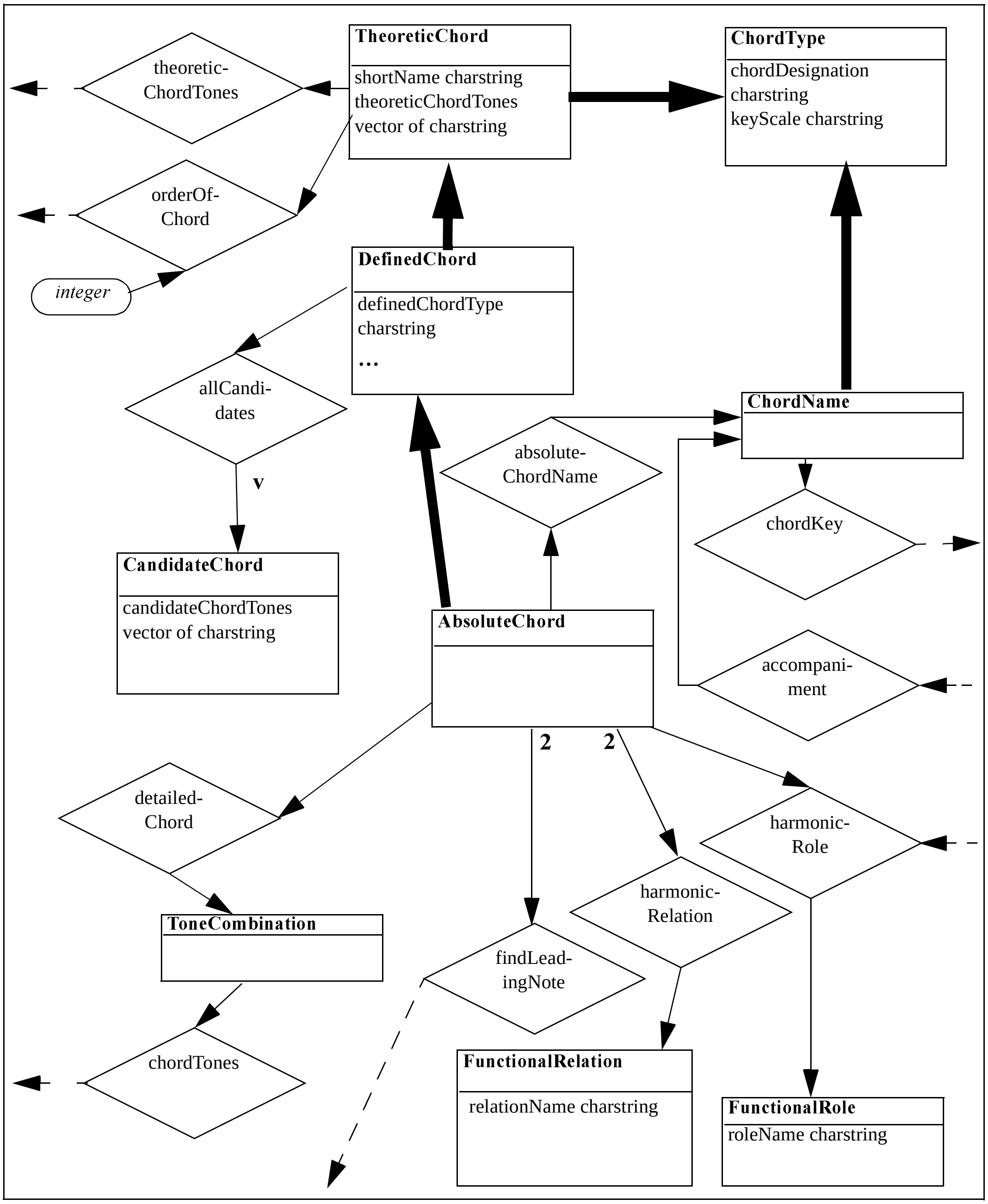


Figure 7: Harmonic information for a Piece

5.7 Genre-specific data

It is not possible to code all types of music without genre-specific information; thus, the type *Genre* mentioned in section 5.2. However, this is probably not enough; you may have to create genre-specific subtypes of some too-general entities, for example *SongPiece* as a subtype of Piece, *SongPart* as a subtype of Part, or *SongDurationUnit* as a subtype of DurationUnit. But in the implementation described in section 6, no such subtypes have been used.

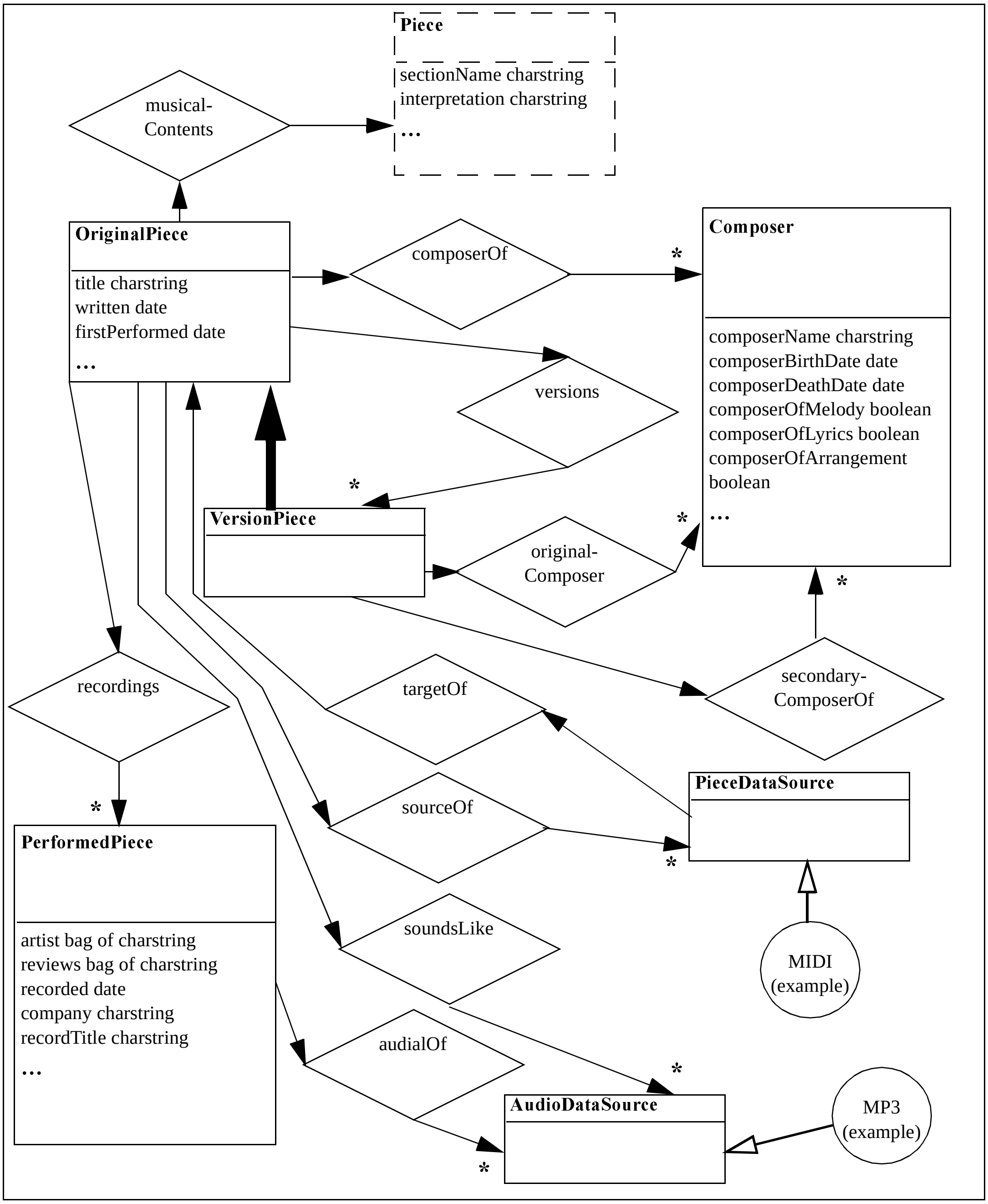


Figure 8: Descriptive, audio, and source data for a Piece

6 Implemented pieces of music

The entity-relationship model shown in section 5 (SMORMIA) has been used to implement seven small pieces of music, to exemplify how the model handles some difficulties that are common to music representations.

The first six pieces are taken from [19]. In [18], they are used as a.k.a. benchmark for the different types of musical codes that are described there. Each code is challenged to implement the example pieces of music as completely as possible, with respect to the code category of each representation (sound, musical notation, or analysis).

The seventh piece is an arrangement of the well-known “Amazing Grace”; this has been included to cover some aspects not inherent in the pieces from [19].

In section 6.1 - section 6.7, it is shortly described how each musical piece has been treated. For a full description of the first six pieces with scores, see [19].

Types and relations not used in the implementation

Not all model elements described in section 5 have been used for implementation. Foremost, no attempt has been made to include the descriptive and audio information from section 5.6. As for source data, all the seven pieces have been hand-coded. This is of course only possible with small data quantities.

The formSchema and its related entities have only been used very superficially, since the coded pieces are too short to need them.

Generally, information that can be derived - like most of the harmonics - has not been coded explicitly.

6.1 Mozart Trio

This example is taken from the second trio section of Mozart's Clarinet Quintet K 581 in A major (for Clarinet, Violin I, Violin II, Viola, and Violincello). It is 16 bars long. In [18], the respondents were only asked to set the first 12 bars (up to a repeat sign).

The piece is rather simple. The main difficulties in the first 12 bars is that the Clarinet part is notated in C major, while the piece itself is in A major. There are also the following interesting features that must be handled:

1. There is a 3-tuplet of quarter notes in the 8th bar.
2. There are two repetitions of the first part.
3. There are double notes in the Violin II and Viola parts (this is in the section that was not required to be coded).

SMORMIA can represent all the features important to analyses, including the last four bars. Features connected to layout, like directions of stems, was not included. However, it is perfectly possibly to add such information as a property of DurationUnit if desired.

Except for tones and rests, *ties* were included (as properties of AbsoluteTone; tie = true means a tie (at least) to the next note). A property *accent* of AbsoluteTone was used to code things like staccatos. The dynamics (here only piano, *p*) was instead stored on Part level since it was the same for all notes.

The difficulty with Clarinet in C major was solved by explicitly setting *keyDesignation* to C major for that specific Part (while the whole Piece has A major as keyDesignation). However, a property *originalKey* was introduced for Part, and set to C major for the Clarinet part. If it is later desirable to *transpose* the Clarinet part to A major (using a function named *transpose*), the original key can be restored later.

Observe that items like originalKey, which are only applicable for a few instances, can be added as an afterthought. They only need to be instantiated for these specific instances and will be undefined for the rest. This is also true for similar entities defined in subsequent sections. For convenience, they will henceforth be called *exceptional properties*.

The 3-tuplet was handled in the following way. First the absolute values (AbsoluteDuration) of the tuplet was coded as quarter notes, as in the score. However, RelativeDuration was set to the "true" duration, for each note, 1/9. Furthermore, an exceptional property *nTuplet* was added to AbsoluteTone, and set to 3 for the three quarter notes in the tuplet. This is not strictly necessary since the same information can be derived from RelativeDuration, but it is a quicker way to find n-tuplets.

The two repetitions in the first part are exactly identical. Thus the *tones* vector for the Piece representing the first repetition was first coded, and was then copied-by-reference to the tone vector of the second repetition. This saves some space, but more importantly it ensures that the two tone segments are identical. Note that this is only feasible when there are *exact* matches (as opposed to the Saltarello, see section 6.3).

Incidentally, bar numbers are not coded explicitly in SMORMIA, but are calculated by an *offset* present in each Piece, together with the meter designation. Thus, the tone vectors themselves are independent of their relative placement in a top-level Piece.

The double notes in the Violin II and Viola parts were handled by introducing several Parts for these instruments. Two variants were coded: in the first, all new Parts (Violin II A, Violin II B, Viola A, and Viola B) were put at the same level as the other instruments in a "flat" way. In the second variant, a meta-level was introduced for these instruments.In figure 9, the Pieces and Parts for this second variant are shown.

The tone vectors for Violin II B and Viola B are as long as the vectors for Violin II A and Viola A, and have rests where there are only single notes. Observe that for external presentation, the tone vectors can easily be "flattened" to obtain a more natural look.

6.2 Mozart Piano Sonata

This example is an excerpt from the Mozart Piano Sonata Clarinet Quintet K 331 in A major; it is only five bars long.

The main difficulty is a lot of *grace notes* that have no "formal" duration but are noted as 1/32. Another problem is an arpeggio in the first bar (right hand), consisting of four parallel notes.

A problem inherent to all keyboard pieces is that the number of parallel "vertical" notes is irregular. However, this was solved in the same way as for Violin and Viola in the Mozart Trio's last section (see section 6.1).

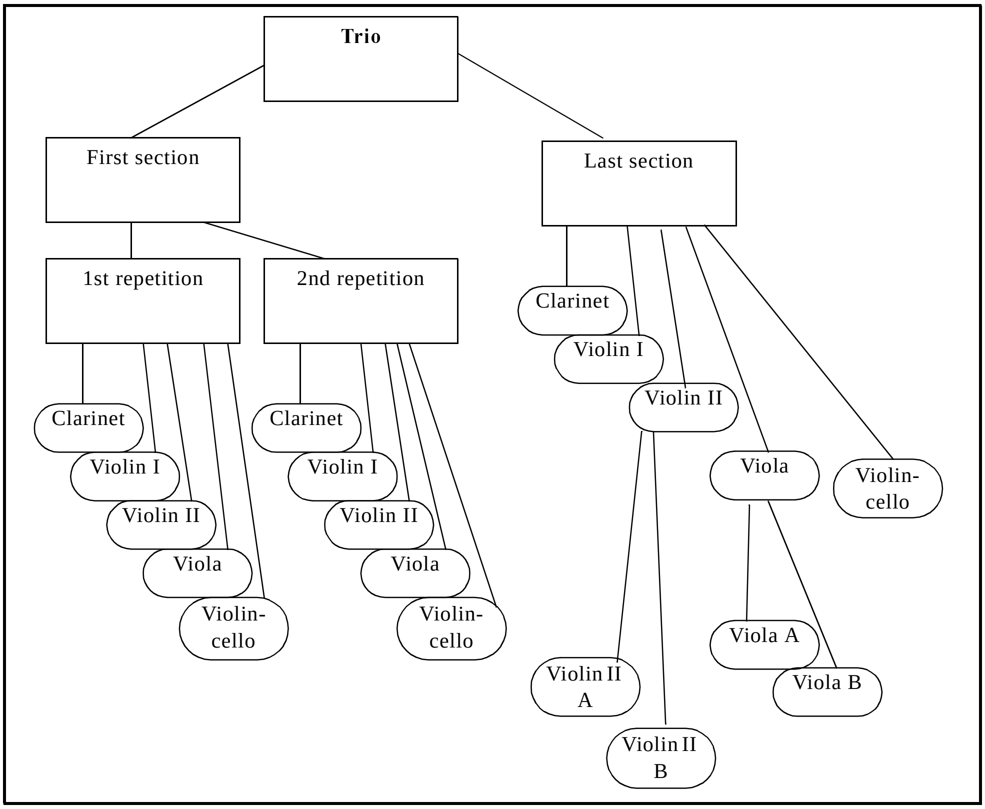


Figure 9: Pieces and Parts of Mozart Trio

Alternative solutions were given to the problem of grace notes. Common to the solutions were that the grace notes were given AbsoluteDuration = zero (0). In the first solution nothing more was done, and the RelativeDurations were also set to zero. In the second solution, the RelativeDurations were divided between all the notes in one bar. However, no attempt was made to make an *exact* match, since the notation of 1/32 for grace notes should not be taken literally, and the result would be a metrics rather hard to read and comprehend. Instead, natural approximations were used. For example, in a bar with three grace notes of 1/32 duration each and four regular notes of 1/8 duration each, the RelativeMeter was set to 1/15 for each grace note, and to 1/5 for each 1/8 note.

Preferable, approximations like these should be made automatically, but this would require a rather intelligent routine.

The property *accent* was set to “grace note” for the concerned notes (not strictly necessary since the duration of length zero provides enough information).

The arpeggio was also coded in two alternate ways. In the first and most simple approach, the real-time durations were disregarded; instead, they were coded as presented in the score. Since the arpeggio consists mainly of parallel quarter notes, rests were added for consistency. The property *accent* was set to “arpeggio” for the concerned notes.

In the more complicated solution, the arpeggio notes were converted to a “real” arpeggio in the following way: the tones were spread out regularly throughout the bar, again not as an exact match. There were three quarter notes and one half note. The half note got duration 7/16, while each quarter note got duration 3/16. The relative duration was set to 1/5 for each quarter note and 2/5 for the half note.

Exact representations of these features were not attempted by any of the contributors in [18].

Since the right-hand chords consist of at most four parallel notes, four Parts were used to represent them: see figure 10. The left hand consists of a single melody line.

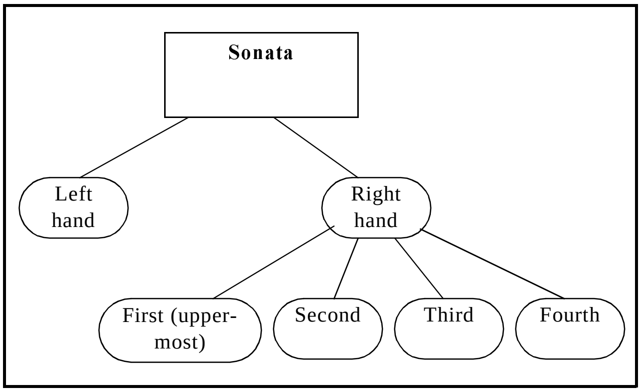


Figure 10: Piece and Parts of Mozart Piano Sonata

6.3 Saltarello

This is an anonymous piece from the early Renaissance. It consists of a single melody line divided into two sections, each of which is repeated. The only difficulty with the piece is that the repetitions have different endings. This is shown in figure 11. Observe that the first ending in the first repetition for both sections are exactly the same, and the same holds for the second ending. The multiple endings make the Saltarello quite complicated at a high level, even if it is otherwise very simple. However, the identical parts can be copied-by-reference, just as was done for the Mozart Trio in section 6.1.

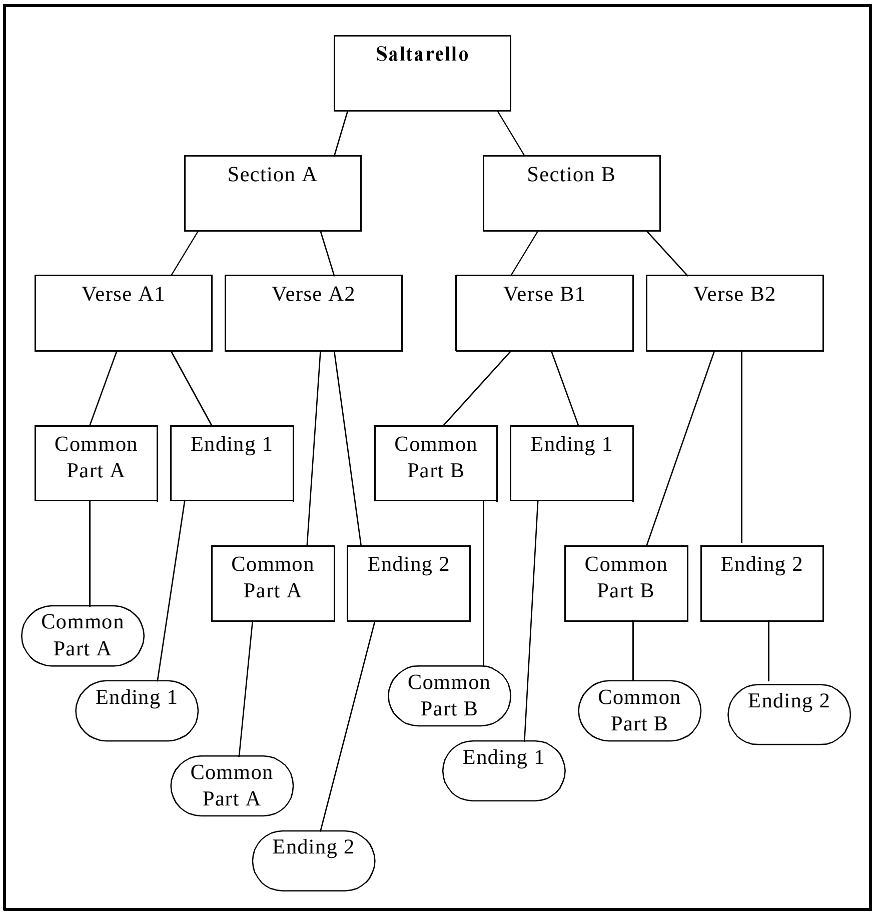


Figure 11: Pieces and Parts of the Saltarello

6.4 Telemann Aria

This is the first 35 bars of the Telemann Aria “Liebe! Liebe! Was ist schöner als die Liebe?” (D major, 3/8 meter). The score is for Voice, Oboe, Violin, Viola, Violincello, and Cembalo.

Like in the Mozart Trio and Piano Sonata, there are irregular numbers of parallel “vertical” notes (here only for Viola) but this is solved as before; three separate Viola parts are required. See figure 12.

The Aria is represented in two Pieces: a wholly instrumental section (the first eleven bars) and a section with Voice and instruments. It could as well has been represented as one single section, as it is in the score; then, the first eleven bars for the Voice part would have been represented with rests, as is also done in the score. Praxis for this varies widely in printed music. Sometimes it is very impractical to represent all parts in parallel from the beginning (for example, when the piece starts with one single voice or instrument, and suddenly “grows” to 12 - 16 instruments).

Lyrics is introduced here; as mentioned before, lyrics is represented with a syllable for each duration unit, but for an aria like this, one syllable often spans over several DurationUnits (i.e., notes). This is indicated by placing ‘-’ before and/or after the text, for example “schö-”, “-ö-”,
“-ö-”, “-ner”.

In many cases, such text fragments can be collapsed to a regular text. However, the text can also be directly represented as a separate *vector of charstrings* in a Part. This may be enough when the correspondence between notes (DurationUnits) and syllables are one-to-one. In other cases, the lyrics must be coded as above, in order not to lose information.

A difficulty mentioned in [19] is the *diacritics* (for German letters, like ‘ö’ and ‘ü’). However, this has more to do with text representation in general than with music in particular.

This piece is the most extensive of the hitherto presented examples, but presents no new principal difficulties for SMORMIA.

Incidentally, the *main melody line* of a Piece can be represented either in a Part or in a DurationUnit, as a boolean value. Mostly the main melody line is held by the same Part, but in the first wholly instrumental section in the Telemann Aria, it wanders between Oboe and Violin. In this case, it makes sense to represent the main melody line in individual DurationUnits. In the second section (with both Voice and instruments) the main melody line is held by the Voice “almost” all the time, except for a few bars when there is a long rest in the Voice part; the main melody line then goes back to the Oboe. If it is not considered very important to have a long unbroken main melody line, it should be alright to “cheat” and only represent the main melody line in the Part for the Voice. Otherwise, it must be represented in DurationUnits all through the current Piece. The latter approach was chosen in the SMORMIA implementation.

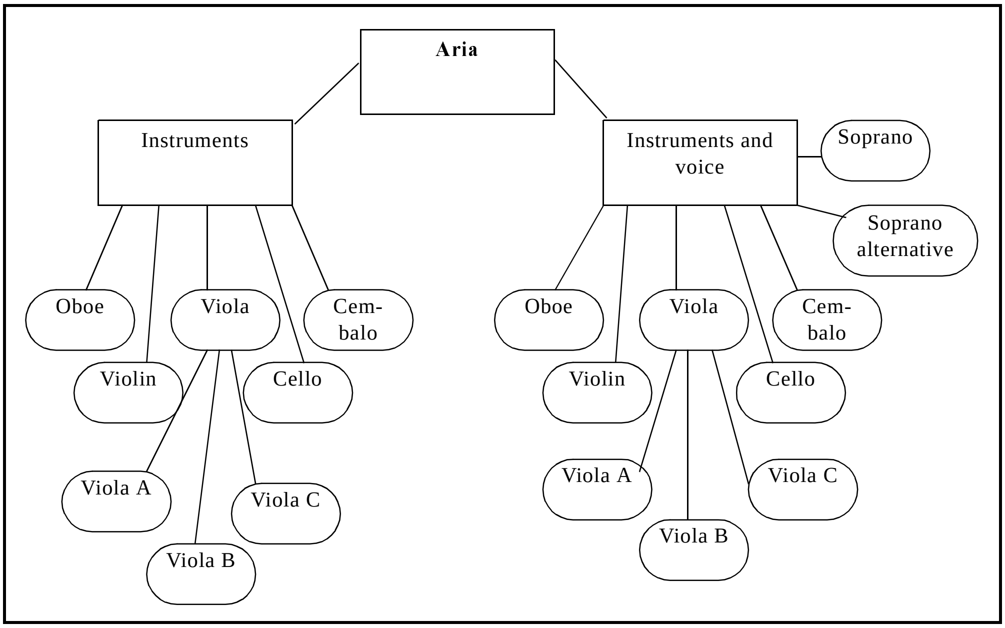


Figure 12: Pieces and Parts of the Telemann Aria

There is a *drill* in the tenth bar; the corresponding AbsoluteTone's *accent* is set to "drill". There are a lot of *cue notes* in the Aria; they are marked as "cue note" in the *accent* property. If a note had been both a drill and a grace note, an exceptional property would have been defined for the drill.

Lastly, there is an alternative variation for the Voice in the 30th bar; it has been stored in a specific Part ("Soprano alternative"). This is an example when it is clearly inconvenient to allow only one single melody line for each Part. Likewise, the third Viola part contains only a few notes.

6.5 Unmeasured Chant

Two unmeasured chants are given as examples. The first, "Alma Redemptoris Mater", is written in *neumes*. The second, "Quem queritis", is written in modern notation.

Common to both chants is that no metric information has been given (which is the essence of "unmeasured"). In "Quem queritis", all notes have the same duration (1/4).

It is not possible to convey any *exact* notation to unmeasured music. However, it is very practical to have some kind of metric notation, even if it is somewhat artificial.

Four different implementations were made for "Quem queritis":

1. Use of a *global default meter* (here set to 4/4) that is automatically inserted when no meter designation is supplied.
2. A meter *n/4* is automatically calculated so that *mod(numberOfTones, n)* = 0 (here resulting in 3/4 meter).
3. The whole chant is considered as *one* single bar and the meter designation is set according to that (here 27/4 meter).
4. The meter is hand-coded according to suitable passages in the text. For example, word limits, syllable limits, and breath marks can be used, here resulting in six bars with the following meters: 2/4, 7/4, 6/4, 4/4, 2/4, and 6/4.

With the fourth choice, it is necessary to divide the piece into six sub-pieces. This is shown in figure 13.

An exceptional property is also created to store the single *breath mark* in the piece, and the property *interpretation* in the top-level Piece is set to "Unmeasured", to indicated that the durations should not be taken too literally.

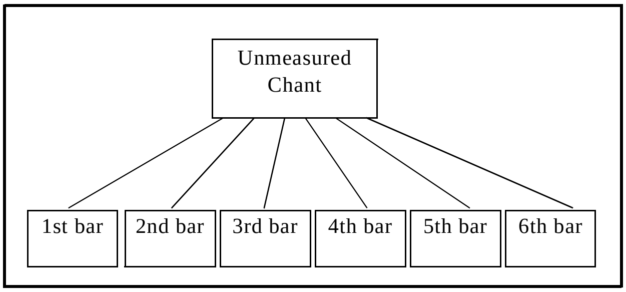


Figure 13: Pieces of the Unmeasured Chant

6.6 Gilles Binchois' Magnificat

This is an excerpt from a Renaissance piece for three voices. The first section is an unmeasured incipit for a single voice. The next part (with 3/4 meter) is for three voices, *Cantus*, *Cantus 2*, and *Tenor*. The last part also contain these three voices, but here Cantus 2 and Tenor are specified as "instrumental".

The difficulties in this piece is mostly in the editorial makeup and interpretation (i.e., notation) so they present no new problems for SMORMIA.

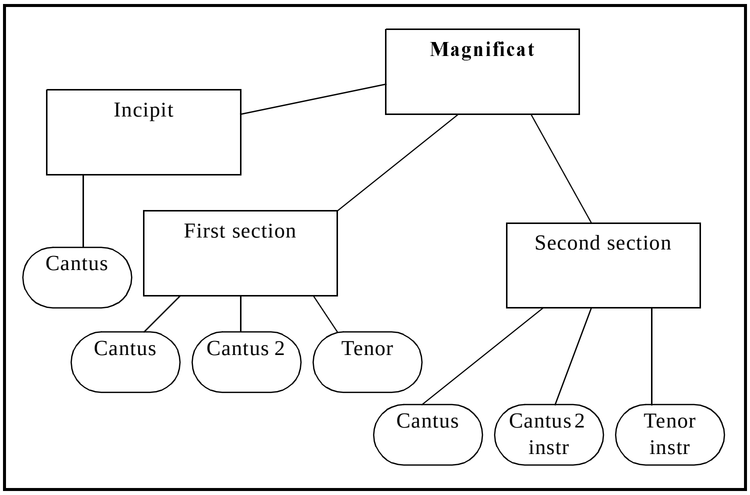


Figure 14: Pieces and Parts of Binchois' Magnificat

There are some *brackets*, which are represented in a similar way as *ties*, and there is one *fermata*, which is represented as an exceptional property.

The Pieces and Parts are shown in figure 14.

6.7 Amazing Grace with accompaniment

This is the well-known tune “Amazing Grace”. It has four verses. In this specific arrangement, the first verse is a single solo voice (can be Soprano or Contralto) with no accompaniment (i.e., sung *a capella*). The second verse is a solo voice and three more voices (Alto, Tenor, Basso) that function as accompaniment to the solo voice (who is the only one that is singing the lyrics). In the third verse there are four voices (Soprano, Alto, Tenor, Basso); only Alto and Tenor are singing the lyrics, and the main melody line is in Alto.The fourth verse has the same voices as in verse 4 but a different arrangement, all voices are singing the lyrics, and there is a *change of key* (from C major to D major). This verse has *explicit accompaniment* (chord analysis), but the instrument that should perform this is unspecified.

Explicit accompaniment has not been used in the previous examples. It was shortly described in section 5.5. Observe that whenever there are several parts sounding at the same time stamp, it should always be possible to automatically deduce the accompanying chords *(implicit accompaniment*) and represent them as AbsoluteChords.

In verse 4, the main melody line is sometimes shared between several voices. As in the Telemann Aria, for consistency reasons only one voice is marked as having the main melody line (here, Soprano). The Pieces and Parts are shown in figure 15.

7 Conceptual, internal, and external views

As mentioned in section 1, it is the *conceptual* schema that has been presented in this paper. Both the internal and external schemas will reasonably look very different from this, for the following reasons.

1. **Internal view**: the conceptual model is not efficient enough to process and compare huge melodic material.

The tone vectors on the leaf-node levels in the Parts can be processed in different ways for efficiency. Above all, the music material that is hierarchically structured needs to be *flattened* so that a lot of tree traversals are avoided. An interesting target representation for this is a two-dimensional *matrix*, where image processing operators can work directly on a pattern of polyphonic music, where the tones are represented as integers (as in a *pixmap*). This representation is quite natural since a musical score itself resembles such a matrix. Furthermore, the tones in the conceptual model are already included in parallel vectors, so the transformation is relatively straight-forward.

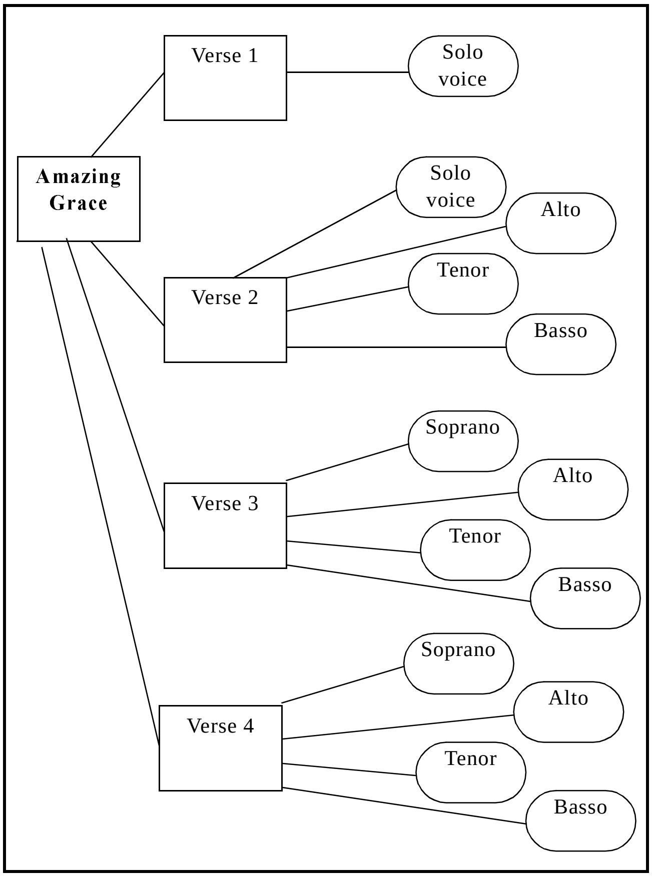


Figure 15: Pieces and Parts of Amazing Grace

A related approach, although working on bitmaps instead of pixmaps, is described in [2] and [3].

Regarding the deep nestling of the Pieces-and-Parts trees shown in section 6, [21] points out that deep-nested collections are normally a bad idea, and results in inefficiency. Despite of this, we have allowed a theoretically unlimited level of nesting in the conceptual schema. The reason is that this is a very natural representation for music, especially as music is extremely dependent of order. For a flattened internal representation, however, the nesting can be replaced by *identifiers* that indicates how the items should be grouped. For example, the Pieces and Parts in the Saltarello shown in figure 11 should then have the following IDs:

IDs:

1	Saltarello
1.1	Section A
1.1.1	Verse A1
1.1.1.1	Common Part A (Piece)
1.1.1.1.1	Common Part A (Part)

1.1.1.2 Ending 1 (Piece)
1.1.1.2.1 Ending 1 (Part)
1.1.2 Verse A2
...
1.2 Section B
...

Furthermore, techniques for *indexing*, *compression*, and *reduction* must be investigated.

2. **External view**: the conceptual model is not very suitable for an end user.

It would sometimes be rather tedious for an end user to directly query the conceptual schema. As a low-level example, a *chordName* is a type consisting of two scalars: the key (e.g., "C") and the harmonic function (e.g., "small septima"). Normally you write that as "C7". It is very easy to transform a chordName to that compact form (a function *compactChordName* does this). The same goes for other non-scalar types, like AbsoluteDuration and AbsoluteTone.

The reason for the more complicated representation on the conceptual level is to preserve *all* information so that it is easy to extract. It is trivial to convert this representation into a text string, but not always the other way around.

Ideally, the user should be able to use a more "musical" interactive method than to give tones, chords and other musical entities in an "SQLish" format (see section 8). There should be a graphical and/or audio GUI, with input methods like a pseudo-keyboard, a musical score, or even direct voice or instrument input. However, these types of interaction are beyond the scope of this work.

8 Examples of queries

Typical queries for current music databases includes items like artist, composer, title, and so on. This is also possible in SMORMIA (using the entities shown in figure 8). However, the real interesting queries are those where you combine high-level data with the low-level musical contents. The query below does this; it asks for the length of the score (numberOfBars), the estimated playing time in minutes (lengthOfPiece), and if the copyright has expired yet (in Sweden, the rule is that the copyright is protected 70 years after the death of the originator/s). The aim of such a question can be to find music for teaching students to make a harmonic arrangement.

```
select title(o) from OriginalPiece o
where numberOfBars(musicalContents(o)) < 20
 and lengthOfPiece(musicalContents(o)) < 1
 and copyRightExpired(o,yearNow(),70)
 = "true";
```

Both numberOfBars and lengthOfPiece are calculated directly from the musical contents (which is a Piece, see figure 8). For faster processing, they can be stored as attributes instead.

It is also possible to ask real low-level queries about the musical contents. The following query returns the positions in the fourth verse of "Amazing Grace" where the accompaniment is a G major chord:

```
set :verse4 =
  vref(sections(:amazingGrace),3);
findAccompaniment(:verse4,"G");
```

The next query returns the positions in the first verse and first part (Soprano) in "Amazing Grace", where the lyrics is sung on the vowel 'a':

```
set :part1 =
  vref(allParts(vref
         (sections(:amazingGrace),0)),0);
findVowel(:part1,"a");
```

These queries (and others of equal simplicity) are written in AMOSQL [24] and are fully executable. For more interesting queries, see suggestions for future work in section 9.

9 Conclusions and future work

A conceptual schema for representing music in an object-relational database has been described, and some examples of representation of music have been shown. For these examples, the model seems to be sufficient for representing different features present in musical scores, at least for analysis and search purposes.

However, for certain types of music, the model seems less suitable. Consider the Aria (section 6.4) where three separate Parts had to be introduced for Viola, with just a few notes included in the second and third parts, and for the alternative voice notes. This also applies for many piano pieces.

A possible solution to this would be to let a tone vector contain not only simple DurationUnits, but also vectors of DurationUnits. Then, a representation like

```
PartA = tones[tone,tone,tone,tone,tone]
PartB = tones[rest,rest,rest,tone,tone]
PartC = tones[rest,rest,rest,tone,rest]
```

can instead look like

```
Part =
tones[tone,tone,tone,[tone,tone,tone],
      [tone,tone]]
```

The drawback of this approach is that it destroys the simplicity of the model, but the advantages may be dominating in specific cases. Recall that alternative SMORMIA implementations was made for several of the examples in [19]. It is natural that alternative representations like these are allowed, due to the specific requirements of each genre.

Some interesting side effect can be obtained from the irregularities of the piano/keyboard scores (compared to other instruments). For example, you can choose to create one Part for each *finger*, according to how you would teach a piano novice to play, and thus obtain automatic finger setting if you ever use the model for printing musical scores.

Future work

The next steps in developing the model and testing its feasibility and robustness are the following:

1. As mentioned in section 6, some elements of the model remain to be tested. The *formSchema* will be utilized when bigger pieces of music are implemented. Apart from search and analysis issues, the formSchema can be a help when the music is initially inserted into the database.
2. A musical query language (*MusiQL*) will be designed and developed. It will be based on AMOSQL [24] but will otherwise confirm to *SQL-99* ([5], also called *SQL3*) as far as possible. When constructing the operators of MusiQL, requirements for harmonic analysis will be especially considered. (See examples of functions, *harmonicRole* and *harmonicRelation*, in figure 7).
3. A lot of functions to facilitate queries, as the *compactChordName* mentioned in section 7, will be developed. Furthermore, enforcement of *integrity constraints* should be introduced (for example, triggers that are activated for insertions).
4. It should be possible to ask more interesting queries. For example, consider the following pseudo query:

Find some arias which fulfil the criteria that the genre is opera or oratorium, it is written for a contralto, the range is in (F3..E5), the style is dramatic, the language is either Swedish, English, or Italian, the difficulty is medium, and it is composed no later than 1750.

Since harmonics is our basic interest, there should be a lot of operators that handles harmonic information. For example, it should be possible to ask for music that has rich harmonics (e.g., more than the basic chord, the dominant septima chord, and the sub dominant chord)

5. A huge amount of music will be loaded into the database to test scalability. This will include as many different types of music (genres) as possible, to see where the model is underdeveloped or overdeveloped.
6. The external and internal schemas will be defined. Index structures will be constructed, and the feasibility of compression and reduction of the music representation will be investigated.
7. Low-level methods for searching, e.g. text-based information retrieval or image processing methods, will be investigated.

10 References

[1] Cronin, Charles.: *Concepts of Melodic Similarity in Music-Copyright Infringement Suits*, chapter 10 in [7]

[2] Dovey, Matthew: *An algorithm for locating polyphonic phrases within a polyphonic music piece*, Focus Workshop on Pattern Processing in Music Analysis and Creation, Symposium on Artificial Intelligence and Musical Creativity Convention (AISB'99), Edinburgh, April 1999

[3] Dovey, Matthew and Crawford, Tim: *Heuristic Models of Relevance Ranking in Searching Polyphonic Music*, Proceedings of the Diderot Forum on Mathematics and Music, 2-4 December 1999, Vienna, Austria, pp. 111-123

[4] Fahl, G; Risch T. and Sköld, M: *AMOS - An Architecture for Active Mediators*, Proc. Workshop on Next Generation Information Technologies and Systems (NGITS'92), Haifa, Israel, June 1993

[5] Gulutzan, P and Pelzer, T: SQL-99, Complete, Really, Miller Freeman, Kansas, 1999

[6] Hewlett, Walter B.: *MuseData: Multipurpose Representation*, chapter 27 in [18]

[7] Hewlett, Walter B. and Selfridge-Field, Eleanor (eds.): *Melodic Similarity - Concepts, Procedures, and Applications*, Computing in Musicology 11, The MIT Press, 1998

[8] *Humdrum **kern Representation*, **http://www.lib.virginia.edu/dmmc/Music/Humdrum/kern_hlp.html**

[9] *The Humdrum Toolkit: Software for Music Research*, **http://www.music-cog.ohio-state.edu/Humdrum/**

[10] *The Humdrum Toolkit: Software for Music Research - A Bibliography*, **http://www.music-cog.ohio-state.edu/Humdrum/bibliography.html**

[11] Huron, David: *Humdrum and Kern: Selective Feature Encoding*, chapter 26 in [18]

[12] Kornstädt, Andreas.: *Themefinder: A Web-based Melodic Search Tool*, chapter 13 in [7]

[13] Lyngbaek, P et al: *OSQL: A Language for Object Databases*, Tech. Report, HP Labs, HPL-DTD-91-4, 1991

[14] *MuseData*, **http://www.ccarh.org/musedata**

[15] *MusicML: an XML experience*, **http://www.tcf.nl/3.0/musicml/index.html** (van Rotterdam, Jeroen)

[16] *Online Music Recognition and Searching (OMRAS)*: **http://www.kcl.ac.uk/kis/schools/hums/music/ttc/IR_projects/OMRAS** (Observe: this is listed as a *temporary* web site only!)

[17] Rubenstein, W. Bradley: *A Database Design for Musical Information*, Proc. of the 1987 ACM SIGMOD Interna-

tional Conference on Management of Data, May 27-29 1987, San Francisco, CA, pp. 479-490

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

[19] Selfridge-Field, Eleanor: *Describing Musical Information*, chapter 1 in [18]

[20] Shipman, D: *The Functional Data Model and the Data Language DAPLEX*, ACM Transactions on Database Systems, 6(1), ACM Press, 1981

[21] Stonebraker, Michael and Brown, Paul: *Object-Relational DBMSs: Tracking the Next Great Wave*, Morgan Kaufmann Publishers, 1999

[22] Taube, Heinrich: *Automatic Tonal Analysis: Toward the Implementation of a Music Theory Workbench*, Computer Music Journal, Vol. 23, No 4, 1999, pp. 18-32

[23] Temperley, David and Sleator, Daniel: *Modeling Meter and Harmony: A Preference-Rule Approach*, Computer Music Journal, Vol. 23, No 1, 1999, pp. 10-27

[24] Risch, T; Josifovski, V and Katchaounov, T: *AMOS II Concepts*, **http://www.dis.uu.se/~udbl/amos/doc**

[25] Wiederhold, G: *Mediators in the Architecture of Future Information Systems*, Volume I & II, Computer Science Press, 1988, 1989

[26] Yip, C.L. and Kao, B: *A study on musical features for melody databases*, Proc. of the 10th Int. Conf. on Database and Expert Systems Applications (DEXA’99), Florence, Aug. 1999