Content-Based Retrieval of Musical Scores in an Object-oriented Database System

Marisa Beck Figueiredo
marisa@icmsc.sc.usp.br

Caetano Traina Júnior
caetano@icmsc.sc.usp.br

Agma J. Machado Traina
agma@icmsc.sc.usp.br

University of Sao Paulo at Sao Carlos- Brazil
Computer Science Department
Po. Box 668 CEP 13560-000 Sao Carlos - SP
Phone: +55-16-274-9128 fax: +55-16-274-9150

Abstract: The aim of this work is to present the Musical Score as a new attribute characteristic in an object-oriented database. It defines a new data type for attributes that can be associated to objects using the MIDI Standard, the Standard MIDI File format, and the General MIDI standards. To offer complete database support to musical scores as a data type, it is necessary to provide storage, manipulation, search and retrieval methods. This work uses the inner structure of Musical Scores, following the MIDI standard, to enable the creation of index mechanisms in the database, allowing for the retrieval of musical scores based on their contents.

Keywords: Musical Scores, Object-Oriented Databases, Standard MIDI Files, Complex Objects Data Retrieval, Content-Based Musical Scores Retrieval.

1. Introduction

Database Management Systems (DBMS) are commonly used when a great amount of data needs to be stored. The DBMS are responsible for the organization, storage and retrieval of data from the database. The data stored in a database must be organized following a set of rules that conceptually structure these data. This set of rules is called a “Data Model”, and when someone uses this data model to describe a given situation, the result is called a “Data Modelling”. In recent years, a new approach has been increasingly used: the development of Object-Oriented Data Models (OODM), to support the construction of Object-Oriented DataBase Management Systems (OODBMS). In such models, the basic idea is to represent the concepts and things of the real world as “objects”, describing for each one its structure (attributes, properties, and so on) and its behaviour (procedures and operations that the object executes as a result of external stimuli received) [Elmasri-94].

The object-oriented data models [Bancilhon-92] [Bertino-93] [Zand-95] have more capacity for data representation than the traditional models, mainly because: OO data models can represent the behaviour of real world entities - this is not the case of the traditional ones; and the OO data models usually have a richer set of modelling elements, too. An element that contributes to increase the modelling power of an OODM is its ability to define new “Data Types”. Traditional data types, “natives” in most of the DBMS (either OO or not), are integers, real numbers, bytes, character strings and so on [Bancolhon-92]. Beyond these native types, the system designers add specific data types needed to his/her application domain, defining new "object types". These application-dependant object types establish the structure and behaviour of the objects for this newly defined type.

The behaviour of an object is defined by a set of methods, which define how the object operates in response to each external demand. Among these demands are the requests for inter-operations with objects either of the same type or of other types. For example, native objects of type integer can be added, multiplied, etc. one with each other. Objects of a specific domain of type cost can be added with a float number. An object structure is defined by its set of properties, the *attributes*, associated with the object. Each object attribute can receive one or more *values* chosen from a set of values permissible for the data type of this attribute.

The work presented in this paper was developed in a specific OODM called SIRIUS [Biajiz-96]. It holds all the object-oriented concepts described, and extends the concepts of data type of an attribute, through the concept of "Attribute characteristic" (or characteristic for short). An attribute characteristic is the set of generic operations that can be automatically applied over attributes of this characteristic by the OODBMS.

For example, the data types integer, float, double, etc. can execute the four arithmetic operations, they can be compared (by order or equality), etc. Thus, all of these data types are of the *number* characteristic. The data types character string, ASCII text, RTF text (Rich Text Format), etc. can be concatenated, and some of their parts can be changed, compared with sub-strings and so on, but they cannot be multiplied. These data types constitute another attribute characteristic, the *text* one. Figure 1 shows an organization of these data types. In SIRIUS, methods associated to an object are also considered to be an attribute, in this case of the characteristic *Rule*.

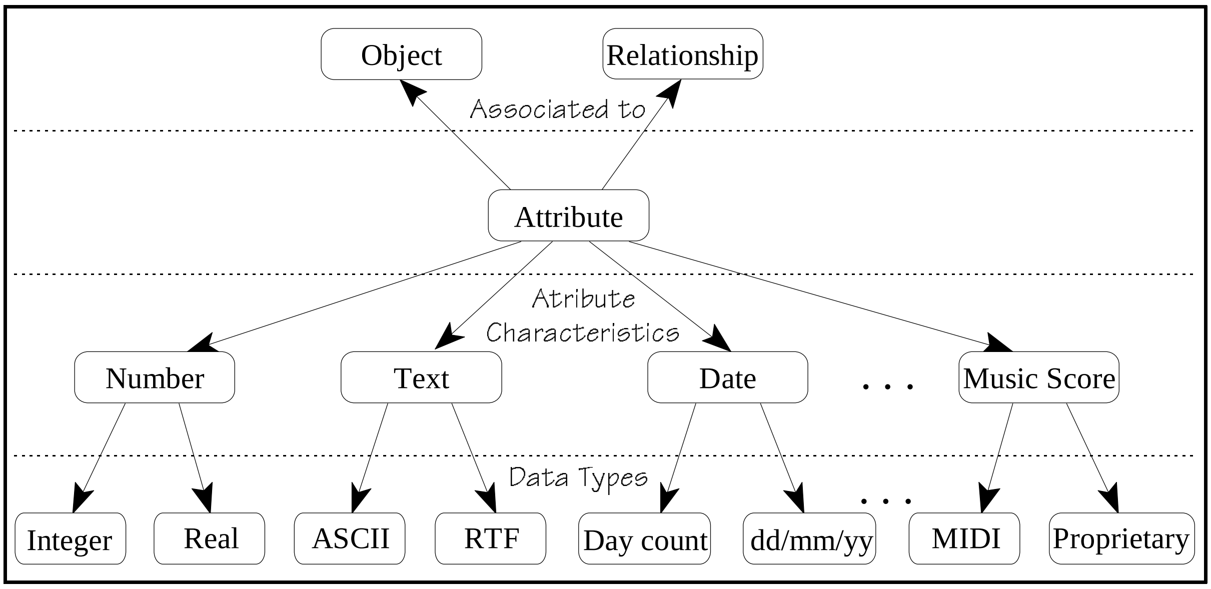

Figure 1- Where attributes are associated to their characteristics and data type.

The operations supported by an attribute characteristic can be performed involving attributes of different data types of this characteristic. For example, numbers of type integer can be multiplied by numbers of real type, while texts of data type RTF can be searched by sub-strings of data type ASCII, etc. However, it is not always possible to inter-operate attributes of two different characteristics with operations of any of the characteristic. For example, it is impossible to divide a number by a string. To define a new attribute characteristic, it is necessary to first define the set of generic operations attached to this characteristic, and at least one data type.

This paper applies these concepts to the representation of musical scores. In this way, Musical Score becomes an attribute characteristic that can be represented in different ways, each being a Musical Score Data Type. We use musical scores represented in the *Standard MIDI File* (SMF) format (MIDI stands for *Musical Instrument Digital Interface*) [Huber-95] [Stolz-93] [Ratton-95]. Thus, SMF is a data type of the Musical Score characteristic, although the adopted data model could support other formats to represent musical scores as other data types, each using the same operations.

This paper deals with the operations of Musical Scores that can be used to store, manipulate, search and retrieve scores in an object-oriented database system. Its structure is as follows: section 2 describes how to generate sound and how sound is represented in computer files. Section 3 describes the musical score concept and the manner in which an SMF represents musical scores is explained. Section 4 describes what needs to be done to store an SMF in a database, so that it can be searched according to its contents. Section 5 describes how the SIRIUS model can be used to build an OODBMS to enable it to support the Musical Score Characteristic. And finally, section 6 presents the concluding remarks of this paper.

2. Audio Representation

Sound is the mechanical movement of materials. The sounds audible to the human ear are called Audio Sounds. There are two basic ways to represent sounds: either through direct representation in a given way, or through a set of parameters that describes sounds that can be used to re-create sound. The first way is called *Sound Recording*, and the second is called *Sound Synthesis*.

In Sound Recording, the sound is collected by a type of transducers, such as a microphone, and after some processing, it is stored in such a way as to represent the original mechanical movements. When computers are used to manipulate recorded audio sounds in some way, it is usual that the processing of the sound converts the analog measurements to a digitalized form of representation. This is called the *Digital Audio* representation.

Sound Synthesis is commonly used to generate sounds that are not previously recorded (although some recent studies allow us to analyse a ‘real world’ sound by extracting its parameters, so it can later be reproduced based only on those parameters). Hence, Sound Synthesis is not commonly used to accurately represent ambient sounds or the human voice, but rather, it is widely - and increasingly - used to represent musical sounds and instruments.

Many technologies have been developed to represent sounds in both ways. For example, digitalized audio can be quantized by a linear, incremental or logarithmic table, can be compressed by many algorithms, and can be sampled in different rates, interpolated or not, etc. There are also many technologies to represent and generate sounds through synthesis. Currently, two of the audio synthesis technologies most frequently adopted by the computer industry are Frequency Modulation (FM) and Sampling Reproduction (Sampling).

There are many established standards to represent audio sounds in a computer. The .WAV (WAVe riff) format is presently the most commonly employed to store Digital Audio. In this format, the sampled values are stored in sequence, alternating the values of both channels if a stereo sound is being represented. A header in the beginning of the file sets the sample rate, the number of channels, and the number of bytes used to store each value. The Standard MIDI File format is currently the most commonly employed to store musical scores [Stol93] [Hube95] [Rattpn-95]. This format represents a piece of music through the sequences of notes, the tempo and the duration that each note should be played.

Other file formats are used to represent instruments, so a musical score can be composed to be played using a specific set of instruments. There are formats specific to each sound synthesis technology, as there are file formats that incorporate both the score and the instrument definition in a same file. Figure 2 represents some file formats in a symbolic form.

For instance, the .SBI format ("Sound Blaster Instrument") describes how an instrument can be synthesized from an FM-based hardware synthesizer. It uses a set of parameters to control many aspects of sound generation, such as the waveform, its volume and filtering variations over time, and so on. The .SBK ("Sound BanK") stores up to 128 instrument definitions in one file, each like the .SBI format. The .SNG ("SoNG") format mixes a score, represented in a way similar to the SMF format, with a set of instrument definitions, each one in the same way the .SBI format represents one instrument. All the .SBI, . SBK and .SNG formats are specific to the use of FM Synthesizers. The .MOD ("MODulation") is a format like the .SNG, but the instruments are represented using the sampling technology. In this format, each instrument consists of a sampling of one note of the intended instrument. To play the music, each note played by each instrument uses the corresponding sampling modulated, so the reproduced frequency corresponds to that of the intended note. The .MOD file format is a rather old one, and is restricted to permit only four instruments to be played together. A plethora of similar formats was developed to extend the much used .MOD format, each adding some new feature. Recently, the MIDI Manufacturers Association (MMA) has been working on the definition of a new standard file format, so a set of instruments can be represented in a file, using the sampler's technology, the .DLS ("DownLoadable Sound") format [AMM-97].

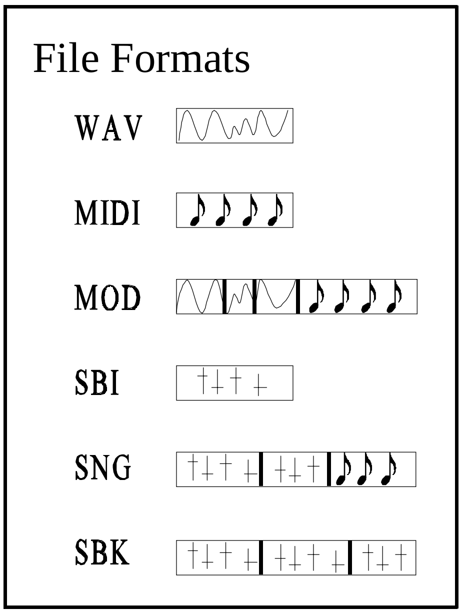

Figure 2 - Contents of some file formats.

3. The .SMF Format and Musical Score Make-up

A musical score is basically a two-dimensional representation of the sound sequence that constitutes a musical piece. The two basic dimensions represented are: the musical notes that each instrument must play and the moment when each note sounds. A full-featured score represents other dimensions, because many other elements are put in a score, such as the intensity of each note, its inflection, and so on. However, the two basic dimensions are essential and sufficient to represent music, and also adequate for the sake of this paper.

MIDI was originally conceived as a standard way to interconnect some electronic musical instruments, mainly keyboards (like the piano) and electronic synthesizers. In this way, a musician could play on a keyboard and a set of synthesizers (sound generators) could receive the events being generated, and produce the corresponding audio sound. This communication occurs in real time, so each event results in a specific action immediately. Connecting pairs of instruments are made through a serial port, so a set of 16 virtual 'channels' can be emulated, each dealing with an instrument.

There are two types of events defined by the MIDI standard: standard MIDI events; and System Exclusive events. System Exclusive events are events that do not follow any standard. Their purpose is to maintain the particularities of each manufacturer, so the distinctive peculiarities of their instruments can be accommodated. Standard MIDI events are involved with a musical performance, and include the beginning and end of each note (when the musician presses and releases each key), the binding (or changing) of an instrument to a channel, and the definition of some musical parameters to each channel, like volume, detune, etc. Table 1 shows the basic standard MIDI events, and the parameters associated to each one.

Standard MIDI Event	Meaning	Parameters
Note ON	Play a note	channel, note and intensity
Note OFF	Finish playing a Note	channel, note and release intensity
Control Change	Change a parameter	channel, parameter number and new value
Program Change	Change a timbre	channel and the new timbre associated to this channel
Polyphony aftertouch	Change parameter of an already sounding note	channel, note and new parameter value
Aftertouch	Change parameter of a channel	channel and new parameter value

Table 1 - Standard MIDI Events.

Excluding the channel number, which ranges from 0 to 15, all information in the MIDI standard is represented by integer numbers ranging from 0 to 127. In this way, the notes and their intensity are numbered from 0 to 127. The timbre (or instrument sound quality) assigned to a channel must be numbered from a 0 to 127, and the parameters chosen from a set of parameters numbered from 0 to 127. For example, number 7 means overall volume, and 11 means the stereo panorama (positioning of the channel in the stereo field).

The MIDI standard does not represent time, since each event must be responded in real time. The MIDI standard, therefore, represents only one of the two basic dimensions of a musical score: the sequence of notes played by each instrument. To record a musical performance in a computer file, each event needs to be stored with the time stamp of when it must occur in the performance. The way that a musical performance can be stored in a computer file was standardized through the Standard MIDI File format (SMF). This format adds the second dimension of a musical score and is, in fact, a way to represent musical scores. The MIDI standard and the SMF format are fully specified in the Official MIDI Specification Document [MMA-96].

Each Standard MIDI File stores only one musical score and is composed of a set of “chunks”. There are two types of chunks: the header chunk and the track chunk. Each file always has one header chunk, which sets parameters for the whole score, like how much time signifies one time unit for this piece, the file type and the number of track chunks. The file type can be 0, 1 or 2. The actual musical information is stored in one or more track chunks, or tracks for short. Files of type 0 store all musical information on only one track. Files of type 1 store the musical information on many tracks, but all the tracks start to play simultaneously, at the beginning of the performance. Today, this is the most commonly used MIDI file type. Files of type 2 also store the musical information on many tracks, but each track can start to play at different times, allowing representation of musical scores with more than one movement in the same file.

Although there is no definition of what kind of information each track stores, it is a common practice to store the part of each instrument on an individual track, so each type 1 or type 2 MIDI file has a separate track for each played instrument. The track structure of a MIDI file has no relation to the virtual channel link of the MIDI standard. A track can store events of many channels, and a channel can have events on many tracks. However, almost all sequencers reserve a track to store the notes of each timbre played by each (electronic) instrument.

If a file has more than one track, they are stored in sequence, in no predefined order. Each track stores a sequence of MIDI events, in the time sequence they occur. The time is represented through the indication of a “delta time”, which indicates how much time units pass between an event and the next on the same track. In this way, during a performance, the tracks can proceed parallel to each other, with events occurring from any of the already started tracks.

The SMF format also defines a new set of events, not directly related to the musical information needed to represent a performance, but intended to provide information about a score, such as the name (a text readable for humans) of the instrument intended by a track, as well as copyright information, etc. The Meta-events are useful to document a score, or to store information for computer software that handles this file. This kind of event is called a “meta-event”, and although stored in a standard MIDI file, they are not sent through a physical MIDI link, as they do not affect the real time performance. The meta-events are not needed in a MIDI file, and only few meta-events have a standardized meaning. Table 2 in section 4 shows some standardized meta-events.

The two standards discussed so far are not enough to guarantee that a MIDI file will sound in a similar way in two different MIDI setups. This is due to the fact that the association of a specific timbre to a timbre number is not defined in either standard. So a third standard was conceived, whose purpose is to define which timbre corresponds to each timbre number (or to each Program number, in MIDI terminology). This standard, called the “General MIDI” (GM) standard, predefines instrument timbres to all 127 instrument numbers, like 1=Piano, 2= electric piano, and so on. The GM standard is not much used in professional fields, because it restricts the expressiveness of detailed, specifically constructed timbres. However, it is a good way to give an overall version of a musical piece, and is extensively used in general multimedia-based products and presentations.

4. Storing Scores in a Database.

This work considers that the musical scores are originally represented in MIDI files, and that the scores are retrieved from the database as MIDI files. This is because MIDI files are the commonest way to disseminate and to interchange musical scores using digital media, and because this is an easy and environment-independent way to listen to the execution of any score.

There are two ways to create a database of musical scores starting from a set of MIDI files: making each object stored in the database “point” to an external file; or storing the file inside the database, together with the other information. The former approach is not a good solution because the responsibility of the storage and of the places where the files are kept is delegated to the operating system, so the DBMS cannot guarantee the consistence of the set of pointers.

The latter approach is the more appropriate choice, and also has two other options. The first is to store the original MIDI file as a “Binary Large OBject”, or a Blob. This option disobliges the DBMS to recognize the internal structure of the Blob, so it is treated as an amorphous mass of data. The second option is to analyse the original MIDI file as it is stored, so its internal structures can be recognized by the DBMS. This option permits the DBMS to answer queries based on the contents of the stored information, and it is the choice of this work. To reach this goal, the concept of Attribute Characteristic is explored, defining a new one: the Musical Score Characteristic. Figure 3 illustrates these different options to store musical scores.

Using this concept, an attribute with the musical score characteristic can be associated to any object or relationship of a database schema. As in the case of any other attribute, each of these attributes must have a name, a data type and a set of values. The name of one attribute is given by the application designer, in the same way as he/she gives names to any other attribute of the application. For example, if the application needs to have objects of type `Country`, storing the national anthem of each instance, `Anthem` could be the name of an attribute whose characteristic is Musical Score.

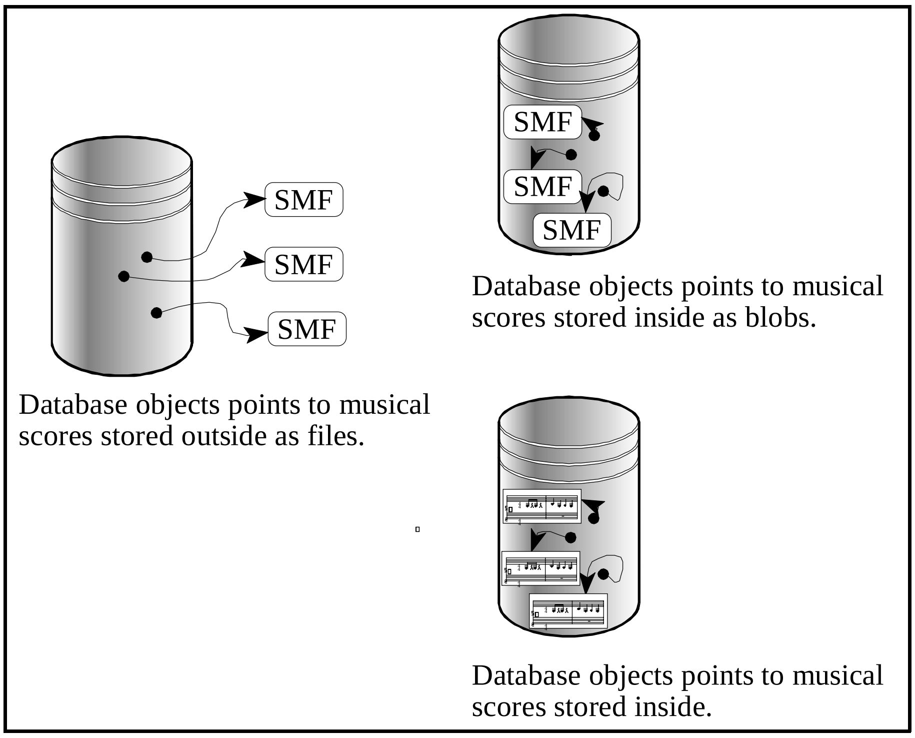
SMF
SMF
SMF
Database objects points to musical scores stored outside as files.
SMF
SMF
SMF
Database objects points to musical scores stored inside as blobs.
Database objects points to musical scores stored inside.

Figure 3 - Ways that a Musical Store can be stored in a database.

The data type of a Musical Score attribute is the specific format used to store the score. This work uses only the MIDI format, so the data type is MIDI. The set of values of a score attribute is a set of scores, each a complete one. Continuing the previous example, it can be said that each `Country` will have its `Anthem` stored in the MIDI format, and that the set of values will be composed of a single value, defining the score of the national anthem. Supposing, also, that the application needs to store a list of commemorative anthems of each country, each indicating the commemorative happening and the score of the anthem itself, then a new attribute, which will be a list of tuples, called `Commemorative_Anthem`, could be associated to the `Country` object type. The tuple will consist of one attribute called `C_Anthem` of the score characteristic, whose data type is also MIDI, and a text attribute called `Happening`. Each value of the attribute `Commemorative_Anthem` will be a pair of a Score attribute (the `C_Anthem` attribute) and a Text Attribute (the `Happening` attribute). Figure 4 shows a draft of the schema of this example. This approach is an improvement over the rigid definition of an object of type Music, whose instances the scores would be associated to, because this situation is only a particular case of the proposed one.

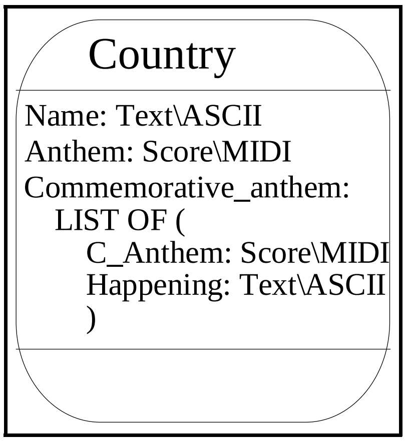

Figure 4 - Example of musical scores associated to objects.

To maintain the consistency of the information stored in the database, it is necessary to prevent all the possible causes of inconsistencies. As one MIDI file has more information than just a musical score, it is important to remove all that extra information from a MIDI file when it is being stored in the database. So, when it is indicated as the file containing one value to be assigned to an attribute, it must be read and analysed and only the relevant information filtered.

The data of each score is always contained in the standard MIDI events and in the Meta-events. The third set of events, the System Exclusive events, are required only to adapt the execution of the score to the use of a particular MIDI setup, specifying the sound generation parameters to each instrument. If another attribute characteristic - the Instrument Definition one - were defined, then these events could be the source of data to these attributes. Since, in principle, it is not necessary to be aware of these events, then in the filtering process all system exclusive events are discarded.

However, meta-events provide important information about each score, although they are not part of the score themselves. Therefore, in the filtering process they are also removed from the MIDI file and stored as attributes of other characteristics associated to the score in the same manner as any other attributes of the database. The specific characteristic used to store each meta-event depends on the meaning of each particular meta-event. For example, the meta-event `Instrument` is stored as an attribute of the characteristic text. After the filtering of the MIDI file, only the standard MIDI event remains with its corresponding timing information. This data is then stored as the value of an attribute of the musical score characteristic. Figure 5 illustrates how a standard MIDI file is transformed and its information stored as a musical score attribute value.

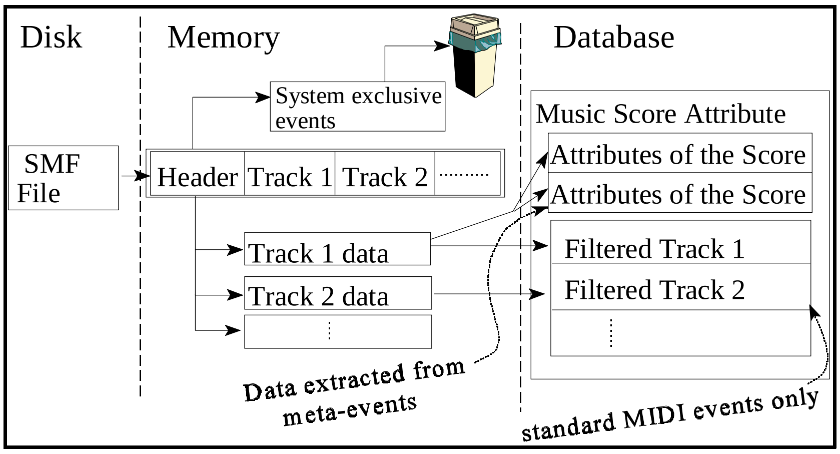

Figure 5 - Filtering the value of a musical score attribute from a MIDI file.

The reason to separate the meta-events from the score, and to store the meta-events as values of other attributes is that the user could, inadvertently or not, store the same information in another part of the object related to that score. As the information reported by a meta-event is, in fact, a number, a text or a date, then it makes sense to store that information as an attribute of a characteristic number, text or date. This prevents redundancy and provides increased flexibility for the solution.

The musical score characteristic is a complex one, and requires that some support be provided by a system that uses it. This includes the pre-definition of all standardized meta-events as attributes that can be associated to the objects or relationships associated to attributes of the musical score characteristic. One application or a database schema that includes an attribute of the musical score characteristic is called score-enabled.

Table 2 shows the main meta-events treated, showing their numbers and meanings in the SMF. Each meta-event is also given the name of the predefined attribute at the moment it is stored in the database, as well as its characteristic and each object to which it is associated.

Conceptually, an attribute value in an object-oriented database is also an object. We call this kind of object a "primary object". A score is an attribute value, so it is an object, and it can have other attributes associated to it. In this way, an object or relationship of a user defined type in the database can have associated attributes of the musical score characteristic. Each value of those attributes is an object, whose type is a predefined one, the type Score, and that can have predefined attributes associated to it. These attributes are at least the set of tracks extracted from the original MIDI file, each one also considered to be another primary object of the predefined type Track. The meta-events extracted from the original MIDI files are in turn associated to the Score and Track objects. Some events, whose position inside a track in relation to the track timing is important, are associated to the Track or Score objects together with the timing offset.

Meta-event number	Meaning	Attribute name	Attribute Characteristic/ data type	Association
0	Track sequential number	`Sequence_Number`	Number/integer	Track
+1	Free text	`Score_text`	Text/ASCII	Score
2	Copyright	`Copyright`	Text/ASCII	Score
3	Track Name	`Track_Name`	Text/ASCII	Track
4	Instrument Name	`Instrument`	Text/ASCII	Track
5	Song lyrics	`Lyrics`	Text/ASCII	ScoreOffset
8	Author Name	`Author`	Text/ASCII	Score
81	Relative change of song speed	`Tempo_change`	Number/long	Score
84	Timing offset (SMPTE)	`Timing_Offset`	Number/byte[5]	ScoreOffset
88	Timing Resolution (SMPTE)	`Timing_resolution`	Number/byte[4]	ScoreOffset
127	Sequencer free data	`Sequencer`	Text/ASCII	Track

Table 2 - Meta-events.

By defining meta-events as a regular attribute and not as part of the internal MIDI-formatted musical score data, the user can freely manipulate them. For example, it does not matter if the `Author` of the music was obtained via one meta-event of the MIDI file or was inserted by a user using a regular attribute - the information is always the same and unique. Whenever a stored score needs to be extracted from the database into a new MIDI file, the user can indicate the set of meta-events he/she wants to have included in the file. Thus, an operation opposite to the filtering one used to store the file is executed, which generates a MIDI file including the MIDI events that constitute the basic score, and the user-selected meta-events. It is not important to know how the value of each included meta-event was inserted in the database.

As an example, using the schema shown in Figure 4, the following command could be used in a score-enabled version of Object-SQL to extract the national `Anthem` of the `country Brazil`, and to store it in a new MIDI file called `Brasil.MID`.

```
SELECT Anthem, Anthem.Author, Anthem.Tracks_are.Instrument_name
FROM Country
WHERE Country.Name = “Brazil”
INTO /path/Brazil.MID   /* the syntax of this line is slightly
                        different! */
```

The resulting file `Brasil.MID` includes: the score of this anthem stored on the existing tracks as standard MIDI events (`Anthem`); the `Author` of the music as a meta-event included on the first track (`Anthem.Author`); and the name of each instrument played on each track included on the corresponding tracks (`Anthem.Tracks.Instrument_name`), also as meta-events.

The syntax of this command imposes some restrictions on the generic syntax of the Select/from/where command of the SQL language. The main restriction is that, when an attribute of the characteristic Musical Score is indicated, all other indicated attributes must be attributes related to that score. Our implementation inserts all other information extracted by such commands as a text of the meta-event `Score_text` in the first event of the first track of the generated .MID file.

5. Architecture of the Software Implemented.

To demonstrate and validate these concepts, a software prototype was implemented based on an OODBMS called SIRIUS/GO, which supports the SIRIUS data model [Biajiz-96], extended to embrace the Musical Score characteristic [Figueiredo-97]. As the extension to the database manager supports only the operations of storage, query and retrieval of musical scores, it was also necessary to construct an application tool, intended to allow the user to interact with the system. This tool was developed in the Windows-95 environment, in C++ language. It is called IHP-SQL, an acronym for Interactive Score-enabled SQL Editor in Portuguese, and its general architecture is shown in Figure 6.

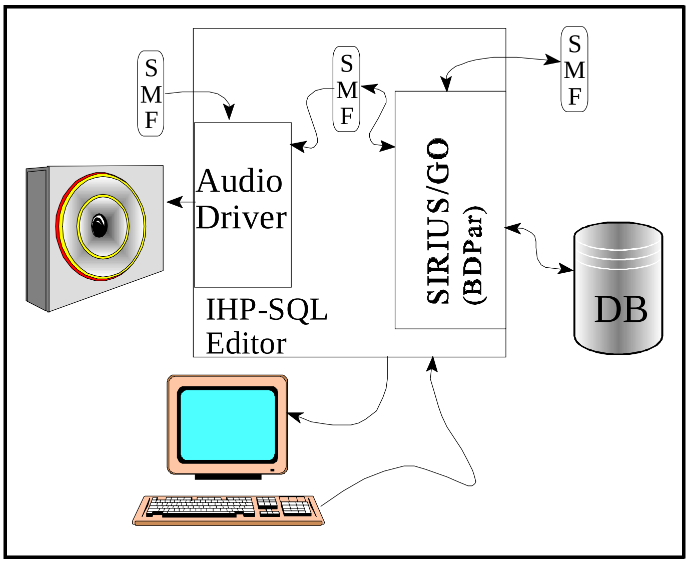

Figure 6 - IHP-SQL general architecture.

This tool was developed to achieve two purposes: first, to enable the user to ask the database manager to perform the operations of store, query and retrieve Musical Scores; second, to permit the user to do some other basic operations on scores, like to play a score and to prepare short melodic lines to be used in the query commands. In this way, the Database Management System does not have any responsibility to interact with music player drivers: it simply manipulates the data structures associated with the representation of musical scores.

The retrieval of scores stored in the database is done in two ways: generation of a MIDI file or of a “response set” in memory. The generation of a MIDI file is done on a file by file basis. The generation of a response set permits that a set of score attribute values can be generated and submitted to the application. In both situations, the retrieval operation constructs memory-based objects that include the data indicated by the retrieval command. It is worth mentioning that when a score is retrieved, it includes the indicated attributes only if the attribute has a defined value. Nonetheless, if it has a value, it is not important to know how the value was defined, that is, if it was retrieved when the original MIDI file was read or if it was inserted manually by the user.

The IHP-SQL tool is an interpreter of an extension of the object-oriented SQL language [Bertino-93]. There are few modifications in the language itself, because Scores are treated as a new kind of attribute. The modifications are those that permit definition of attributes of the Musical Score characteristic in the `Define Object` command, and the identification of the terms of the three adopted standards (the MIDI standard, the Standard MIDI File format, and the

General MIDI Standard) as constant values.

The SIRIUS/GO database manager is constructed using the concept of attribute characteristics, so the extension made to support Musical Scores consisted basically of the inclusion of the support of a new characteristic. The extension is based on the meta-system concept, in which the data model is represented in the system as a meta-schema. To support the Musical Score characteristic, the meta-schema was extended to include the information corresponding to the three standards adopted.

Figure 7 shows the extension made in the meta-schema. The object named `Object` represents the concept of Object, and it is the meta-type of all types of objects defined in the schema. The object named `Attribute` is the concept of Attribute, and represents all Attributes defined in the schema. The arc labelled `Associates/Associated_to` is the representation that associates attributes to objects (Figure 7 does not show that those attributes can also be associated to relationships). The notation represents the fact that a `Score` is a specialization of the concept of attributes. Like any attribute, Scores are objects, and they can have the attributes associated to them. They always have the attributes `N#tracks`, `Tempo` and `MIDI_Type`, extracted from the MIDI file header, and the attributes `Score_text`, `Copyright`, `lyrics`, `Author`, `Timing_offset`, `timing_resolution` and `Tempo_change`, possibly extracted from a meta-event.

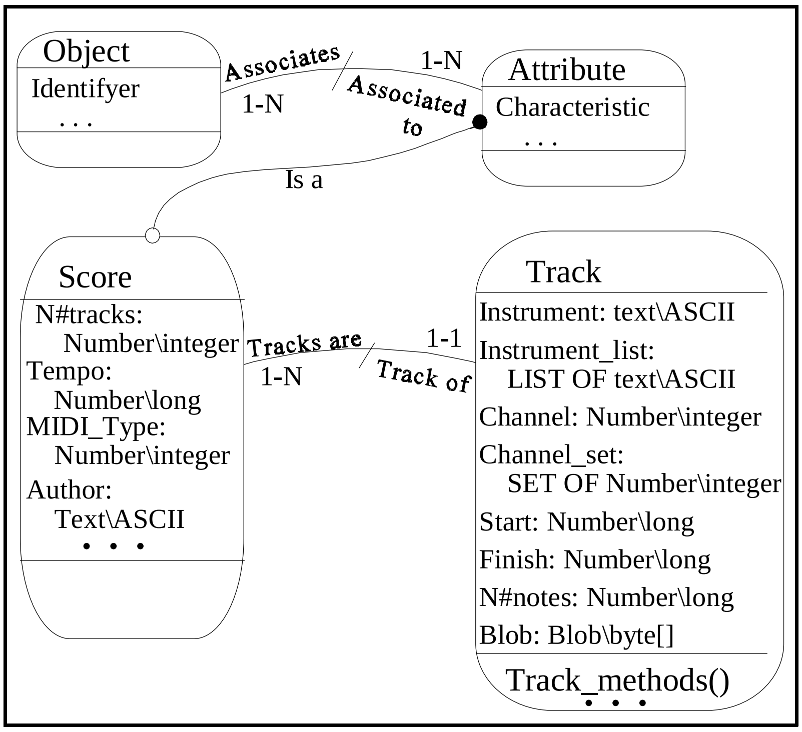

Figure 7 - Extension to the Meta-schema of SIRIUS/GO to support the Musical Score Characteristic.

A `score` object can also be associated to the relationship `Tracks_are/Track_of` to at least one `Track`. Tracks are objects defined in the meta-schema, and can have both the attributes shown in Figure 7 and the attributes named `Track_name`, `Instrument` and `Sequencer`, shown in table 2, associated to them. In particular, the attribute `Blob` is where the sequence of MIDI events that composes the track is stored. As a Track is a concept defined in the meta-schema, the methods assigned to the `Track` object meta-concept deal with its contents.

The extended meta-schema provides support to store all the data contained in a MIDI file in a database (except the system exclusive events). However, to complete the support for all three adopted standards in a score-enabled application, the application schema must be predefined including:

the definition of the Object Type `Instrument`;

the definition of 196 objects of the type `Instrument`, one for each of the 128 melodic instruments plus the 64 percussion set instruments of the General MIDI standard; and

the definition of the attributes corresponding to the Control Change standard events (12 attributes were defined in the current implementation).

When a database is created and the user must informs if it is score-enabled, upon which all these definitions are automatically inserted in an initial application schema that the user needs to complete his implementation dependent definitions.

To permit information queries based on the contents of the score, it is necessary to provide ways index the information contained in the scores, so that it can be treated by the OODBMS like any other information. This indexing of Musical Scores obtained from MIDI files can be made on two levels. The indexing of attributes of plain characteristics, like numbers and text, are common in any database, so the extraction of meaningful data from the MIDI file and its storage as numbers or text enables the indexing of this information in a simple way, that is, as any other text or number, a system designer can designate index structures for them. This corresponds to the first level of indexing of a Musical Score in the database, and can be accomplished by the predefined elements of the schema.

Using this level of indexing, the system can answer queries such as "What are the Commemorative Anthems, of any Country, played with a flute ?". This query can be expressed as:

```
SELECT Name, Commemorative_Anthem.Happening
FROM Country
WHERE Commemorative_Anthem.C_Anthem.Tracks_are.Instrument_name
      = "Flute"
```

The result of this query will be a list of the names of countries and the corresponding happenings that have `Commemorative_Anthem` with at least one track that uses a flute.

The second level of indexing corresponds to the indexing of information that cannot be represented as text or number. This occurs with the sequence of standard MIDI events stored in the attributes `Blob` on the `Tracks` of the Musical Scores. The indexing of that kind of information requires a definition of what needs to be searched, and what can be considered a match. This definition depends on what is the objective of the search, and on the musical considerations of what needs to be analysed. The search mechanism is also dependent on algorithms that can execute the intended search of information in each bulk sequence of events.

So far, we have developed the described system in the hope of making available a system with a basic set of tools to store and retrieve Musical Scores in a database, enabling the query of information based on the contents of the stored scores. The search operations implemented so far are only the basic ones, and more resources will be added to meet the needs of musicians querying this information, mainly based on the musicians' opinions about their intentions and experiences with the system.

The only search mechanism implemented so far is a rather crude one that works in the following way: When each MIDI file is read into the memory to be filtered, it is also submitted to an algorithm that traces the main scale of the whole song. Based on the recovered note each track is analysed to separate the musical "words". A musical word is considered to be a sequence of five or more notes played in a sequence that is separated by the next word or groups of less than five notes by a period at least four times greater than the maximum period between notes of that word. Only the notes pertaining to the identified scale are considered to be part of a word and only the one with the highest pitch of each chord. All the identified musical words are normalized to the C scale, after which they are indexed in a B^*-tree, rooted in the definition of the attribute from which the score is a value. In this way, each attribute of the Musical Score characteristic defined in the schema has its proper indexing structure, regardless of the track where each musical word is located.

6. Conclusion

This work describes a technique to extend an Object Manager to support the storage of musical scores as another characteristic of attributes that can be associated to objects, and to enable the content-based retrieval of these scores. It is common to think of Scores as object types, and not as attribute characteristics, as treated in this work. However, the latter approach is broader, so the representation of scores as object types is only one of the modelling possibilities enabled by it. Musical Scores is another attribute characteristic supported by the DBMS, like the numbers, texts and dates, and can be associated with any other modelling element that attributes can be associated with. The technique presented here is being developed focused on four objectives:

to allow the exploration of musical ideas in large sets of scores, - supporting, for example, the research of musical styles, compositional rules, etc.;

to provide a way to construct databases of music-related material, which includes musical scores as part of the data, in a closely structured manner - supporting, for example, the development of multimedia documents;

to provide a way to organize large numbers of musical scores, which can be retrieved in different ways, including search using the musical contents of the score and search using associated external attributes - supporting, for example, the search of databases of musical pieces to construct presentation shows, radio and television programs, etc.; and

to provide a way to automate the confrontation of two or more musical scores sets in order to identify some kind of similarity - supporting, for instance, the construction of systems aimed at identifying existing copyrights, plagiarisms, etc.

The techniques presented herein allow integration of support for the storage of Musical Scores in an object-oriented database manager, maintaining a syntactic homogeneity with other DBMS resources.

Unlike the usual approach, where the user works with one score at a time, these techniques were developed to permit the user to work with sets of scores. The first benefit of this approach is that the user has improved ways to organize a set of scores and retrieve them, based on their contents. The second benefit is that it allows the development of retrieval methods specific for an intended purpose, and the expansion of the system to fulfill new function does not affect the applications already in use.

The adopted approach also ensures that data is maintained in a consistent way, eliminating possible duplication of information, even when the data comes from different sources, like MIDI files or user inserted complementary information. Two ways to search for the desired scores are defined: based on the values of attributes associated to the scores or based on the contents of the scores. The associated attributes can be extracted from the original MIDI files and used to feed the database, if the files contain that information, or they can be inserted in other ways. The search based on the musical contents of the score is based on algorithms that can be developed separately and integrated to the DBMS as methods of the meta-schema of the OODBMS, allowing the interaction of musicians with the system from the earliest phase of the development of the system. In this sense, the software already developed is a good workbench to develop new algorithms to manipulate scores. It offers an extensive set of elementary tools to aid in the development of new algorithms to analyse, modify, compare, etc. one or more scores. It also offers elaborate tools to manage sets of scores, providing all the basic operations of a database system to store, manipulate, search and retrieve scores.

References

[Bancilhon-92] Bancilhon, F.- **“The O_2 Object-Oriented Database System”**, Proceedings of the 1992 ACM Sigmod - International Conference of Management of Data, San Diego, Califórnia, vol.2, No.1, p.7, June 2-5, 1992.

[Bertino-93] Bertino, E.; Lorenzo, M.- **“Object-Oriented Database Systems”**,International Computer Science Series, Addison-Wesley, 1993.

[Biajiz-96] Biajiz, Mauro - “**Modelling Data Models Using Abstraction Parametrizations**” (in portuguese), PhD Thesis presented to IFSC - University of Sao Paulo - Brazil, September 8, 1996.

[Elmasri-94] Elmasri, R.; Navathe, S.B.; **“Fundamentals of Database Systems”**, Addison Wesley Publishing Company, 1994, 2nd edition.

[Figueiredo-97] Figueiredo, M. B. - “**Representation of Audio Sounds in Databases**” (in portuguese), MSc. Dissertation presented to ICMSC - University of Sao Paulo - Brazil, September 26th , 1997.

[Huber-95] Huber, M.D.- **“The MIDI Manual”**, SAMS, Prentice Hall Computer Publishing, 1995.

[MMA-97] MIDI Manufacturers Association - “**DLS Level 1 Overview**”, in http:/www2.midi.org/mma/dls/dlsoview.html (accessed 02/sept/97).

[MMA-96] MIDI Manufacturers Association - “**The Complete MIDI 1.0 Detailed Specification - Version 96.1**" , MMA MIX Bookshelf, march 1996.

[Pentfold-90] Pentfold, R. A. - **“The Practical MIDI Handbook”**, PC Publishing Co., 1990.

[Stolz-93] Stolz, A.- **“The Sound Blaster Book”**, Abacus, 1993.

[Zand-95] Zand, M.; Collins, V.; Caviness, D.- **“A Survey of Current Object-Oriented Databases”**, Database Advances, vol.26, no.34, pp.14-29, February 1995.