GMQL: A graphical multimedia query language

Zongda Wu\textsuperscript{a}, Guandong Xu\textsuperscript{b}, Yanchun Zhang\textsuperscript{b}, Zhongsheng Cao\textsuperscript{c}, Chenglang Lu\textsuperscript{a}, Zhiwen Hu\textsuperscript{a}

\textsuperscript{a}Oujiang College, Wenzhou University, Wenzhou, 325025, Zhejiang, China
\textsuperscript{b}School of Engineering and Science, Victoria University, PO Box 14428, Vic 8001, Australia
\textsuperscript{c}College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China

Abstract

The rapid increase of multimedia data makes multimedia query more and more important. To better satisfy users’ query requirements, developing a functional multimedia query language is becoming a promising and interesting task. In this paper, we propose a graphical multimedia query language called GMQL, which is developed based on a semi-structured data organization model. In GMQL, we combine the advantages of graphs and texts, making the query language much clear, easy to use and with powerful expressiveness. In this paper, we first present the notations and basic capabilities of GMQL by query examples. Second, we discuss the GMQL query processing techniques. Last, we evaluate and analyze our multimedia query language through the comparison with other existing multimedia query languages. The evaluation results show that, GMQL has powerful expressiveness, and thus is much applicable for multimedia information retrieval.

Keywords: multimedia query language, data model, query processing

1. Introduction

As a powerful tool for specifying users’ multimedia query requirements, a multimedia query language is one of the most essential components in a multimedia database management system. Nowadays, most of published multimedia query languages are text-based, e.g., MQSL\cite{1}, CVQL\cite{2}, BVQL\cite{3}, UMQL\cite{4}, etc. However, such text-based languages are generally difficult for common users to use. As an intuitive solution, a graphical query language or interface may help people carry out multimedia queries. In this paper, we present a semi-structured multimedia data organization model, and based on this model, we design a graphical query language used to satisfy users’ multimedia query requirements. On the one hand, we tend to make the query language clear and concise in syntax expression for providing a user-friendly query environment. On the other hand, we tend to make the query language as powerful on query expressiveness as possible.

The rest of this paper is organized as follows. In the following subsections, we briefly talk about the criteria of a good database query language (Section 1.1), multimedia query requirements (Section 1.2), as well as the contributions of this paper (Section 1.3). In Section 2, we introduce a multimedia data organization model that is the basis of our query language. In Section 3, we introduce our graphical multimedia query language with definitions and examples, and discuss the query processing techniques of the query language. In Section 4, we evaluate our query language through the comparison with other related multimedia query languages. In Section 5, we discuss the characteristic of our query language. Last, we in Section 6 summarize this paper and present the future work.

1.1. Criteria of a Good Database Query Language

As pointed out in \cite{5}, from the view of users, a good database query language should satisfy the following four requirements.

1. \textbf{Expressiveness}: a good query language should be able to express most users’ queries, i.e., it should have powerful expressiveness so as to satisfy users’ query requirements.

2. \textbf{Conciseness}: a good query language should be as clear and concise as possible in syntax expression, so as to help people use the query language.

3. \textbf{Completeness}: a good query language should be able to support not only data extraction but also data manipulation (e.g. INSERT, UPDATE etc.), data definition as well as data control.

4. \textbf{User-friendliness}: most query languages are developed for human users to use; however, most human users are not experts in database systems. Therefore, it is required that a good query language should be easy to learn, easy to write and easy to read.

In order to meet the requirements mentioned above, a graphical solution, i.e., a graphical multimedia query language or interface, is an intuitive approach to building a good multimedia query environment.
1.2. Multimedia Query Requirements

Audio, video, image, graphics, multimedia document etc. are familiar multimedia data types. They are with different forms and with different content information, so they present various query requirements. However, as pointed out in [6], the content information, which should be described by a multimedia query language with good expressiveness, can be divided into the following three levels.

1. **Bottom-level feature information**: they generally can be extracted from multimedia original data using automatic analysis techniques, such as color feature, texture feature and shape feature.

2. **Structure information**: they represent what people sense when looking or listening, namely what are included in multimedia data and what features they contain.

3. **Spatio-temporal information**: they can be regarded as content information with higher-level semantics, such as temporal relationship between video events and spatial directional relationship between image interior objects.

Based on the above three points, we presented sixteen rules, which are shown in Table 1, as the criteria for a multimedia query language with good expressiveness. The rules are compatible with each other, which reflect user’s multimedia query requirements, consequently, supplying a clear requirement for designing a multimedia query language with good expressiveness. In this paper, we tend to make our graphical query language not only clear and concise in syntax form so as to provide a friendly query environment for users, but also accordant to the sixteen rules as much as possible, consequently, with powerful expressiveness.

<table>
<thead>
<tr>
<th>NO</th>
<th>RULE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C01</td>
<td>support to query based on objects’ normal or complex attributes</td>
</tr>
<tr>
<td>C02</td>
<td>support to query based on multimedia bottom-level features</td>
</tr>
<tr>
<td>C03</td>
<td>support to query based on similarity among multimedia objects</td>
</tr>
<tr>
<td>C04</td>
<td>support to query based on semantic notions of objects</td>
</tr>
<tr>
<td>C05</td>
<td>support to query based on containing relationships between objects</td>
</tr>
<tr>
<td>C06</td>
<td>support to query based on context relationships between objects</td>
</tr>
<tr>
<td>C07</td>
<td>support to query based on spatial directional relationships of objects</td>
</tr>
<tr>
<td>C08</td>
<td>support to query based on spatial topological relationships of objects</td>
</tr>
<tr>
<td>C09</td>
<td>support to query based on space-related computation</td>
</tr>
<tr>
<td>C10</td>
<td>support to query based on 3-dimension spatial relationships of objects</td>
</tr>
<tr>
<td>C11</td>
<td>support to query based on temporal relationships between objects</td>
</tr>
<tr>
<td>C12</td>
<td>support to query based on time-related computation</td>
</tr>
<tr>
<td>C13</td>
<td>support fuzzy queries</td>
</tr>
<tr>
<td>C14</td>
<td>support incremental queries</td>
</tr>
<tr>
<td>C15</td>
<td>support query rewrites</td>
</tr>
<tr>
<td>C16</td>
<td>support composite queries</td>
</tr>
</tbody>
</table>

Table 1: Evaluation criteria for a multimedia query language with powerful expressiveness

1.3. Contributions

In this paper, the main work is to design a graphical multimedia query language and discuss its query processing techniques, so the main contributions can be highlighted as follows.

1. **Graphical query interface**: we define a complete set of concise icons used as graphical query interface of our multimedia query language, consequently, making our query language able to well describe users’ various multimedia query requirements, namely, capable of querying based on multimedia feature, structure, and spatio-temporal information.

2. **Query processing**: we introduce the prototype system of our query language, which includes a query interface, a grammar analyzer, a query translator and a display interface, where the query translator is used to map each graphical query into its equivalent text-based query, and then use the text-based query to retrieve data from a multimedia database.

3. **Effectiveness evaluation**: based on the rules given in Table 1, we evaluate our query language through the comparison with existing multimedia query languages, in terms of effectiveness. The evaluation results show that, our query language has powerful expressiveness, and thus is much applicable for querying multimedia information.

2. Multimedia Data Organization Model

In recent years, there have been many multimedia data model proposals for logically modeling complex multime-
dia data types (please see the survey in [7]), all of which can be used to represent various multimedia content information. Hence, in this paper do not discuss multimedia logic modeling methods. And, we assume that multimedia content information have been extracted and stored into a multimedia database, and the content information is organized in a semi-structured form that we call multimedia data organization model.

Below, we present an introduction about the multimedia data organization model. It is different from the relational model and the object-oriented model, including such notions as constructed data type, collection data type, attribute, object, child object and so on. These notions are described as the following definition.

Definition 1. The multimedia data organization model is described as follows,

\[
< \text{data type} > ::= < \text{predefined data type} > \mid \n\]
\[
< \text{collection data type} > \mid < \text{constructed data type} > ;
\]
\[
(\langle \text{constructor} \rangle \mid \langle \text{constructor} \rangle ) \rangle ;
\]
\[
< \text{constructor} > ::= < \text{identifier} > :\langle \text{data type} \rangle 
\]

where the meanings of all construct items are described as follows:

1. Every predefined data type is a subtype of itself and no other data types, such as FLOAT, INTEGER, CHAR and DATE.
2. If \(D\) is a data type, then \(D^*\) is also a data type that is called the collection data type of \(D\). Its instance is a composite value comprising of zero or more elements, each an instance value of \(D\).
3. A constructed data type is a composite data type comprising of several predefined data types, collection data types and other constructed data types, where each item in a constructed data type is called a constructor. Any constructed data type cannot be defined by itself. In other words, it is not allowable that, in a constructed data type \(D\), there is a constructor with the data type \(D\).
4. An instance of a constructed data type is called an object. Every constructor of an object is an attribute, whose value is called an attribute value of the object, and also called child objects of the object if the data type of the constructor is a collection data type or a constructed data type. To simplify presentation, a constructed data type represents both the data type name and the structure of objects belonging to the data type.

For conciseness and effectiveness, our multimedia organization model introduces the notion of collection data type, making the model semi-structured and thus more applicable for expressing complex multimedia content information. This is because the organization structure of most existing multimedia logic data models is also semi-structured, such as MPEG7.

In Figure 1, we present a data organization schema as the query foundation of the examples given in the subsequent sections. In Figure 1, CHAR, INTEGER, STRING, DATE, C_OBJECT and I_OBJECT are predefined data types; and DIRECTOR, MOVIE, VIDEO, CLIP, EVENT and IMAGE are constructed data types. The data organization schema denotes that: 1) each video contains video clips and video events; 2) each video clip contains many salient clip objects; 3) each image contains several salient image inner objects; and 4) the data type of salient object is predefined by system.

Besides reflecting the nested structure, the multimedia data organization model also can specify the constraints of containing relationships among data types, which is the information that cannot be expressed in the traditional relational data model. On the help of the multimedia content information organized in the form according with such data organization model, we begin to develop our graphical multimedia query language in the following section.

3. Our Graphical Multimedia Query Language

Our graphical query language called GMLQ (Graphical Multimedia Query Language) is developed as a visual user-friendly tool for human users to extract information from a multimedia database. To do this, GMLQ should be able to express most multimedia queries clearly and concisely (i.e., without ambiguity), and be simple to draw and easy to read. In this paper, the functionality on data extraction will be discussed; and the other functionalities such as data manipulation, data control and data integration will not be included.

In this section, we first introduce the basic concepts and advanced concepts in GMLQ (in Sections 3.1.1 and
3.1.2). Next, we show that how GMQL is to express various multimedia query operations, mainly including structure selection operation (Section 3.2.1), feature selection operation (Section 3.2.2), spatio-temporal selection operation (Section 3.2.3), join and projection (Section 3.2.4). In Section 3.3, we last discuss the GMQL query processing techniques.

3.1. General Concepts in GMQL

GMQL consists of many graph icons, and all these graph icons are built on the concepts of the multimedia data organization model given in Section 2. Using the GMQL graph icons, we can express various multimedia query operations.

3.1.1. Basic graphical icons

In GMQL, there are the following three main types of graphical icons: data icons, connections and boxes, which are shown in Figure 2. Below, we give a brief introduction on these icons.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>① Rectangle</td>
<td>Represents a group of objects with the same constructed data type. A rectangle is correlative with some attributes, including normal attributes (which are of predefined data type) and complex attributes (whose data types are constructed data type or collection data type). A rectangle has a name identifier (that is unique in a GMQL query), a number (that should be more than one) and a type identifier, which are called rectangle name, rectangle number and rectangle type, respectively.</td>
</tr>
<tr>
<td>② Circle</td>
<td>Represents an attribute, and has to be correlative with a rectangle or a condition box (see Remark 6). When we map a circle into the data organization model in Section 2, a circle is corresponding to an attribute of a constructed data type. A circle is linked with its correlative rectangle or condition box using a line (see Remark 3), and has a name identifier.</td>
</tr>
<tr>
<td>③ Condition Box</td>
<td>Contains one or more rectangles, which is used to express condition constraints that should be satisfied by all the rectangles that it contains. In a GMQL query, there may be several rectangles of the same data type and satisfying the same condition constraints. At this moment, we can use a condition box to express the common condition constraints for all these rectangles uniformly. In GMQL, a condition box is represented using a gray rectangle, which has to be correlative with a conditional expression.</td>
</tr>
<tr>
<td>④ Unidirectional Arrow</td>
<td>Is used to link two rectangles, and can be separated into two types based on whether it is with or without an identifier. If two rectangles are linked by an arrow without an identifier, it denotes a containing relationship between the two rectangles; or else, it denotes a spatio-temporal relationship, and its identifier is called spatio-temporal identifier.</td>
</tr>
<tr>
<td>⑤ Bidirectional Arrow</td>
<td>Is used to link two scene boxes, which denotes the logical relationship between the two scene boxes. A bidirectional arrow has a logic identifier whose value may be ‘AND’ or ‘OR’.</td>
</tr>
<tr>
<td>⑥ Scene Box</td>
<td>Describes a basic GMQL description, which contains a composite graph comprising of many data icons, connections, condition boxes, collection boxes, and even other scene boxes. GMQL uses a rounded rectangle to represent a scene box. A GMQL query generally includes one or more scene boxes.</td>
</tr>
</tbody>
</table>

Figure 2: Basic graphical icons

(1) Data icons: GMQL includes two types of data icons, i.e., rectangle and circle, which are both the basic elements in a GMQL query.

- **Remark 1.** A rectangle represents a group of objects with the same constructed data type. A rectangle is correlative with some attributes, including normal attributes (which are of predefined data type) and complex attributes (whose data types are constructed data type or collection data type). A rectangle has a name identifier (that is unique in a GMQL query), a number (that should be more than one) and a type identifier, which are called rectangle name, rectangle number and rectangle type, respectively.

- **Remark 2.** A circle represents an attribute, and has to be correlative with a rectangle or a condition box (see Remark 6). When we map a circle into the data organization model in Section 2, a circle is corresponding to an attribute of a constructed data type. A circle is linked with its correlative rectangle or condition box using a line (see Remark 3), and has a name identifier.

(2) Connections: GMQL includes three types of connections: line, unidirectional arrow and bidirectional arrow, which are used to link graphical data icons, so as to express semantic relationships within the data icons.

- **Remark 3.** A line is the simplest connection designed to link a circle with its correlative rectangle. In such process, it is required that there actually exist an attribute identical to the circle name in the rectangle type. Moreover, a line also can be used to link two rectangles, denoting a join operation with the two rectangle types; and at this moment, the line should be with a join conditional expression.

- **Remark 4.** A unidirectional arrow is used to link two rectangles, and can be separated into two types based on whether it is with or without an identifier. If two rectangles are linked by an arrow without an identifier, it denotes a containing relationship between the two rectangles; or else, it denotes a spatio-temporal relationship, and its identifier is called spatio-temporal identifier.

- **Remark 5.** A bidirectional arrow is used to link two scene boxes, which denotes the logical relationship between the two scene boxes. A bidirectional arrow has a logic identifier whose value may be ‘AND’ or ‘OR’.

(3) Boxes: GMQL includes three types boxes: condition box, collection box and scene box, which are used as the containers of rectangles, circles and other graph icons.

- **Remark 6.** A condition box contains one or more rectangles, which is used to express condition constraints that should be satisfied by all the rectangles that it contains. In a GMQL query, there may be several rectangles of the same data type and satisfying the same condition constraints. At this moment, we can use a condition box to express the common condition constraints for all these rectangles uniformly. In GMQL, a condition box is represented using a gray rectangle, which has to be correlative with a conditional expression.

- **Remark 7.** A scene box describes a basic GMQL description, so it contains a composite graph comprising of many data icons, connections, condition boxes, collection boxes, and even other scene boxes. GMQL uses a rounded rectangle to represent a scene box. A GMQL query generally includes one or more scene boxes.
3.1.2. Advanced concepts

In the above introduction on GMQL graph icons, we use the following three types of concepts: spatio-temporal identifier, conditional expression and path expression. We below give an introduction on these concepts.

- **Remark 8.** A **collection box** generally contains several condition boxes. A collection box is corresponding with a collection attribute of a rectangle, and thus it is correlative with a path expression. GMQL uses a dashed rectangle to represent a collection box.

- **Remark 9.** A **spatio-temporal identifier** is used in a unidirectional arrow, which includes a spatio-temporal predicate and a spatio-temporal parameter, used to identify the spatio-temporal relationship between two rectangles or condition boxes. About the particular meanings of spatio-temporal identifiers, we will continue to talk in Section 3.2.2.

- **Remark 10.** A **condition identifier** is used in a condition box, denoting a group of condition constraints (they can be divided into feature constraints and normal constraints) that are applied onto all the rectangles that the condition box contains. In a GMQL query, each condition identifier stands for a certain part of query condition constraints.

- **Remark 11.** A **path expression** is used in a collection box to identify a collection attribute of a rectangle. A path expression is in the form of $P.P_1.P_2.....P_n$, where $P$ has to be the name of a rectangle. Let’s take a GMQL query defined on top of the data schema in Figure 1 as an example, if $m$ is the name of a rectangle with the data type MOVIE, then $m$.video.clips is a path expression.

3.2. Basic Query Operations

As traditional database query languages, GMQL is also able to express basic data query operations such as Selection, Projection and Join. However, in order to satisfy users’ various multimedia query requirements, GMQL further extends the selection operation with structure selection, feature selection and spatio-temporal selection, which are corresponding to the three-level multimedia content information mentioned in Section 1.2, respectively. Below, we introduce GMQL how to express these basic query operations.

3.2.1. **Structure selection operation**

A structure selection operation is a query based on multimedia structure information. For example, query images in which there are four inner image objects; and query videos which contain eight video clips, and each of these clips contains four inner clip objects. GMQL implements structure selection operation through declaring several rectangles and the containing relationship among these rectangles, i.e., linking these rectangles using unidirectional arrows. In such process, GMQL generally uses three graph icons: rectangle, unidirectional arc and collection box.

**Query 1.** Find out all the movies, the poster of each of which contains three inner image objects, in which two objects are ‘HORSE’ and the rest one is ‘SUN’.

In Figure 1, we use GMLQ to describe a structure selection operation, where $P$ is path expression; $R_0, R_1, R_2, ..., R_n$ are all rectangle names; $N_0, N_1, N_2, ..., N_n$ are rectangle numbers; and $D_0, D_1, D_2, ..., D_n$ are rectangle types and $D_1, D_2, ..., D_n$ are identical with each other. In Figure 3, the structure selection operation $G$ denotes that: in a GMQL query that contains $G$, if the condition constraints defined over $R_1, R_2, ..., R_n$ are $C_1, C_2, ..., C_n$, respectively, then any object $o$ in $R_0$ satisfies $G$ if and only if there are $N_1 + N_2 + ... + N_n$ child objects expanded from $o$ based on the path expression $P$, where $N_1$ objects satisfy $C_1, N_2$ objects of the rest satisfy $C_2$ and so on.

In a structure selection operation, some child rectangles are declared, and they are all contained by a collection box and are linked to a parent rectangle by unidirectional arrows. By using several nested structure selection operations, we actually define the inner structure for multimedia target data. Besides, in a structure selection, it not only defines the containing relationship between rectangles, but also presents the particular containing number through rectangle number.

![Figure 3: A structure selection operation implemented using GMQL](image-url)

In Figure 3, we use GMQL to describe a structure selection operation, which is implemented structure selection operation through declaring rectangle number.

![Figure 4: Query 1, a structure selection operation](image-url)

In Query 1, we use two child rectangles named by `horse` and `sun`, respectively, to represent two groups of objects, which are both contained in the collection attribute of the rectangle named by $m$; moreover, the two gray rectangles...
are condition boxes which are used to specify the semantic notions for the rectangles horse and sun.

**Query 2.** Find out the movies, the video of each of which contains one video clip satisfying that 1) it contains more than five video frames, and 2) it contains three inner clip objects: two ‘HORSE’ and one ‘SUN’.

In GMQL, it is also allowable for common users to use some structure selection operations to make a nested query specification. In Query 2, the graph has two structure selection operations, where the one is nested by the other, representing two level containing relationship declarations, i.e., video contains video clips and video clip contains clip objects.

### 3.2.2. Feature selection operation

A feature selection operation is used to query multimedia data based on their bottom-level features, semantic notions and other multimedia features. For example, query images similar with a given image on color feature; and query images of festive mood. GMQL implements feature selection operation using condition box, where the feature constraints, which should be satisfied by the rectangles, are presented in the form of conditional expression. Such feature constraints are expressed with the help of a series of feature functions, which are defined by systems or users. The feature functions for familiar multimedia features have been implemented by systems, such as TEXTURE, COLOR, SHAPE and IS; however, it is also allowable for common users to define themselves functions for constructed data types, as long as users abide by function definition rules.

**Query 3.** Find out the movies, the poster of each of which contains three inner image objects: two ‘HORSE’ and one ‘SUN’, where the ‘HORSE’ have more than 75% color feature similarity with the image ‘horse.bmp’, and the ‘SUN’ have more than 75% shape feature similarity with the image ‘sun.bmp’.

In Query 3, we use a structure selection operation to declare two child rectangles named by horse and sun, both contained by m; and then use two condition boxes to specify the semantic notion and bottom-level features for the objects represented in the two rectangles. Moreover, a condition box also can be used to specify the condition constraints among normal attributes (see Query 7).

### 3.2.3. Spatio-temporal selection operation

A spatio-temporal selection operation is used to query multimedia data on spatial and temporal information. For example, query images where there are two inner image objects: one PERSON and one CAR, and the PERSON is located on the left of the CAR. It is one of the most essential requirements for a good multimedia query language to implement the query based on spatio-temporal information.

The temporal relationships between multimedia objects can be expressed using the following seven predicates perfectly, i.e., BEFORE, FINISHES, MEETS, STARTS, DURING, EQUALS and OVERLAPS, which are defined originally by J. F. Allen in [8]. Moreover, in 3-dimension space, the directional or topological relationships between spatial objects also can be expressed on horizontal direction (X direction), vertical direction (Y direction), as well as front and back direction (Z direction), respectively. In this process, each spatial object is seen as a cuboid in 3-dimension space. So, the spatial relationships can be expressed by the above seven predicates. To specify the spatio-temporal relationships between multimedia objects uniformly, we define a set of predicates as follows:

\[ Q_S = \{ q[t] \mid q \in \{ \text{BEFORE, FINISHES,} \]
\[ \text{OVERLAPS, MEETS, DURING, STARTS,} \]
\[ \text{EQUALS} \}, t \in \{ X, Y, Z, T \} \]

, where \( q \) is a predicate and \( t \) is a parameter. On the help of this predicates’ set \( Q_S \), spatio-temporal identifier is defined; and then using unidirectional arrow with spatio-temporal identifier, GMQL defines spatio-temporal selection operation.

**Query 4.** Find out the movies, the video of each of which contains two clips, where the first clip contains more
than sixty-five video frames, and two inner clip objects that have 75% color feature similarity with the image ‘horse.bmp’; the second clip contains more than fifty video frames, and one inner clip object that has 75% shape feature similarity with the image ‘sun.bmp’; and last the second video clip appears before the first one.

Figure 7: Query 4, a spatio-temporal selection operation

In Query 4, we use three structure selection operations to define five rectangles and the containing relationships between them, consequently to describe the structure of multimedia target data; then, we use four collection boxes to present the semantic notion and bottom-level feature constraints for these rectangles; and finally, we use a spatio-temporal arrow to describe the temporal relationship between the rectangles clip1 and clip2. Besides, GMQL also can describe the spatial directional or topological relationships (see Query 5).

Query 5. Find out the movies, the poster of each of which contains three inner image objects: two ‘HORSE’ and one ‘SUN’, and the ‘HORSE’ are located on top of the ‘SUN’.

Figure 8: Query 5, a spatio-temporal selection operation

3.2.4. Projection and join operation

GMQL is also able to implement projection and join operations, with the help of circles and lines. Below, we introduce the implementation of projection and join operations through two query examples.

Query 6. Find out the movies, the poster of each of which contains three inner image objects: two ‘HORSE’ and one ‘SUN’; and the projection attributes of the target movies are ‘name’, ‘video’ and ‘poster’.

Figure 9: Query 6, a projection operation

In Query 6, we use three circles to represent the attributes that need to be projected. In a GMQL query, if there is a rectangle not correlative with any circle, then it means that all attributes of the rectangle should be projected; otherwise, if there is a rectangle correlative with only one circle that is without identifier, it means that no attribute of the rectangle need to be projected.

Query 7. Find out all the ‘Angle Lee’-directed movies, the poster of each of which contains three inner image objects: two ‘HORSE’ and one ‘SUN’.

Figure 10: Query 7, a join operation implemented using GMQL

In Query 7, we use a line with a join condition to express a join operation between the two variables: m and d. If the join condition expression correlative with a line is ‘*’, then it denotes a Cartesian product operation.

3.2.5. Advanced query operation

In the above four subsections, we have given the introductions of how GMQL is to express basic query operations, i.e., structure selection operation, feature selection operation, spatio-temporal operation, projection and join operation. In these query operations, we need to use such GMQL graph icons as rectangle, circle, line, unidirectional arrow, bidirectional arrow, condition box and collection box. We in this part introduce how to use scene box to express some advanced query operations.

In a condition box, its conditional expression is only able to express the condition constraint over the rectangles
that are contained by the condition box; but sometimes, we need to express complex constraints that are correlative with many rectangles with different data types. To solve this, we introduce the concept of scene and implement it using scene box. Generally, a scene box contains data icons, connections, condition boxes, collection boxes, and even other scene boxes; however, the box itself can be contained by a collection box or another scene box.

**Query 8.** Find out the movies satisfying that: 1) the poster of each one of these movies contains three inner image objects: two ‘HORSE’ and one ‘SUN’; or 2) the poster of each one of these movies contains two inner image objects: one ‘PERSON’ and one ‘DOG’.

**Query 9.** Find out the movies satisfying that: 1) the poster of each of these movies contains three inner image objects: two ‘HORSE’ and one ‘SUN’; or 2) the video of each of these movies contains five video frames and one video clip, and the video clip contains three inner clip objects: two ‘PERSON’ and one ‘DOG’.

### 3.3. GMQL Query Processing

As other database query languages, the processing implementation of a GMQL query also includes such steps as grammar check, internal plan generation, internal plan optimization, internal plan execution and so on. In our query processing, we consider to map a GMQL query into other multimedia query languages, consequently, simplifying the design and implementation of our prototype system. In an earlier work [4] of our team, a text-based multimedia query language, which we call UMQL, has been presented. Due to its not user-friendly interface, UMQL is difficult for common users to use, which is also one of the important motivations that we further propose GMQL. However, UMQL allows users to query multimedia data based on structure, feature and spatio-temporal information, so it has approximately equivalent expressiveness with GMQL. In addition, we have also done a series of work on UMQL query processing [6, 9], and designed and implemented a UMQL prototype system, which, given a UMQL query, can extract the interesting targeted data from a multimedia database.

Based on such a consideration, the GMQL prototype system that we use presently includes the following components: a query interface used by common users to draw a graph query; a grammar analyzer used to check the grammar validity of a graph query; a query translator used to translate a graph query into its equivalent text query; and a display interface used to show the query results. The framework of our system is shown in Figure 13. As we see from Figure 13, our system is built on top of the UMQL prototype system, i.e., it doesn’t involve the processing of query plan generation, optimization, execution etc., all of which are implemented with the help of the UMQL prototype system, so the main work of our system includes a grammar analyzer and a query translator.

1) A **query interface** is a visual freedom environment, in which users are allowed to express various query operations, e.g., structure selection operation, spatio-temporal selection operation etc., making users able to describe their multimedia query requirements in a natural way. To draw a basic GMQL query, we need the following several steps: first, to define several rectangles and the containing relationships between these rectangles (i.e., structure selection operation); next, to describe the condition constraints for these rectangles (i.e., feature selection operation); and last, to express the spatio-temporal relationships (i.e., spatio-temporal selection operation). Furthermore, to draw the
whole GMQL query, we also need to define several scene boxes (each being a basic query), the logic relationships among these scenes, as well as projection and join operations.

Figure 13: Framework of GMQL prototype system

Figure 14: Translation from a GMQL query to a UMQL query

(2) A grammar analyzer is used to check the grammar constraints of graph icons in a GMQL query. In GMQL, there are two types of grammar rules: a) syntax rules that present the constraints among various graph icons, e.g., it is not allowable that there is a spatio-temporal relationship between two scenes, so it is not allowable that there is an unidirectional arrow between two scene boxes; b) semantic rules that consider the particular meaning of each graph icon, e.g., it is not allowable that there is a rectangle type that is not defined in the data schema. GMQL includes a complete set of complex syntax and semantic rules. Due to the space limitation, we here cannot present these rules, but we will discuss the design and implementation of the GMQL grammar analyzer in another paper, where we will detail these grammar rules.

(3) A query translator is used to generate an equivalent UMQL query for a GMQL query. Here, we use an example shown as Figure 14 to briefly illustrate such translation process. UMQL does not have the concept of scene. To solve this problem, we first translate each scene in a GMQL query into a UMQL query; then we use each UMQL query to retrieve data from a multimedia database, respectively; and last, we refilter these data based on the logical relationships among scenes. Obviously, such a way would lead to decreased query execution performance. To overcome this problem, our future work will consider to generate an internal query plan for a GMQL query immediately, instead of a UMQL query.

(4) A display interface is used to show the query results which are returned by a GMQL query. Presently, the query results are shown in the form of many tables.

4. Related Work and Effectiveness Evaluation

Before GMQL, there are many query languages or interfaces designed for querying multimedia content information. In this section, we first give an introduction about some existing typical multimedia query languages; and then we use the sixteen rules mentioned in Section 1.2 to evaluate these query languages, and make a comparison in terms of expressiveness between these query languages and GMQL.

MQSL[1], CVQL[2], BVQL[3] and UMQL[4] are all multimedia query languages proposed on the level of multimedia database management systems, which are able to query some familiar multimedia data types. In MQSL[1], there are such conditional expressions as content condition, temporal condition and spatial condition. Using these conditional expressions, MQSL can well describe the spatio-temporal relationships between the multimedia objects in documents. However, MQSL cannot sufficiently describe the inner structure of multimedia objects. CVQL[2] is a content-based video query language. It implements video
queries through using some query predicates to describe the spatial and temporal relationships between video inner objects. However, it cannot support topological spatial queries, because it treats each video inner object as a point in two-dimension space. BVQL[3] is a SQL-like video query language, which can implement object query, spatial query, object track query, temporal query and bottom-level query; however, these implementations are built on a large number of functions, making BVQL difficult to use for users. UMQL[4] is also a SQL-like query language, which extends the WHERE clause to introduce feature condition, structure condition and spatio-temporal condition, and consequently can well meet users’ query requirements.

The above query languages are all text-based, and thus they are generally difficult for common users to use. To solve this problem, many graphical multimedia query languages or interfaces were designed, such as MQuery[10], VISUAL[11], WS-QBE[12] and so on. MQuery[10] is proposed on a multimedia data model that combines the entity relationship model and the object oriented model. MQuery can express questions over multimedia, timeline and simulation data using a single set of related query constructs. VISUAL[11] is designed for applications where the data has spatial properties, also including sequences and complex objects. VISUAL borrows the example-element concept of Query-by-Example to formulate query objects. Moreover, it uses such concepts as hierarchical subqueries, and internal and external queries. However, VISUAL cannot express temporal relationships. WS-QBE[12] is the combination of a schema to weight query terms and concepts from fuzzy logic and Query-by-Example. In a WS-QBE query, the basic elements are many query tables such as weight tables, temporal tables and spatial tables.

In addition, there are also other related query languages [13, 17, 20] or systems [14, 15, 18], as well as related query processing techniques [16, 19]. Although being able to support to query multimedia content information to a certain extent, they are all not designed aiming to multimedia databases, consequently, making them not suitable to be applied into querying multimedia data.

Now, let us use the rules mentioned in Section 1.2 to evaluate the above multimedia query languages. The evaluation results are shown as Table 2. In Table 2, we use C01-C16 to represent all the rules in Section 1.2, and use ‘YES’, ‘NO’ and ‘WEAK’ to represent support, non-support and weak support, respectively. As we see from Table 2, GMQL can satisfy most evaluation rules, and has more powerful effectiveness than other multimedia query languages.

<table>
<thead>
<tr>
<th>RULE</th>
<th>MQSL</th>
<th>CVQL</th>
<th>BVQL</th>
<th>UMQL</th>
<th>MQuery</th>
<th>VISUAL</th>
<th>WS-QBE</th>
<th>GMQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>C01</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>C02</td>
<td>YES</td>
<td>NO</td>
<td>WEAK</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>WEAK</td>
<td>YES</td>
</tr>
<tr>
<td>C03</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>C04</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>C05</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>WEAK</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>C06</td>
<td>NO</td>
<td>NO</td>
<td>WEAK</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>C07</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>C08</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>WEAK</td>
<td>YES</td>
</tr>
<tr>
<td>C09</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>C10</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>C11</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>C12</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>C13</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>C14</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>C15</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>C16</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>WEAK</td>
<td>WEAK</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Table 2: Comparison on expressiveness between GMQL and other multimedia query languages.

5. Discussion

In the above sections, we have detailed GMQL, including its data model (Section 2), graphical icons (Section 3.1), as well as how to describe each type of query operation (Section 3.2), and also have evaluated GMQL in terms of query effectiveness (Section 4). The results show that, GMQL not only has good query expressiveness, but also have relatively concise syntax, consequently, providing a powerful and friendly query tool for querying multimedia information. Below, we briefly summarize the characteristic of GMQL.

(1) User-friendly query interface. GMQL consists of 8 graphical icons, including 2 data icons, 3 lines and 3 boxes, which are clear and concise, so easier to write and easier to read to a certain extent, than a text-based query
language, consequently, supplying a visual query environment for common users, i.e., making users able to well describe their multimedia query requirements in a natural way.

(2) Powerful expressiveness. As mentioned in Section 4, compared with existing multimedia query languages, GMQL has more powerful expressiveness, which can support feature selection operation, structure selection operation, spatio-temporal selection operation etc., consequently, well satisfying the three levels of multimedia query requirements given in Section 1.2 and applicable for querying multimedia information.

(3) General-purpose. GMQL can be applied to various multimedia data types, due to the common query requirements (Section 1.2). In addition, it is also allowable for users to add a new multimedia data type. When introducing a new multimedia data type, a user only needs to construct a corresponding constructed data type and define a set of feature functions for the data type if necessary.

(4) Semi-structured model. We introduce a semi-structured data model, used as the basis of our multimedia query language, which extends the relation model to introduce the notion of collection data type, making it not only can reflect out the nested relationship of data types, but also can describe the constraints of containing relationship among data types, consequently, making GMQL built based on the data model more general.

However, there still exist several disadvantages on our query language, mainly including the following three points.

(1) Formalization definition. In this paper, we only present how to implement each type of query operation by examples, without the accurate definition on these query operations as well as their related graphical icons, which is disadvantageous for our query language to popularize its application. (2) Completeness. In this paper, actually, we only describe a data extraction language, without the definition on data manipulation etc. (3) Query processing. At present, we implement the GMQL prototype system with the help of the existing system, where we need to first map each graph query into its equivalent text query and then use the text query to retrieve data from a multimedia database, which obviously would lead to a decreased query performance (due to the additional expense on the translation from graph query to text query).

6. Conclusion and Future Work

GMQL (Graphical Multimedia Query Language) is a powerful visual query language for querying multimedia contents, which, based on a semi-structured organization model, presents a user-friendly query environment that allows common users to construct any view graphs in a natural way. GMQL combines the advantages of graphs and texts together, which not only keeps the graphical part clear but also makes the textual part easily understood.

The future research work on GMQL is given as follows. First, we need to present the formalization definition on our query language, including each type of query operation as well as each type of basic icon. Second, we need to extend our query language, including data manipulation, data integration, as well as view maintenance, so as to exploit the power of graphical way for querying multimedia information. Third, we need to study the query processing techniques suitable for our query language, instead of mapping a graph query expression into a text query expression.

7. Acknowledgements

We thank anonymous reviewers for their valuable comments. This work is supported by the National Science Foundation of the Zhejiang Province of China under grant Y1100137, the Educational Research Program of the Zhejiang Province of China under grant Y201016197, and the Science and Technology Program of the Wenzhou City of China under grant 2009G0339 and S20100055.

References


