International Journal of Database Theory and Application
Vol.8, No.3 (2015), pp.235-248
http://dx.doi.org/10.14257/ijdta.2015.8.3.20

SVQL: A SQL Extended Query Language for Video Databases

Chenglang Lu[1], Mingyong Liu[1] and Zongda Wu[2,*]

[1]Northwestern Polytechnical University, Xi'an 710072, Shanxi Province, China
[2]Oujiang College, Wenzhou University, Wenzhou 325035, Zhejiang Province, China
zongda1983@163.com

Abstract

With the rapid increasing of video data, video queries are becoming increasingly important. To better describe users' video query requirements, developing a functional video query language has become a promising and interesting task. In this paper, we present a novel query language called SVQL for video databases, which is developed based on an extension of the traditional database query language SQL. In SVQL, we remain the clear and concise grammatical framework of SQL, thereby making SVQL easy to learn and use for traditional users, i.e., making SVQL with a user-friendly interface. Moreover, we extend the WHERE clause of SQL to introduce new conditional expressions, such as variable declaration, structure specification, feature specification and spatial-temporal specification, thereby, making SVQL with powerful expressiveness. In this paper, we first present the formal definitions of SVQL and illustrate its basic query capabilities using examples. Then, we discuss the SVQL query processing techniques. Finally, we evaluate SVQL through the comparison with other existing video query languages, and the evaluation results demonstrate the practicality and effectiveness of our proposed query language for video databases

Keywords: *Video database; video query language; query processing*

1. Introduction

In the past decades, with the development of the Internet and the availability of video capturing devices, the amount of video data has increased dramatically, which demands support for a video database to facilitate video data storage and retrieval. Now, while there are lots of works dedicated to detection, tracking and recognition for video entities [1], few works have been done for accessing video databases.

A database query language is a powerful tool for database users to describe query requirements, and thus it is one of the most essential components in a database management system [2]. Despite the great success of traditional database query languages (*e.g.*, SQL [3], OQL [4], *etc.*) over the past decades, obviously, all the languages cannot be applicable immediately to video databases, because various kinds of semantic information contained in video data (*e.g.*, the spatial-temporal relationships among video entities) present many new specification requirements for video query languages. While there have been many multimedia query language proposals, *e.g.*, CVQL [5], BVQL [6], SRQL [7], UMQL [8], GMQL [9] *etc.*, some of them have complex grammatical forms, which are different with those of traditional database languages, thereby, making them difficult for traditional users to use. Moreover, most of the languages have relatively weak expressiveness on video queries, thereby, making them unable to meet the diversified video query requirements (see Section 5 for detailed analysis on existing video query languages).

[*] Corresponding Author

In this paper, we present a structured video query language (called SVQL) for video databases. SVQL in syntax form is an extension of the traditional database query language SQL, so as to facilitate traditional database users to learn and use. Moreover, we make the query language as clear and concise in syntax expression as possible to provide a user-friendly interface, and make the query language as powerful on video query expressiveness as possible.

The rest of this paper is organized as follows. In the following two subsections, we will briefly talk about the criteria of a good database query language (in Section 1.1) and video query requirements (in Section 1.2). In Section 2, we present the data model that our video query language is developed based on. In Sections 3 and 4, we introduce our proposed video query language with definitions and examples, and discuss the processing techniques for the query language. In Section 5, we review related work and evaluate our query language through the comparison with existing video query languages. In Section 6, we summarize this paper and present the future work.

1.1. Criteria of a Good Database Query Language

As pointed out in [9, 10], from the view of users, a good database query language should satisfy the following four requirements.

- **Expressiveness:** a good query language should be able to describe most user queries, i.e., it should have powerful expressiveness so as to satisfy user query requirements.
- **Conciseness:** a good query language should be clear, concise and easy to use in syntax expression, so as to help people use the query language.
- **Completeness:** a good query language should support not only data extraction but also data manipulation (*e.g.* INSERT, UPDATE *etc*.), data definition and data control.
- **User-friendliness:** most database query languages are developed for human users to use, and most users are not experts in database systems, which requires that a good query language should be easy to learn, easy to write and easy to read.

Based on the observation that common users have accustomed to the traditional structured query language SQL, to satisfy the above requirements (especially, the requirement for user-friendliness), developing a SQL-extended video query language is an intuitive and interesting way to create a good interface for querying video databases.

1.2. Video Query Requirements

Development in video technology and occurrence of new video-related applications has led to an enormous growth in the volume of video data. Compared with traditional data, video data has more complex structure, which contains more rich semantic information, resulting in more diversified query requirements. As pointed out in [9, 12], the content information conveyed by video data can be divided into the following three levels.

- **Low-level feature information:** It generally can be extracted from original video data using automatic analysis techniques, such as color feature, texture feature and shape feature.
- **Syntactic information:** It describes what contained in video data, including video entities, their spatial-temporal position, and the spatial-temporal relationships between them.
- **Semantic information:** It can be regarded as content information with higher level semantics, which describes what is happening in video that can be perceived by human users, such as video objects, video events, and the spatial-temporal relationships between them.

Table 1. The Evaluation Criteria for a Video Query Language with Powerful Expressiveness

No	Evaluation Rule
R1	Support for object and user defined object attributes
R2	Support for event and user defined event attributes
R3	Support of distinguishing activity (event type) and event
R4	Support for special relationships between objects and events
R5	Support for user defined relationships between objects or events
R6	Support for uncertainty
R7	Support for queries based on low-level or syntactic information
R8	Support for queries based on temporal relationships of objects
R9	Support for queries based on spatial directional relationships of objects
R10	Support for queries based on spatial topological relationships of objects
R11	Support for queries based on 3-dimension spatial relationships of objects
R12	Support for detecting logical errors in semantic information
R13	Support for similarity queries among objects or events
R14	Support for composite queries
R15	Support for incremental queries
R16	Support for query rewrites

In [10,11], the authors have presented a set of rules for evaluating video semantic models. Based on the evaluation rules for video models, combining with the three levels of video content information mentioned above, we present 16 rules that are shown as Table 1, as the criteria for the expressiveness of a video database query language. The 16 rules are compatible with each other, which reflect users' video query requirements, therefore, supplying a clear requirement for us to develop a video query language with good expressiveness. In this paper, we will make our video query language not only as clear and concise in syntax form as possible, so as to provide a friendly interface for users, but also accordant to the 16 rules as much as possible, so as to have powerful expressiveness.

2. Video Model

In recent years, there have been many video model proposals (see the survey [10]) for modeling rich video content information. Thus, in this section, we only focus on which types of video entities and which attributes of video entities have been modeled, while not discussing detailed video models. We assume that video entities and their attributes have been extracted from original video data, and stored into a video database.

From the users' point of view, the video entities which should be represented by our video model include (1) two types of semantic entities: video events (EVENT) and video objects (OBJECT), and (2) two types of physical entities: video clips (CLIP) and video frames (FRAME). Generally, the two types of physical entities can be extracted from original video using automatic analysis techniques. Thus, we here mainly focus on semantic entities. Tables 2 to 4 present the two types of semantic entities and their key attributes, as well as the relationships among entities. All the information can be obtained using automatic or semi-automatic analysis techniques (*e.g.*, that in [7]).

Table 2. Video Objects (OBJECT)

Name	Data Type	Explanation
ID	Varchar	the unique identifier of a video object
Label	Varchar	the label of a video object in its media
TimePoint	Datetime	the time point of a video object in its media
TimePoint2	DateTime	the time point of a video object in the real world
TimeDuration	BigInteger	the time interval of a video object in its media
SpaceRegion	Varchar	the label of space region that a video object embodies
DocID	Integer	the identifier of the document that an object belongs to
MediaID	Varchar	the identifier of the video clip that an object belongs to

The video query language proposed in this paper is developed based on the video model, i.e., we assume that all these information have been extracted from original video and stored into the video model (or the video database).

Table 3. Video Events (EVENT)

Name	Data Type	Explanation
ID	Varchar	the unique identifier of a video event
Label	Varchar	the label of a video event in its media
PlaceLabel	Varchar	the label of the location where an event takes place
TimePoint	Datetime	the time point of a video event in its media
TimePoint2	DateTime	the time point of a video event in the real world
TimeDuration	BigInteger	the time interval of a video event in its media
Region	Varchar	the label of region that the place of an event belongs to
DocID	Integer	the identifier of the document that an event belongs to
MediaID	Varchar	the identifier of the video clip that an event belongs to

Table 4. Relationships between Events, Objects or Objects and Events

Name	Data Type	Explanation
DocID	Integer	the identifier of the document that a relation belongs to
SourceID	Varchar	the identifier of the source of a relation
TargetID	Varchar	the identifier of the target of a relation
RelationName	Varchar	it describes the relation between source and target
RelationType	Varchar	it describes the relation type between source and target

3. SVQL: A Structured Video Query Language

The video query language proposed in this paper is called SVQL, which is developed based on an extension of the traditional structure query language SQL. SVQL is an orthogonal expression language, in the sense that all the operators can be composed with each other as long as the types of the operands are correct. It extends SQL by introducing some new elements such as video entities, video variables, conditional expressions *etc*. It has one basic statement for retrieving information, whose basic syntax framework is described as follows:

```
SELECT [DISTINCT] <projection-items>
FROM <tables>
[WHERE <search-condition>]
[GROUP BY <group-items> [HAVING <search-condition>]]
[ORDER BY <order-items> [ASC | DESC]];
<search-condition>::= <normal-condition>
[AND | OR [NOT] <search-condition>]
| <variable-declaration>
```

```
[AND | OR [NOT] <search-condition>]
| <structure-specification>
[AND | OR [NOT] <search-condition>]
| <feature-specification>
[AND | OR [NOT] <search-condition>]
| <spatial-temporal-specification>
[AND | OR [NOT] <search-condition>].
```

In the above framework, <projection-items> is a list of attribute names whose values are to be retrieved by a query. In the FROM clause, <table> lists all the tables which a query is related to. In the WHERE clause, which is the most important component of a query, a search conditional expression is used to identify the target tuples to be retrieved by the query.

It can be observed that the overall syntax framework of SVQL is identical to that of SQL, i.e., both using the SELECT-FROM-WHERE statement. However, the main changes for SVQL are in the WHERE clause, i.e., on the basis of <normal-condition> that is originally contained by SQL, it introduces 4 new conditional expressions <variable-declaration>, <structure-specification>, <feature-specification> and <spatial-temporal-specification>, so as to specify video query requirements. In the following subsections, we will introduce the 4 conditional expressions, respectively.

3.1. Variable Declaration

Generally, a video query is implemented by describing the inner structure of target videos, specifically including types of video entities (such as objects, events, clips and frames), as well as various features and relationships over the video entities. To do these, first of all, we need to define the types for the video entities. For this purpose, a <variable-declaration> expression is introduced, i.e., which is used to define several types of video entities contained in target videos.

Definition 1. The formalization of a <variable-declaration> expression is described as,

- "V1, V2, …, Vn : Type" is a basic <variable-declaration> item, where "V1, V2, …, Vn" are user-defined variables, whose names should be unique identifiers of characters; and "Type" denotes the type of all the variables, whose value can be EVENT, OBJCECT, CLIP or FRAME.
- "I1, I2, …, Im WITH Path" is a basic <variable-declaration> expression, where "I1, I2, …, Im" are <variable-declaration> items; and "Path" denotes an attribute of video type.
- If C1 and C2 are <variable-declaration> expressions, then so are C1 AND C2, C1 OR C2, NOT C1 and (C1).

In the above definition we only present the formalization of a <variable-declaration> expression, without detailing the detailed meaning of the expression. Below, we use a simple example to illustrate its meaning.

Example 1. Find movies (query movie names and videos) that contain at least 3 video entities, where the 2 ones should be video objects, and the rest one should be a video event.

```
SELECT m.name, m.video FROM MOVIE m
WHERE obj1, obj2: OBJECT, evn: EVENT
WITH m.video.  --a <variable-declaration>
```

In Example 1, "MOVIE" denotes the name of a table that contains some attributes, *e.g.*, "name" of type of character strings, "video" of video type *etc*. In Example 1, in the <variable-declaration> expression, we use "obj1, obj2: OBJECT" (that is a basic <variable-declaration> item) to declare two variables "obj1" and "obj2" of the type

OBJECT, and we use "evn: EVENT" to declare a variable "evn" of the type EVENT. All the three variables are contained in the attribute "m.video".

Given any movie in the video database, if its video attribute contains not less than two video objects and not less than one video event, then it should be a target movie. It can be seen that <variable-declaration> is a very weak conditional constraint. Generally, <variable-declaration> should be used together with the other three types of conditional expressions, so as to further specify the conditional constraints that each variable defined in <variable-declaration> should meet.

3.2. Feature Specification

In videos, each type of video entities has many kinds of specific features, *e.g.*, the category features for semantic entities, the general features for physical entities *etc*. A <feature-specification> expression is used to specify the features for video entities. Specifically speaking, the feature types of video entities include general features (*e.g.*, the color feature for a video frame image), category features (*e.g.*, the category of a video object) and spatial-temporal features (*e.g.*, the timepoint of an entity in its video). The spatial-temporal features will be discussed in Section 3.4. In this subsection, we only discuss the general features and the category features.

Definition 2. The formalization of a <feature-specification> expression is described as,

- "V1, V2, …, Vn IS Label" is a category <feature-specification> expression, where "V1, V2, …, Vn" are variables defined in a <variable-declaration> expression in the same query; and "Label" is a string of characters denoting the category of all the variables.
- A general <feature-specification> expression is implemented based on feature functions that are defined by the system or users in advance. The functions for familiar multimedia features have been implemented by the system, *e.g.*, texture, color and shape for frame images. It is also allowable for users to define themselves feature functions.
- If C1 and C2 are <feature-specification> expressions, so are C1 AND C2, C1 OR C2, NOT C1 and (C1).

As shown in Definition 2, a <feature-specification> expression is conducted over some variables defined in <variable-declaration> items of the same query. A <feature-specification> expression is used to describe the feature constraints that each variable should satisfy.

Example 2. Find those movies, each satisfying the following conditions: (1) there are three video objects, two 'HORSE' and one 'SUN', in its video; (2) the color feature of the 'HORSE' is similar (similarity value is more than 0.75) to that of the given image, 'horse.bmp'; and (3) the texture feature of the 'SUN' is similar to that of the given image, 'sun.bmp'.

```
SELECT m.name, m.video FROM MOVIE m
WHERE horse1, horse2, sun:OBJECT WITH m.video
AND (horse1, horse2 IS 'HORSE')
AND (sun IS 'SUN')  --a category specification
AND TEXTURE(sun, 'sun.bmp') > 0.75
--a general <feature-specification>
AND COLOR (horse1, 'horse.bmp') > 0.75
AND COLOR(horse2, 'horse.bmp') > 0.75.
```

In Example 2, we use the <variable-declaration> expression to declare three variables "horse1", "horse2" and "sun", all of type of video objects, and then use the <feature-specification> expression to present the category feature constraints and bottom-level feature constraints for the three variables. In Example 2, the constants 'HORSE' and

‘SUN’ denote two categories of video objects, and the constants ‘horse.bmp’ and ‘sun.bmp’ are used to denote two given images.

3.3. Structure Specification

In the same video, there may be containment relationships among video entities, *e.g.*, a video object named by “Michael Jordan” is contained in a video event named by “Slam Dunk”. Hence, the support for querying videos based on their inner structure information (namely the containment relationships between user-declared variables) is important for video databases. Thus, we introduce a <structure-specification> expression for supporting structure queries.

Definition 3. The formalization of a <structure-specification> expression is described as,

- “V1, V2, …, Vn IN V0” is a basic <structure-specification> expression, where “V0, V1, V2, …, Vn” are variables defined in a <variable-declaration> expression.
- If C1 and C2 are <structure-specification> expressions, then so are C1 AND C2, C1 OR C2, NOT C1 and (C1).

In a basic <structure-specification> expression “V1, V2, …, Vn IN V0”, “V1, V2, …, Vn” are called child variables, and “V0” is called a parent variable. Moreover, it is required that the containment relationships should exist between the child variables and the parent variable, *e.g.*, a video object cannot be contained in another video object. However, such surface semantic errors can be checked out by the SVQL grammar analyzer.

Example 3. Find movies directed by ‘Ang Lee’, each of whose videos satisfies (1) there is a video event that belongs to the category ‘EMBRACE’, and (2) there are two video objects that belong to the category ‘PERSON’, contained in the video event.

```
SELECT m.name, m.video, d.name
FROM MOVIE m, PERSON d
WHERE d.name = ‘Ang Lee’ AND d.id=m.director   --a traditional <table-join> expression
AND embr : EVENT WITH m.video  --a <variable-declaration> expression
AND person1, person2 : OBJECT WITH m.video
AND embr IS ‘EMBRACE’
AND person1, person2 IS ‘PERSON’ --a <feature-specification>
AND person1, person2 IN embr.  --a <structure-specification> expression
```

In Example 3, we declare three variables named after “person1”, “person2” and “embr”, respectively, then describe their category features, and last we use a <structure-specification> expression to describe the containment relationships among the variables, i.e., the video objects “person1” and “person2” are contained in the video event “embr”.

3.4. Spatial-temporal Specification

Video spatial-temporal queries denote to query videos based on the spatial-temporal information contained in video entities. For example, query some videos in each of which there are two video objects: a ‘PERSON’ and a ‘DOG’, where the ‘PERSON’ is located on the left of the ‘DOG’. Since the spatial-temporal information widely exists in video entities, it is one of the most important requirements for a powerful video query language to support spatial-temporal queries.

From Table 1, we see that a good video query language should support spatial-temporal queries of various kinds, *e.g.*, spatial directional queries, spatial topological queries, 3-dimenstion spatial queries, temporal queries etc. To better support spatial-temporal queries, we introduce two types of spatial-temporal operators: one-variable operators and two-variable operators.

We design two one-variable operators, i.e., BeginOf and EndOf, which both need to follow along with a parameter: X, Y, Z or T. “BeginOf[T] V” returns the start timepoint of a variable or time constant named after “V”, and “EndOf[T] V” returns the end timepoint. We use a minimum bounding box (MBR) to describe the space region for a video object. Thus “BeginOf[X] V” returns the lower boundary of the MBR projection of the variable “V” over the X direction, and “EndOf[X] V” returns the upper boundary. “BeginOf[Y] V”, “BeginOf[Z] V”, “EndOf[Y] V” and “EndOf[Z] V” are also similar.

We also design four two-variable operators: BEFORE, AFTER, OVERLAPS and DURING, based on the one-variable operators. Their meanings are given as follows.

- “V1 BEFORE [T] V2” <==> “EndOf [T] V1 < BeginOf [T] V2”.
- “V1 AFTER [T] V2” <==> “BeginOf [T] V1 > EndOf [T] V2”.
- “V1 DURING [T] V2” <==> “EndOf [T] V1 < EndOf [T] V2 AND BeginOf [T] V1 > BeginOf [T] V2”.
- “V1 OVERLAPS[T] V2” <==> “BeginOf[T] V2 < EndOf[T] V1 AND BeginOf[T] V2 > BeginOf[T] V1 OR EndOf[T] V2 < EndOf[T] V1 AND EndOf[T] V2 > BeginOf[T] V1”.

Besides, the situations for the four two-variable operators with another parameter (i.e., X, Y or Z) are similar.

It should be noted that the two operands of a spatial-temporal operator should have the same or compatible types; and for video entities of type FRAME, CLIP or EVENT, they can only support temporal queries, while not supporting spatial queries.

Definition 4. The formalization of a <spatial-temporal specification> expression is described as follows,

- If “V1” and “V2” are variables or constants, both having spatial-temporal features, then “V1 SP[OP] V2” is a basic <spatial-temporal specification> expression, where “SP” is a two-variable spatial-temporal operator, and “OP” can be X, Y, Z or T.
- If “V1” is a variable or constant of spatial-temporal feature information, then “SP[OP] V1” is a basic <spatial-temporal specification> expression, where “SP” is an one-variable spatial-temporal operator, and “OP” can be X, Y, Z or T.
- If C1 and C2 are <spatial-temporal specification> expressions, then so are C1 AND C2, C1 OR C2, NOT C1 and (C1).

Example 4. Find movies, each satisfying the condition as follows: (1) there are two video clips in each of these movies, where the first clip contains more than one hundred frames; (2) there is one video object in the first video clip, which belongs to the category ‘HORSE’; (3) there is one video object in the second video clip, which belongs to the category ‘SUN’; and (4) the second video clip appears before the first one.

```
SELECT m.name, m.video FROM MOVIE m
WHERE clip1,clip2:CLIP,horse,sun:OBJECT WITH m.video --a <variable-declaration>
AND horse IN clip1 AND sun IN clip2 --a <structure-specification>
AND FRAME(clip1) > 100 --a general <feature-specification>
AND horse IS ‘HORSE’ AND sun IS ‘SUN’ --a category <feature-specification>
AND clip2 BEFORE[T] clip1. --a <spatial-temporal specification> expression
```

In Example 4, we first declare four variables (clip1, clip2, horse and sun), and the containment relationships among them, then we use the <feature-specification> expression to describe the categories of the variables, and last we use the <spatial-temporal-specification> expression to describe the temporal constraint. Besides, SVQL can also describe the spatial directional or topological constraints among variables (see Example 5).

Example 5. Find movies, each of which contains one video frame satisfying the conditions: (1) there are two video objects, one 'BALL' and one 'GOAL', in the frame; and (2) the 'BALL' is located inside the 'GOAL'.

```
SELECT m.name, m.video FROM MOVIE m
WHERE ball, goal : OBJECT, frm : FRAME WITH m.video
AND ball, goal IN frm AND ball IS 'BALL'
AND goal IS 'GOAL' AND ball DURING[X, Y, Z] goal.
    --a <spatial-temporal specification> expression
```

In Example 5, the spatial directional constraint, "the ball located inside the goal", is equivalently translated into the spatial projected interval constraints on the X direction, Y direction and Z direction. Besides, "DURING[X, Y, Z]" is the abbreviation of three spatial specification items "DURING[X]", "DURING[Y]" and "DURING[Z]".

4. Query Processing

As other database query languages, the query processing implementation for a SVQL query also includes such steps as grammar check, query plan generation, query optimization, query execution and so on. In order to simplify the design and implementation of a SVQL query processing prototype system, we consider using a RDBMS (Relation Database Management System) to construct the video database, and then mapping each SVQL query into its corresponding SQL query. Based on such a consideration, the SVQL query processing prototype system that we use presently mainly includes the following four components: (1) a query interface used for users to write a SVQL video query; (2) a grammar analyzer used to check the syntax and semantic validity for a SVQL query; (3) a query translator used to translate a SVQL query into its corresponding SQL query; and (4) a display interface used to show the query results.

The main framework of our prototype system is shown in `Figure 1. It should be pointed out that a complete SVQL query processing implementation should include the following two components: (1) a query optimizer, used to optimize a query plan produced by an antecedent plan generator, so as to minimize the execution cost of the query plan; and (2) a query executor, used to interpret and execute the optimized query plan. However, as shown in Figure 1, instead of the query plan generator, we use the query translator to translate a SVQL query into a SQL query, and then, we submit the SQL query to the database immediately. Thus, our system is built on top of a RDBMS, and the processing of query plan generation, optimization, and execution are implemented with the help of the RDBMS. Below, we briefly introduce the functions for the components in Figure 1.

A query interface is a graphic environment, in which users are allowed to express their query requirements using SVQL. It can also help users to generate basic SVQL syntax framework, i.e., SELECT-FROM-WHERE.

A grammar analyzer is used to check the grammar constraints for a SVQL query. In SVQL, there are two types of grammar rules: (a) syntax rules that describe the formal grammatical characteristics of SVQL, and can be given using Backus-Naur form (EBNF); and (b) semantic rules that present the logical grammatical characteristics of SVQL, and are used to check the logical validities of variables in a SVQL query. SVQL includes a complete grammar model and a set of complex grammar rules, which will be discussed in another paper.

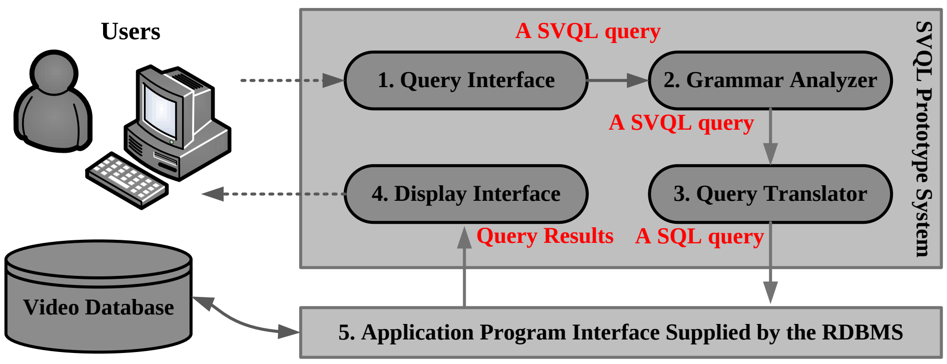

Figure 1. The Framework of Our SVQL Query Processing System

A query translator is used to generate an equivalent SQL query for a SVQL query. In our system, we use several relation tables (as shown in Tables 2-4) that are related to each other to store video information. This requires that the translator should map the video operations in a SVQL query into a SQL query over these tables. Obviously, such way would decrease the query execution performance. To overcome this problem, our future work will construct and optimize the query plan for a SVQL query immediately, instead of a SQL query.

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

5. Evaluation Comparison

Before GVQL, there have been many multimedia query languages. In this section, we present an introduction about some typical query languages that are designed for querying video data or other multimedia data. Then, we use the 16 evaluation rules in Section 1.2 to evaluate these languages, so as to compare GVQL with these languages in terms of query expressiveness.

Table 5. Comparison on Expressiveness between SVQL and other Query Languages

Rule	CVQL	BVQL	SRQL	UMQL	MQuery	VISUAL	WS-QBE	GMQL	SVQL
R1	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
R2	No	No	Weak	Yes	No	No	No	Yes	Yes
R3	No	Yes	Weak	No	No	No	No	No	Yes
R4	No	No	Yes	Yes	No	No	No	Yes	Yes
R5	No	No	Yes	Yes	Weak	No	No	Yes	Yes
R6	No	No	Weak	Weak	No	Yes	No	Weak	Weak
R7	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
R8	No	Yes	Yes	Yes	Yes	No	No	Yes	Yes
R9	Yes	Yes	Weak	Yes	Yes	No	No	Yes	Yes
R10	Yes	Yes	Weak	Yes	Yes	Yes	No	Yes	Yes
R11	No	Yes	Yes	Yes	Yes	No	No	Yes	Yes
R12	No	No	Weak	Weak	Weak	No	No	Weak	Weak
R13	No	No	No	Weak	No	Yes	No	Weak	Weak
R14	Yes	Yes	Yes	Yes	Weak	Yes	Yes	Yes	Yes
R15	No	No	No	Yes	No	No	No	Yes	No
R16	No	No	No	No	No	No	No	No	No

CVQL [5], BVQL [6], SRQL [7] and UMQL [8] are all text-based query languages proposed on the level of database management systems. CVQL [5] 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 [6] 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. SRQL [7] is a SQL-like query language for surveillance video retrieval, which enables to make queries at the semantic level, enables the users to define their own scenarios based on semantic events, and retrieves videos with both exact matching and similarity matching. UMQL [8] is a SQL-like query language, which extends the WHERE clause to introduce feature condition, structure condition and spatial-temporal condition, so as to well meet users' query requirements. However, it is a general-purpose multimedia query language, and is not designed for video databases.

In order to supply a more visual interface, some graphical multimedia query languages were also proposed, *e.g.*, MQuery [13], VISUAL [14], WS-QBE [15], GMQL [9], *etc*. MQuery [13] can express questions over multimedia, timeline and simulation data using a single set of related query constructs. VISUAL [14] is designed for applications where the data has spatial properties. It borrows the example-element concept of Query-by-Example to formulate query objects, and it uses such concepts as hierarchical sub-queries, and internal and external queries. However, the language cannot express temporal relationships. WS-QBE [15] 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. GMQL [9] is a visual interface defined over the other general-purpose multimedia query language UMQL.

Besides, aiming at other unconventional data types, there are also some query languages (*e.g.*, those in [10, 16, 17] for XML data), or interfaces [18, 19, 20] Although being able to support to query video data to a certain extent, they are all not designed for video databases, thereby making them not suitable to be applied into video databases.

Then, we use the evaluation rules mentioned in Section 1.2 to evaluate our proposed query language, as well as all the above query languages. The evaluation results are shown as Table 5. In Table 5, we use R1-R16 to represent all the 16 rules, and use 'Yes', 'No' and 'Weak' to represent support, nonsupport and weak support, respectively. From Table 5, we see that SVQL can satisfy most rules, and thus it owns more powerful video query expressiveness compared with other existing query languages.

6. Conclusion and Future Work

In this paper, we have proposed a query language called SVQL (Structured Video Query Language) for video databases, which is developed based on an extension of the traditional database language SQL. SVQL keeps the clear syntax framework of SQL, making it with a user-friendly interface; and SVQL allows to users to query target videos based on various kinds of syntactic and semantic information, making it with powerful expressiveness.

The future research work on SVQL includes: (1) we first need to enhance the query language by developing a more user-friendly environment; (2) then in the second step, we will expand the content of SVQL including data manipulation (*e.g.*, INSERT, UPDATE *etc.*), data integration, as well as a more powerful graphical interface for displaying target videos; and (3) we last need to further study the query processing techniques more suitable for our query language.

Acknowledgements

This work described in this paper is supported by grants from the Zhejiang Provincial Natural Science Foundation of China (Nos. LY15F020020, LQ13F020009, Y15F020066 and LY13F010005) and the National Natural Science Foundation of China (Nos. 61202171 and 51379176).

References

[1] N. Durak, A. Yazici and R. George, "Online Surveillance Video Archive System", Proceedings of International Multimedia Modeling Conference, **(2007)**; Singapore.

[2] M. V. S. Nepal, Ramakrishna and J. A. Thom, "A Fuzzy Object Query Language (FOQL) for Image Databases", Proceedings of DASFAA, **(1999)**; Hsinchu, Taiwan.

[3] ISO, Information Technology-Database Language SQL. Standard No. ISO/IEC 9075:1999, International Organization for Standardization (ISO), (**1999**).

[4] M. Kaufmann, "The Object Database Standard", ODMG 3.0, (**2000**).

[5] T. C. T. Kuo and A. L. P. Chen, "Content-based Query Processing for Video Databases", IEEE Transactions on Multimedia, vol. 2, no.1, **(2000)**.

[6] M. E. Dönderler, E. Şayko, U. Arslan, Ö. Ulusoy and U. Güdükbay, "BilVideo: Design and Implementation of a Video Database Management System, Multimedia Tools and Applications", vol. 27, no.1, **(2005)**.

[7] T.-L. Le, M. Thonnat, A. Boucher and F. Bremond, "A Query Language Combining Object Features and Semantic Events for Surveillance Video Retrieval", Proceedings of International Multimedia Modeling Conference, **(2008)**; Singapore.

[8] Z.-S. Cao, Z.-D. Wu, and Y.-Z. Wang, "UMQL: A Unified Multimedia Query Language", Proceedings of International Conference on Signal-Image Technologies and Internet-based Systems, (**2008**); Shanghai, China.

[9] Z.-D. Wu, G.-D. Xu, Y.-C. Zhang, Z.-S. Cao, G.-L. Li and Z.-W. Hu, "GMQL: A Graphical Multimedia Query Language, Knowledge-Based Systems, vol.26, **(2012)**.

[10] Y. Wang, C.-X. Xing and L.-Z. Zhou, "Video Semantic Models: Survey and Evaluation", International Journal of Computer Science and Network Security, vol.6, **(2008)**.

[11] Z.-S. Cao, Z.-D. Wu and Y.-Z. Wang, "Multimedia Query Languages and Their Evaluation Rules", Journal of Computer Science, vol.36, no.3, **(2009)**.

[12] W. Ni and T. W. Ling, "GLASS: A Graphical Query Language for Semi-structured Data", Proceedings of DASFAA **(2003)**; Japan.

[13] J. D.N. Dionisio and A. F. Cárdenas, "MQuery: A Visual Query Language for Multimedia, Timeline, and Simulation Data", Journal of Visual Language and Computing, vol.4, **(1996)**.

[14] N. H. Balkir, G. Ozsoyoglu, and Z. M. Ozsoyoglu, "A Graphical Query Language: VISUAL and its Query Processing", IEEE Transactions on Knowledge and Data Engineering, vol.5, **(2002)**.

[15] I. Schmitt, M. N. Schulz and M. T. Herstel, "WS-QBE: A QBE-like Query Language for Complex Multimedia Queries", Proceedings of International Conference on Multimedia Modeling, (**2005**); Hongkong, China.

[16] S. Flesca, F. Furfaro and S. Greco, "A Query Language for XML based on Graph Grammars", World Wide Web Journal, vol.5, no.2, **(2002)**.

[17] D. Braga, and A. Campi, "XQBE: A Graphical Environment to Query XML Data, World Wide Web", Journal, vol.8, no.3, **(2005)**.

[18] I. Khalil and F. Sufi, "Cooperative Remote Video Consultation on Demand for e-Patients", Journal of Medical Systems, vol.33, **(2009)**.

[19] D. Fötsch and E. Pulvermüller, A Concept and Implementation of Higher-level XML Transformation Languages, Knowledge-Based Systems, vol.22, No.3, **(2009)**.

[20] A. G. Tapeh and M. Rahgozar, A Knowledge-based Question Answering System for B2C eCommerce, Knowledge-Based Systems, vol.21, no.8, **(2008)**.

Authors

ChengLang Lu, he is currently pursuing his Ph.D. degree from School of Marine Engineering, Northwestern Polytechnical University, China. He received his B.S. degree in Computer Science from Wenzhou University in 2005 and Master of Software Engineering degree from Huazhong University of Science and Technology (HUST) in 2009. His research interests include content-based video information retrieval and query processing.

MingYong Liu, he is a professor at School of Marine Engineering, Northwestern Polytechnical University, China. He received his Ph.D. degree from Northwestern Polytechnical University, China in 2001. His research interests are primarily in the area of information fusion and its application.

ZongDa Wu, he is an associate professor in Computer Science at Wenzhou University. He received his B.Sc. degree in Computer Science from Wenzhou University in 2005 and Ph.D. degree in Computer Science from Huazhong University of Science and Technology (HUST) in 2009. His research interests are primarily in the area of information retrieval and information security.