VisualMOQL: The DISIMA Visual Query Language*

Vincent Oria, M. Tamer Özsu, Bing Xu†, L. Irene Cheng and Paul J. Iglinski
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2H1
{oria, ozsu, bing, lin, iglinski}@cs.ualberta.ca

Abstract

Multimedia data are now available to a variety of users ranging from naive to sophisticated. To make querying easy, visual query languages have been proposed. Most of these languages have a low expressive power and have their own query processors. Efforts have been made to design query languages with proper semantics to facilitate query optimization and processing in existing database systems. The majority of multimedia database systems are built on top of object or object-relational database systems with the underlying query facilities inherited. The DISIMA system is being built on top of a commercial OODBMS and we have chosen to extend the standard object-oriented query language OQL with some multimedia functionalities. The resulting language is called MOQL. This paper presents VisualMOQL, a visual query language implementing the image component of MOQL.

1 Introduction

In this paper we present the visual query interface, VisualMOQL, of the DISIMA distributed image database management system under development at the University of Alberta. The topics under investigation include (i) the development of an object-oriented DBMS kernel that provides flexibility for user-defined classification of images, provides support for feature-based and spatial querying over image content (by means of salient objects), and enables reasoning over spatial relationships for query optimization; (ii) the development of query languages and primitives for querying image databases; and (iii) the provision of scalability and open access to image repositories. The DISIMA prototype is being implemented on top of the ObjectStore system [9].

VisualMOQL is based on a textual query language we developed, Multimedia OQL (MOQL) [10]. MOQL extends the standard object query language OQL [4] by adding spatial, temporal, and presentation properties for content-based image and video data retrieval, as well as for queries on structured documents. VisualMOQL implements only the image part of MOQL for the DISIMA project. A query specified using VisualMOQL is translated into MOQL to make use of the MOQL parser and query processor.

The complex structure and semantics of multimedia data make their access by a classical query language non-trivial. Since the media are inherently visual, it makes sense to provide visual querying capability. The approach whereby user requests are visually represented is known as *visual language*, *iconic language*, or *graphical language* [3]. *Graphical language* [2] refers to visual languages based on semantic models which make use of graphs, flow-charts or block-diagrams to represent objects and relationships defined among them. In *iconic languages* [7], queries are expressed by selecting and combining icons (visual metaphors) to produce new ones. In general, the expressive power of visual query languages is low since they are directed at naive users and are often not based on a textual query language.

The capabilities of visual languages can be enhanced if they are based on powerful multimedia query languages, which themselves may be extensions of object or object-relational query languages. This provides a visual query language that enables easy querying of multimedia databases, while benefiting from the query facilities provided by the database management system (DBMS). This is the approach we have chosen. In this paper, we present VisualMOQL, a visual language for the image component of MOQL. A demo of the system is available at http://www.cs.ualberta.ca/~database/DISIMA/Interface.html. A description of the demo is provided in [14].

The remainder of this paper is organized as follows: Section 2 gives an overview of the DISIMA project, Section 3 presents VisualMOQL, Section 4 explains the semantics of Visual MOQL queries, Section 5 discusses the implementation of the query processor, Section 6 discusses some related work.

---

*This research is supported by a strategic grant from the Natural Science and Engineering Research Council (NSERC) of Canada.

†Current address: IBM Toronto Lab, Ontario, Canada, xubing@ca.ibm.com

2 The DISIMA System Overview

This section gives an overview of the DISIMA model, the MOQL query language and the image annotation process. Details on the DISIMA model can be found in [12, 13] and MOQL is fully defined in [10].

2.1 The DISIMA Model

The DISIMA model [12, 13], is composed of two main blocks: the image block and the salient object block. We define a *block* as a group of semantically related entities.

The image block is made up of two layers: *image* layer and *image representation* layer. We distinguish an image from its representations to maintain an independence between them (*representation independence*). At the *image* layer, the user defines an image type classification which allows categorization of images.

DISIMA views the content of an image as a set of *salient objects* (i.e., interesting entities in the image) with certain spatial relationships to each other. The *salient object* block is designed to handle salient object organization. For a given application, salient objects can be defined by the user and identified in images by means of an annotation process. The definition of salient objects can lead to a type lattice. DISIMA distinguishes two kinds of salient objects: physical and logical. A *logical salient object (LSO)* is an abstraction of a salient object that is relevant to some application; a *physical salient object (PSO)* is a syntactic object in a particular image with its semantics given by a logical salient object. Figure 1 shows examples of both a salient object and an image class hierarchy.

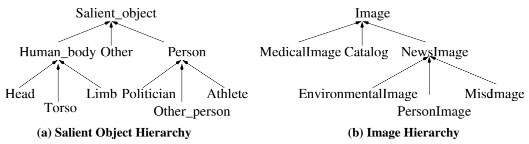

Figure 1. An Example of Image and Logical Salient Object Hierarchies.

The DISIMA model addresses both image and spatial databases and allows for the independence of image representations and applications. Moreover, it distinguishes the existence and identity of logical salient objects from their appearance in an image (physical salient objects).

2.2 Image Annotation

The representation of salient objects and their spatial relationships assumes the detection of these objects. We have tackled this issue within the DISIMA project by focusing on face detection. The reason for this choice is that the driving application of a news image database contains pictures with many persons in them. The image processing software first detects the faces contained in the image, marking them with a minimum bounding rectangle (useful for spatial relationships), and then provides color and texture values. Next, a human-annotator assigns a logical salient object to the face. In addition, an image has some descriptive properties (i.e., meta-data), such as date and photographer, that have to be provided. For this paper, we assume that the information at the two levels of salient objects is provided.

2.3 MOQL: A Multimedia Extension of OQL

An OQL query is a function which returns an object whose type may be inferred from the operators contributing to the query expression. As an embedded language, OQL allows applications to query objects that are supported by the native programming language. The basic statement of OQL is:

select [**distinct**] projection_attributes
from query [[**as**] identifier] {, query [[**as**] identifier] }
[**where** query] [**group by** partition_attributes] [**having** query]
[**order by** sort_criterion {, sort_criterion}]

Most extensions introduced to OQL by MOQL are in the **where** clause, in the form of four new predicate expressions: *spatial_expression*, *temporal_expression*, *contains_predicate*, and *similarity_expression*. The *spatial_expression* is a spatial extension which includes spatial objects, spatial functions, and spatial predicates. The *temporal_expression* deals with temporal objects, functions, and predicates for videos. The *contains_predicate* is defined as: *contains_predicate* ::= *media_object* **contains** *salientObject* where, *media_object* represents an instance of a particular medium type, e.g., an image or video object, while *salientObject* is an object within the *media_object* that is deemed interesting (salient) to the application (e.g., a person, a car or a house in an image). The **contains** predicate checks whether or not a salient object is in a particular media object. The **similarity** predicate checks if two media objects are similar with respect to some metric. VisualMOQL uses the DISIMA model to implement the image facilities of MOQL. A query Q: "Find images with 2 people next to each other without any building, or images with buildings without people, or images with animals" can be expressed in MOQL as follow:

```
SELECT m FROM image m, animal a, building b1,
              person p1, person p2
WHERE m contains a
OR ( m contains b1 and m not in
       (SELECT m1 FROM image m1, person p3
              WHERE m1 contains p3))
OR (m contains p1 and m contains p2
              and p1.MBB west p2.MBB and m not in
       (SELECT m2 FROM image m2, building b2
              WHERE m2 contains b2))
```

This example points the need for a visual query interface. Although the user may have a clear idea of the kind of images he/she is interested in, the expression of the query is not straightforward. VisualMOQL proposes an easier way to express queries, and then translates them into MOQL.

3 VisualMOQL

VisualMOQL [15] implements the image part of MOQL and allows users to query images by their semantics. Image semantics are based on the DISIMA model that views images as composed of salient objects with some properties.

The user can query the database by specifying the salient objects in the image. The query can be refined by defining the color, shape, and other attribute values of these salient objects. Furthermore, the user can specify the spatial relationships among salient objects in the image, which include both topological and directional ones. The user can also specify properties of the image meta-data - data members defined in the image class, such as the name of the photographer and the date.

VisualMOQL has these particular features:

- It is a declarative visual query language with a step by step construction of queries, close to the way people think in natural languages.
- It has a clearly defined semantics based on object calculus. This feature can be used to conduct a theoretical study of the language, involving concepts such as expressive power and complexity, which we consider out of the scope of this paper.
- It combines several querying approaches: semantic-based (query image semantics using salient objects), attribute-based (specify and compare attribute values), and cognitive-based (query by example). A user can start a query using the semantic and/or attribute-based approach and then choose an image for a cognitive-based query.

Although the cognitive-based querying is defined in MOQL and VisualMOQL, this feature is not yet implemented for the DISIMA system. The DISIMA model is rich enough to combine general image properties, including colors and texture, together with salient objects having semantics, colors, texture, shape, and spatial relationships. This leads to the definitions of several possible global image similarity functions. Basically, the user should be able to say "I want the similarity to be done on global image color features, with or without texture, with or without salient objects". The salient object semantic and syntactic features can be used to refine the similarity measurement. We are working on defining a flexible index able to handle all the possible similarity measures before making this feature available for DISIMA.

3.1 Query Interface

The VisualMOQL window (Figure 3) consists of a number of components to design a query. The user specifies a query by choosing the image class he/she wants to query and the salient objects he/she wants to see in the images. Several levels of refinement are offered, depending on the type of query and also on the level of precision the user wants the result of the query to have. The startup window consists of:

- A chooser to select the image classes. Images stored in the database are categorized into user-defined classes. Thus, the system allows the user to select a subset of the database to search over. The root image class is set as the default.
- A salient object class browser which allows the user to choose the objects that he/she wants. All salient objects and their associated attribute values are identified during database population. They are organized into a salient object hierarchy and the root salient object class is set as the default.
- A horizontal slider to specify the maximum number of images that will be returned as the result of the query. This is a quality of service parameter used by the query result presentation interface.
- A horizontal slider to specify the similarity threshold between the query image and the target images stored in the database. It is also used for color comparison. This is also a quality of service parameter for the presentation interface.
- A working canvas where the user constructs queries step by step.
- A query canvas where the user can construct compound queries based on simple queries (sub-queries) defined in the working canvas using AND, OR, and NOT operators.

3.2 Working Canvas

The working canvas is where the user constructs or modifies query blocks. The user first selects an image class, then selects a salient object class in the class browser. He/she inserts the selected salient object in the canvas by pressing the "Insert" button. The object appears as a rectangle in the working canvas. This rectangle is also used for determining the spatial relationships between objects. It could later be resized and moved. The user can also define the color, shape, texture, and other attribute values of any objects on canvas by using a dialog box shown in Figure 2. VisualMOQL allows the user to compare textual attributes. The default comparison predicate is `=' but can be changed to $\{<, >, \leq, \geq, \neq\}$. Since the variables used to refer to objects in the MOQL translation are shown on the object icons, they can be used to express join operators. For example, "find images with 2 persons of the same name" can be expressed by inserting two salient objects of type person in the working canvas. Assume VisualMOQL refers to them as P01 and P02. Then the user can edit one of the salient objects (e.g., P01) and type "P02.name" as the value for the attribute name (Figure 2). The query can involve image global properties like name of the photographer or time the image was taken, as well as global colors and textures. A dialog box obtained by clicking on the button "Image Property", is provided to let the user enter such information.

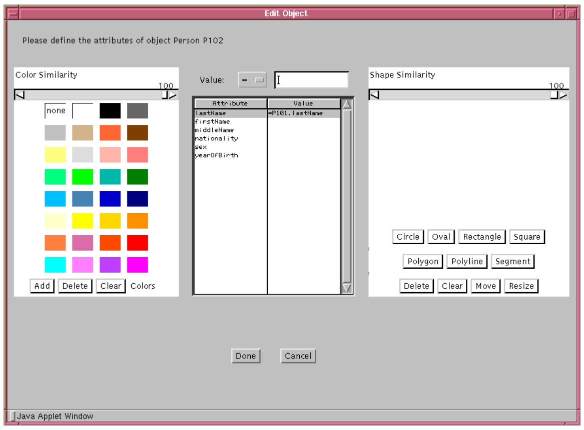

Figure 2. Dialog box for editing object attributes.

Topological relationships will be added automatically for any intersected objects. Directional relationships must be defined explicitly through a dialog box. The user specifies which axes (x-axis and/or y-axis) matter. The centroid of the rectangles representing salient objects is used to calculate the directional relationships. When both axes matter, we can express complex spatial relationships such as *northwest, southeast, overlap, etc*. When the user specifies that only one axis matters, the spatial relationships are *north, south, east, or west*.

We will use the term sub-query to refer to query blocks obtained from the working canvas. By clicking on the 'Validate' button, the user ends the sub-query specification. It is then moved into the query canvas where it can be combined with other sub-queries to form the final query.

3.3 Query Canvas

The query canvas is the space for the user to construct compound queries. Each sub-query is represented by a square box on the query canvas named *Query n* (n is an integer). Compound queries are constructed by combining sub-queries or smaller compound queries using AND, OR, and NOT operators. A sub-query in the query canvas can be modified and revalidated at any stage by using the `Edit' button. This moves the sub-query to the working canvas.

Finally, the user presses the query button to submit the query. Before translating this visual query, the system will check the query canvas to make sure there are no dangling queries. That is, all the sub-queries have to be linked using the AND, OR, or NOT operators. It will then translate the VisualMOQL query into MOQL and display the resulting string before submitting it to the query processor.

3.4 An Example of a VisualMOQL Query

Let us express the query Q: "find images with 2 people next to each other without any building, or images with buildings without people, or images with animals" in VisualMOQL. This query is a combination of three queries: Q_a (images with 2 people next to each other without any building), Q_b (images with buildings without people), and Q_c (images with animals). The final expression of the query is given in Figure 3. Q_a is expressed in MOQL by *(Query1 AND NOT Query2)* where *Query1* is a sub-query expressing images with two people, one to the west of the other (see the working canvas of Figure 3) and *Query2* expresses images with buildings. Q_b is expressed in MOQL by *(Query3 AND NOT Query4)* where *Query3* expresses images with buildings and $Query4$ expresses images with people. Q_c is expressed by the sub-query *Query5* and expresses images with animals. The final expression is obtained by combining the sub-queries using the *OR* connective. The VisualMOQL expression is translated into MOQL (Figure 4) before being submitted to the query processor.

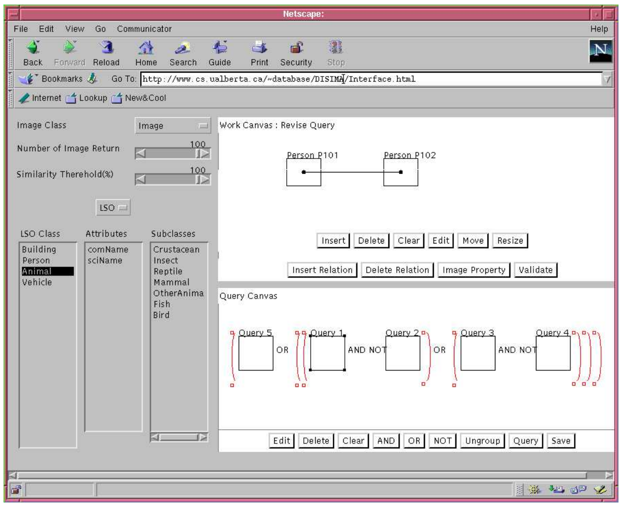

Figure 3. Example of Query.

4 VisualMOQL Query Semantics and Translation

VisualMOQL has a well-defined semantics based on object calculus. The working canvas represents an empty image; inserting salient objects into the working canvas, the user expresses the kind of salient objects he/she wants to see in query results (existence condition). The user can attach additional conditions on salient objects, including, spatial relationships among salient objects. A query at the working canvas level is viewed as a sub-query in the query canvas. If we view the images and the salient objects classes as complex value relations [1], a sub-query

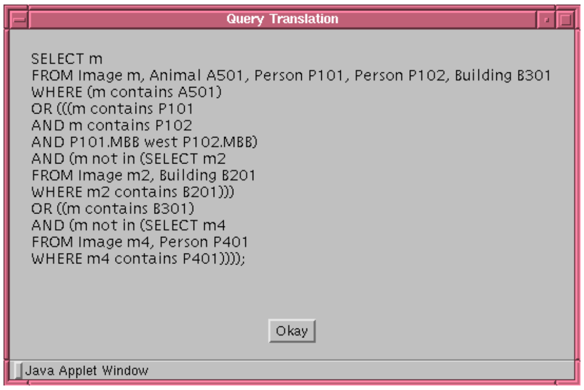

Figure 4. Query Translation.

from the query canvas is, in fact, a formula without any free variable. The general framework of such a formula is: $(\exists m \in M\ \exists s_1 \in S_1 \ldots \exists s_k \in S_k cond(s_1, \ldots, s_k) \wedge cond(m) \wedge m\ contains\ s_1 \wedge \ldots \wedge m\ contains\ s_k)$ where M is an image class, $S_1 \ldots S_k$ are salient object classes, $cond(s_1, \ldots, s_k)$ expresses boolean conditions on salient objects and $cond(m)$ expresses conditions on the image. There is only one image class in a sub-query. The default image class is *Image*, the root of the image class hierarchy, but this can be changed. As a formula without any free variable, a sub-query is evaluated to *true* or *false*.

At the query canvas level, sub-queries are combined using boolean operators and turned into queries. If ϕ and ψ are sub-queries then: $\neg\phi$, $(\phi \vee \psi)$, $(\phi \wedge \psi)$ are valid sub-queries. We insure there is no dangling sub-query in the case of more than one sub-query. The type of operators used in the query canvas raise some safety problems common in calculus languages. In the case of a query with a negation, what is the range of the query result? For example "images without buildings" will give different results depending on the universe it is applied to, normally a specification the user does not control. The problem is usually solved by range-restricting quantified or free variables. By construction, the variables in a sub-query are range-restricted. A sub-query has one free variable ranging over an image class and some salient object variables ranging over salient object classes.

4.1 Simple Queries

A sub-query ϕ, in which m is the variable over an image class is turned into a query Q as follows: $Q = \{i \mid \phi \wedge i = m\}$. The free variable i also has to be range restricted if ϕ is a negative formula: $Q = \{i \mid I(i) \wedge \phi \wedge i = m\}$. The example "images without buildings" makes sense if it is expressed as "images from the news image class without buildings". The problem in this case is where to get the images from. We set some simple rules to insure the safety of VisualMOQL queries. A sub-query is always applied to an image class (the root image class is set by default). If ϕ is an atomic sub-query with or without negation, the free variable i can be restricted to the image class M in ϕ as follows: $Q = \{i \mid M(i) \wedge \phi \wedge i = m\}$. The user will be able to express queries like "images without buildings" only within the context of a range of existing image classes.

4.2 Composed Queries

When the sub-query is a composed formula ($\phi\theta\psi$ with $\theta \in \{\vee, \wedge\}$), with two image variables that range over two different image classes, combining result objects of different types is problematic. We first determine the image class in which the query can be expressed and understood independently from its semantics. This is the least common ancestor of the two image classes in the image type system, i.e., a class where images from both image classes have the same type. Since the DISIMA image type system is rooted, the common ancestor will be the image root class in the worst case. Then we check the consistency of the query.

Assume we have an image variable m_1 ranging over M_1 in ϕ and a variable m_2 ranging over M_2 in ψ. This can also be seen as combining the results of two queries in an algebraic language: $Q_1\theta Q_2$ ($\theta \in \{\cap, \cup, -\}$). In this case, the images must be compatible:

- $\phi \wedge \psi$: there are two cases for which this query makes sense: (1) $M_1 = M_2$ or (2) M_1 is an ancestor of M_2 (or the reverse) in the type system hierarchy. Otherwise an error is detected.
- $\phi \vee \psi$: the type of the query result is set to the common ancestor (M) of M_1 and M_2. The query is understood as $(M \cap Q_1) \cup (M \cap Q_2)$ and the result is a set of M objects.

The final query is then transformed into a safe-range normal form so that the child of each negation is an existentially quantified formula (sub-query). The normalized formula is obtained by:

- variable substitution: the same image variable cannot range over two distinct image classes.
- push negation: replace $\neg\neg\phi$ by ϕ, $\neg(\phi_1 \vee \ldots \vee \phi_n)$ by $(\neg\phi_1 \wedge \ldots \wedge \neg\phi_n)$, and $\neg(\phi_1 \wedge \ldots \wedge \phi_n)$ by $(\neg\phi_1 \vee \ldots \vee \neg\phi_n)$.

For example, a query like "images with people and without buildings" can logically be expressed as "not(images with buildings or images without people)". The normalization facilitates the translations, as a negative existentially quantified formula expresses a set difference and can be translated using a nested MOQL query.

5 Query Processing

Although ObjectStore provides some querying facilities over collections, it does not have a built-in declarative query language. Therefore, we have fully implemented a MOQL parser and query processor for MOQL queries. Details on the parser and the query processor can be found in [5]. The result of the parser is an internal query tree structure which is later transformed into an execution plan.

5.1 The MOQL Parser

As MOQL queries follow the same SELECT-FROM-WHERE structure as traditional queries, the design of the DISIMA parser is able to make use of basic rules defined in SQL parsers [8]. The new rules defined on top of the basic rules deal with objects and the clauses introduced by MOQL. The objective of the DISIMA parser is to check the semantics and syntax of the external query, which is in the form of a character string. The parsed string is then converted into a query tree. A query object stores all the information given by the query string. Its main components are: *the select object* which stores the information given in the SELECT clause, *the from object* which stores the information given in the FROM clause, and *the where object* which stores the information given in the WHERE clause. The similarity thresholds and number of images required to be specified in the query are also stored in the query object. Figure 5 is the query object corresponding to the query:

SELECT m
FROM image m, LSO o
WHERE m contains o
AND o.color similar colorgroup(255,0,255)
AND o.texture similar texturegroup(0.6)
global similarity 0.8
image_required 30;

5.2 The Query engine

In our prototype query engine, we use the query tree directly to generate an unoptimized execution plan. Figure 6 shows the query execution tree corresponding to Figure 5.

In a query tree, each node is associated with either one or two conditions. When there are two conditions, either intersection or union is performed upon the results of the conditions for *and* and *or* operators, respectively. Once the query tree is constructed, the query engine initiates post-order traversal. The left-condition is executed before the right-condition, and any sub-node before the higher-level node.

The leaves of the query execution tree in Figure 6 represent three class extents: LSO, colorgroup, and texturegroup extents. The objects in the LSO extent that satisfy the *contains* condition are selected, in this case the entire extent.

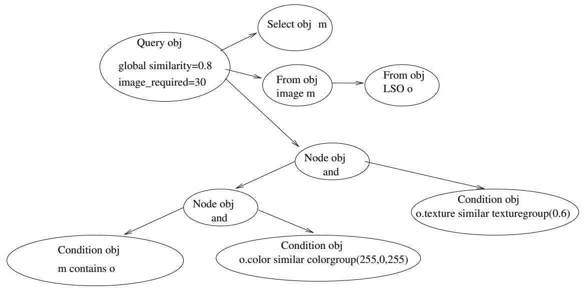

Figure 5. An example of a query object.

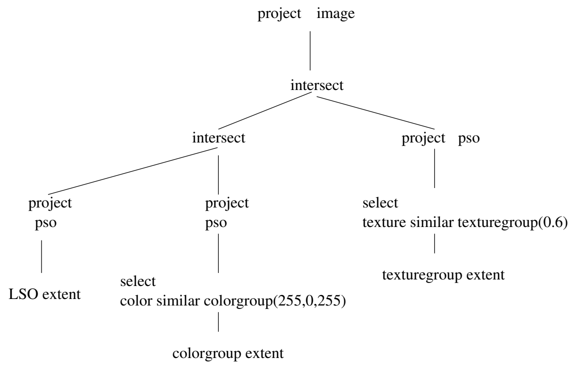

Figure 6. DISIMA execution tree.

The result of this condition is the set of all the PSOs associated with these LSOs. In executing the next condition, only colorgroup objects which have a color similar to colorgroup (255,0,255) are filtered from the colorgroup extent. Their associated PSOs then become the result of that condition. The logical connective *and* on the node above these first two conditions is then executed as an intersection operation. Similarly, the other branches are executed, resulting in a set of PSOs that satisfies all the conditions. Lastly, the images associated with these PSOs are returned as the query result.

6 Related Work

We focus on visual query languages that are coupled with internal query languages. This type of visual language provides an abstract set of operators and a mechanism to express operators defined in the internal language. There are a few prototypes using such an approach. The QBD* visual language [2] is not directed at multimedia applications but is one of the first visual languages to separate external model and querying from the internal query language. The QBD* visual query language is based on an entity-relationship at the external level and uses SQL internally. A query is expressed by selecting an entity, then following a path by choosing a relationship. Conditions can be expressed on attribute values. $Delaunay^{MM}$ [6] is a framework for querying multimedia data stored in distributed data repositories including the Web. The $Delaunay^{MM}$ query interface provides a format in which the user can enter SQL-like object queries. The queries are then translated into the syntax accepted by the destinations.

SEMCOG (SEMantics and COGnition-based image retrieval)[11] integrates semantic- and cognition-based approaches with a visual approach. The query is posed by specifying image objects and their layout using the visual query interface IFQ (In Frame Query). SEMCOG can be used to express a query like “Retrieve all images in which there is a man to the right of a car, and the man looks like this image”. The user introduces the first object (represented as a bullet) and describes it by attaching the descriptors "i_like <image>" and "is man". The second object is given "is car" descriptor. The spatial relationship is described by drawing a line labeled by *to-the-right-of* from the man object to the car object. The equivalent CSQL query is generated and shown in a separate window as the user specifies the query.

The main difference between VisualMOQL and SEMCOG is that the user does not need to be aware of the schema in SEMCOG - the reason why SEMCOG does not allow the user to assign values to attributes. SEMCOG integrates a *Facilitator* in charge of query reformulation. The facilitator will consult the *Terminology Manager*, which is a dictionary to replace "man" in the query by "person" if necessary. In VisualMOQL the user starts a query by browsing the schema to find (salient) objects he/she is interested in and also the kind of images (image class) he/she wants. From the implementation point of view, the VisualMOQL translator needs less work to translate user queries.

7 Conclusion

In this paper, we have presented VisualMOQL, a visual query language that implements the image features of MOQL, and described how the MOQL query processor is implemented. A demo of the system is available at http://www.cs.ualberta.ca/~database/DISIMA/Interface.html. VisualMOQL is a declarative visual query language with a step by step construction of queries, close to the way people think in natural languages, and is based on a clearly defined semantics using an object calculus. This feature can be the basis of a theoretical study of the language. VisualMOQL combines several querying approaches: semantic-based (query image semantics using salient objects), attribute-based (specify and compare attribute values), and cognitive-based (query by example). A user can start a query using the semantic and/or attribute-based approach and then choose an image for a cognitive-based query.

The complete cognition-based approach (query-by-example) is not yet implemented although the similarity functions on the colors, shapes, and texture are provided. Image semantic modeling in DISIMA combines general image properties including colors and texture, together with salient objects having colors, textures, and spatial relationships. This leads to the definition of several possible global image similarity functions. However, implementing several similarity matches using the same indexing structure is not trivial. While MOQL is a general-purpose multimedia query language, VisualMOQL implements only the image component. Work is in progress to extend VisualMOQL. To keep the query interface simple, our idea is to provide a VisualMOQL interface for each type of application (e.g., video, document, image). These interfaces must interact to achieve the expressive power of MOQL.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.

[2] M. Angelacio, T. Catarci, and G. Santucci. QBD a graphical query language with recursion. *IEEE Transactions on Software Engineering*, 16(10):1150—1163, 1990.

[3] C. Batini, T. Catarci, M. F. Costabile, and S. Levialdi. Visual query systems: A taxonomy. In E. Knuth and L. M. Wegner, editors, *Visual Database System, II*, pages 153—168, The Netherlands, 1992. Elsevier Science Publisher B. V. (North-Holland).

[4] R. G. G. Cattell, D. Barry, D. Bartels, M. Berler, J. Eastman, S. Gamerman, D. Jordan, A. Springer, H. Strickland, and D. Wade, editors. *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann, San Francisco, CA, 1997.

[5] L. I. Cheng. Image databases: A content-based type system and query by similarity match. M.Sc thesis, Department of Computing Science, University of Alberta, 1999.

[6] I. F. Cruz and W. T. Lucas. A visual approach to multimedia querying and presentation. In *Proceedings of Fifth ACM International Conference on Multimedia*, pages 109—120, Seattle, WA, November 1997.

[7] A. Del Bimbo, M. Campanai, and P. Nesi. A three-dimensional iconic environment for image database querying. *IEEE Transactions on Software Engineering*, 19(20):997—1011, October 1993.

[8] T. M. J.R. Levine and D. Brown. *Unix Programming Tools: Lex and Yacc*. O'Reilly and Associates, Inc, 1995.

[9] C. Lamb, G. Landis, J. Orenstein, and D. Weinreb. The objectstore database system. *Communications of ACM*, 34(10):50—63, 1991.

[10] J. Z. Li, M. T. Özsu, D. Szafron, and V. Oria. MOQL: A multimedia object query language. In *Proceedings of the 3rd International Workshop on Multimedia Information Systems*, pages 19—28, Como, Italy, September 1997.

[11] W.-S. Li, K. S., Candan, K. Hirata, and Y. Hara. SEMCOG: an object-based image retrieval system and its visual query language. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 521—524, Tucson, Arizona, May 1997.

[12] V. Oria, M. T. Özsu, X. Li, L. Liu, J. Li, Y. Niu, and P. J. Iglinski. Modeling images for content-based queries: The DISIMA approach. In *Proceedings of 2nd International Conference of Visual Information Systems*, pages 339—346, San Diego, California, December 1997.

[13] V. Oria, M. T. Özsu, D. Szafron, and P. J. Iglinski. Defining views in an image database system. In *Proceedings of the 8th IFIP 2.6 Working Conference on Database Semantics (DS-8) "Semantic Issues in Multimedia Systems"*, pages 231—250, Rotorua, New Zealand, January 1999.

[14] V. Oria, B. Xu, and M. T. Özsu. VisualMOQL: A visual query language for image databases. In *Proceedings of 4th IFIP 2.6 Working Conference on Visual Database Systems - VDB 4*, pages 186—191, L'Aquila, Italy, May 1998.

[15] B. Xu. A visual query facility for image databases. M.Sc thesis, Department of Computing Science, University of Alberta, 1999.