The VisualMOQL user manual *

DISIMA Group
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2H1
database@cs.ualberta.ca

1 Introduction

VisualMOQL is based on a textual query language that we developed called Multimedia OQL (MOQL) [LÖSO97]. MOQL extends the standard object query language OQL [CBB+97] by adding spatial, temporal, and presentation properties for content-based image and video data retrieval as well as for queries on structured documents. Hence, MOQL is a general-purpose multimedia query languages. VisualMOQL implements only the image part of MOQL for the DISIMA project [OÖL+97] and combines semantic-based (query image semantics using salient objects), attribute-based (specify and compare attribute values), and cognitive-based (query-by-example) querying. This combination leads to a flexible, and yet powerful visual query interface. A query specified using VisualMOQL is translated into MOQL to make use of the MOQL parser and query processor.

2 The Querying Interface

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.

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

- 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.

The VisualMOQL window (Figure 4) consists of a number of components to design a query. The user specifies a query by choosing the image class he wants to query and the salient objects he 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 the following components:

- A chooser to select the image classes. Images stored in the database are categorized into user-defined classes. By doing this, 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 wants. All salient objects and their associated attribute values are identified during database population. These objects 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.

2.1 Working Canvas

The working canvas is where the user constructs or modifies query blocks. The user first selects an image class and then selects a salient object class in the class browser. The user 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 1. VisualMOQL allows the user to compare textual attributes. The default comparison predicate is ‘=’ but can be changed to $\{<, >, \leq, \geq, <>\}$. Since the variable 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 respectively to them as P01 and P02. Then the user can edit one of the salient objects (let us say P01) and type “P02.name” as the value for the attribute name (Figure 1). The query can involve image global properties like name of the photographer or the time the image was taken. A dialog box (Figure 2) obtained by clicking on the button “Image Property” is provided to let the user enter such information.

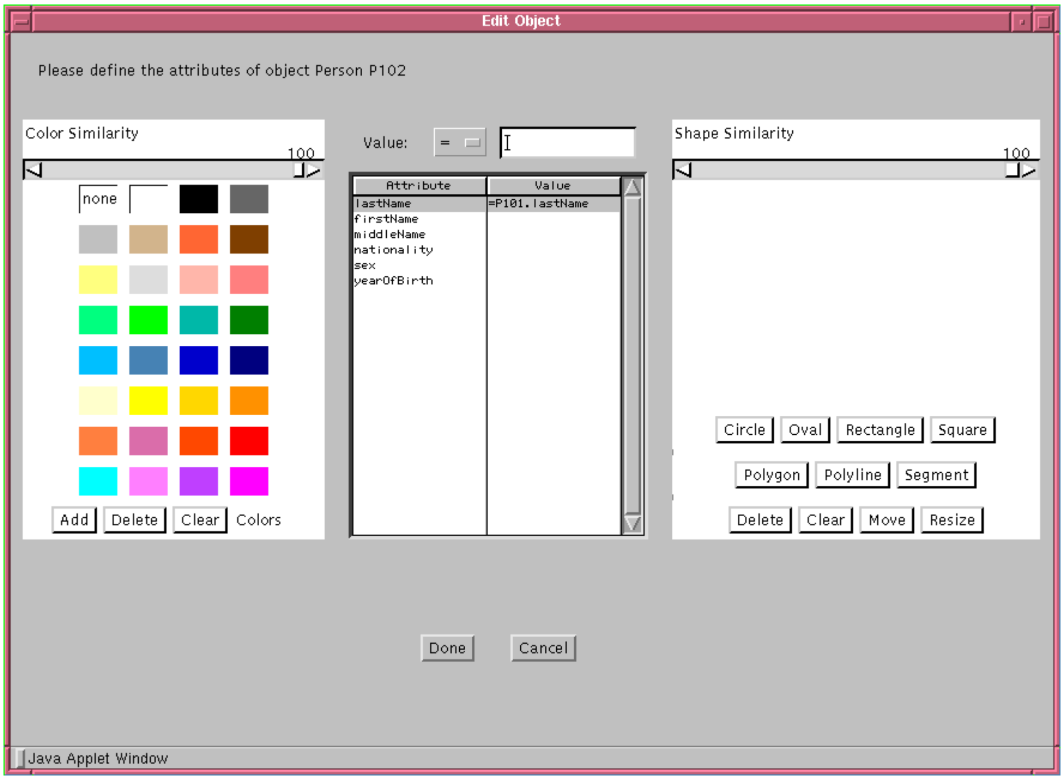

Figure 1: *Dialog box for editing object attributes.*

Topological relationships will be added automatically for any intersected objects. Directional relationships have to be defined explicitly through a dialog box shown in Figure 3. The user specifies which axes

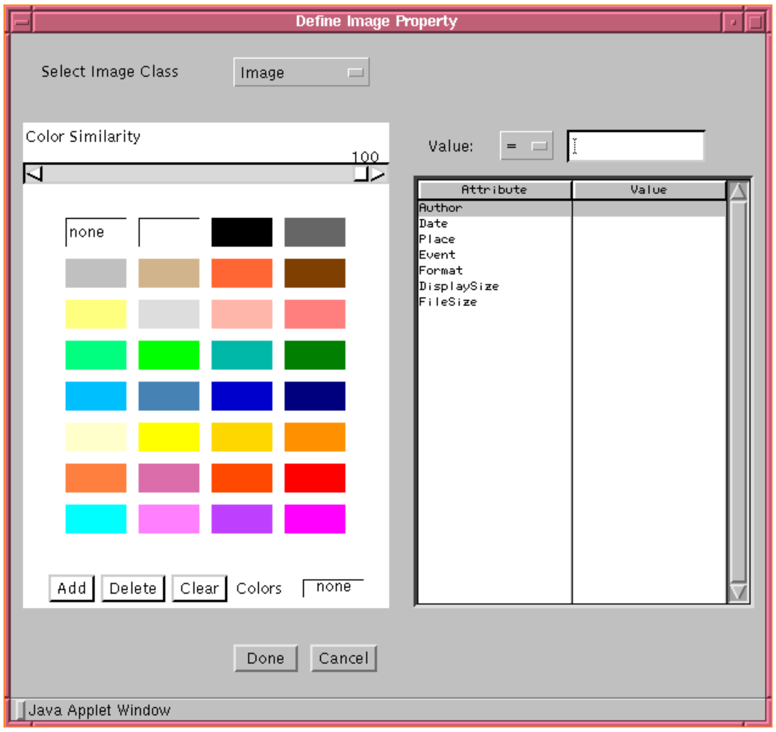

Figure 2: *Dialog box for defining image properties*

(x-axis and/or y-axis) matter. The centroid of the rectangles representing salient objects are 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. The sub-query is then moved into the query canvas where it can be combined with other sub-queries to form the final query.

2.2 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 and 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 the visual query

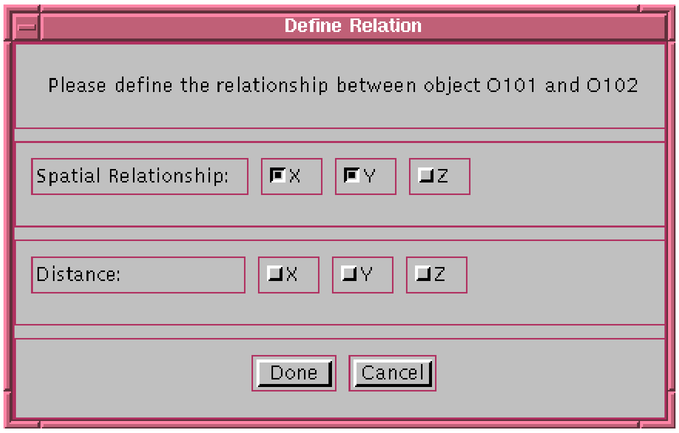

Figure 3: *Dialog box for defining directional relationship*

constructed by the user, 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. After this, it will translate the VisualMOQL query into MOQL and display the resulting string before submitting it to the query processor.

3 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 in fact 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
- Q_c: images with animals

The final expression of the query is given in Figure 4. 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 4) 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.

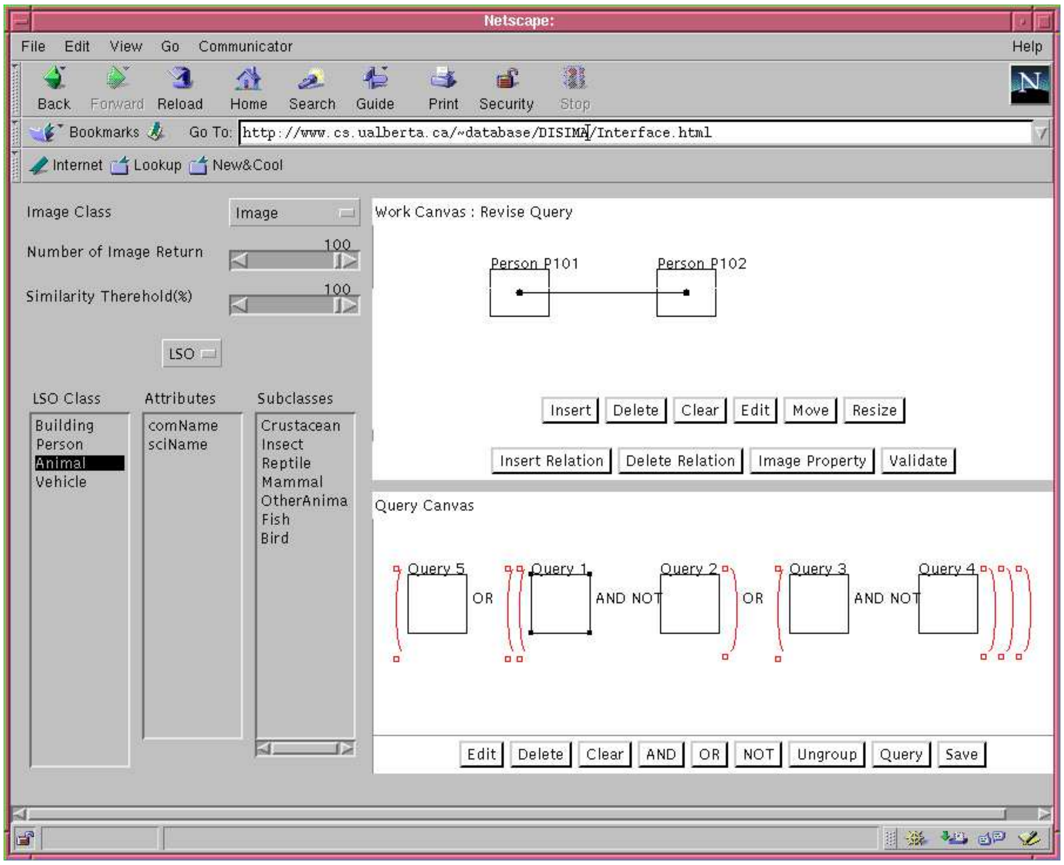

Figure 4: *Example of Query.*

The VisualMOQL expression is translated into MOQL (Figure 5) before being submitted to the query processor. Part of the query result can be seen in Figure 6.

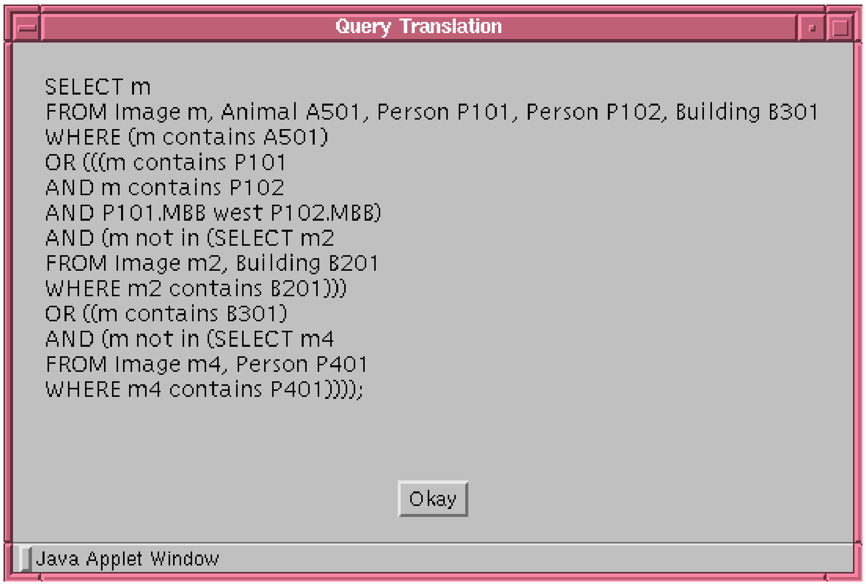

Figure 5: *Query Translation.*

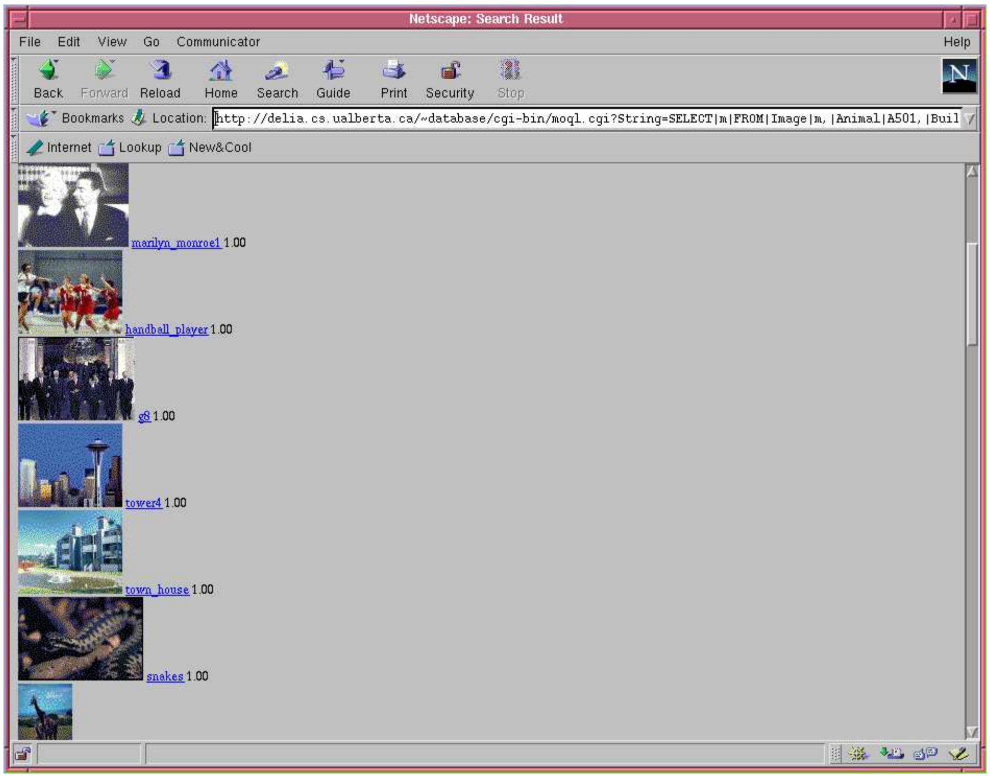

Figure 6: *Query Results.*

4 Conclusion

Visual query languages are seen as the next step in query language evolution. The aim of visual query languages is to provide users with a friendly interface that easily allows them to pose complex queries. Most of the existing visual query languages have a low expressive power due to the fact that they are often just GUIs designed for naive users. A powerful query language significantly helps to simplify multimedia database access. Such languages must provide constructs for querying based on the structure of multimedia data. For this purpose, we have defined MOQL (Multimedia Object Query Language). MOQL extends OQL by including extensions related to spatial properties, temporal properties, and presentation properties.

References

[CBB+97] 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.

[LÖSO97] 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.

[OÖL+97] 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.