MOQL: A Multimedia Object Query Language *

John Z. Li, M. Tamer Özsu, Duane Szafron, and Vincent Oria
Department of Computing Science, University of Alberta
Edmonton, Canada T6G 2H1

Abstract

We describe a general multimedia query language, called MOQL, based on ODMG's Object Query Language (OQL). In contrast to previous multimedia query languages that are either designed for one particular medium (e.g. images) or specialized for a particular application (e.g., medical imaging), MOQL is general in its treatment of multiple media and different applications. The language includes constructs to capture the temporal and spatial relationships in multimedia data as well as functions for query presentation. We illustrate the language features by query examples. The language is implemented for a multimedia database built on top of ObjectStore.

Keywords: multimedia, OQL, MOQL, Object-oriented, database, query, language

1 Introduction

One of the basic functionalities of a database management system (DBMS) is to be able to efficiently process declarative user queries. The penetration of DBMS technology into multimedia information systems necessitates the development of query languages appropriate for this domain. The complex spatial and temporal relationships inherent in the wide range of multimedia data types make a multimedia query language quite different from its counterpart in traditional DBMSs. For example, the query languages of traditional DBMSs only deal with exact-match queries on conventional data types. Although this might be sufficient to deal with queries posed against metadata or annotations of multimedia data, content-based information retrieval requires non-exact-match (fuzzy) queries which go beyond the traditional approaches.

Powerful query languages significantly help simplify multimedia database access. These languages have to provide constructs for querying, based on the structure of multimedia documents as well their context. Furthermore, *query presentation*, which refers to the way query results are presented, is more complex in multimedia systems than in traditional DBMSs. This is because multimedia presentations have to take into account the synchronization of various media. In recent years, there have been many multimedia query language proposals [BRG88, OM88, RFS88, AB91, DG92, CIT+93, Ege94, CIT94, HK95, KC96, ATS96, MS96, MHM96]. These proposals can be classified into three categories:

- Entirely new and specialized languages: [CIT+93, HK95, KC96, ATS96].
- Languages that are based on a logic or functional programming approach: [DG92, MS96].
- Languages that are extensions of SQL: [BRG88, OM88, RFS88, AB91, OT93, Ege94, CIT94, MHM96].

One problem with a brand new multimedia query language is the lack of acceptance by users. In general, it is difficult to convince users to learn and use a new language for each application. Another problem with the current proposals is the lack of theoretical framework to reason about the soundness and expressive power of the languages. In fact, none of the proposed new languages has ever addressed this problem.

*This research is supported by a grant from the Canadian Institute for Telecommunications Research (CITR) under the Network of Centres of Excellence (NCE) program of the Government of Canada and by a strategic grant from the Natural Science and Engineering Research Council (NSERC) of Canada.

Specifying queries using logic and functional programming approaches is relatively difficult for users. Therefore, this method is not very attractive, despite the expressive power of these languages. They may be suitable as formal multimedia query languages, but not as user languages. The majority of existing approaches to designing multimedia query languages are based on extensions of SQL. This is generally due to the popularity of SQL for traditional database applications. A common problem with all the existing SQL-based multimedia query languages is that they are designed either for a particular medium or for a specific application domain, not for general use. For example, VideoSQL [OT93] is used only for video databases, SEQL [CIT94] is designed mainly for medical sequence image databases, ESQL [AB91] is good only for image databases, and PSQL [RFS88] and SpatialSQL [Ege94] are suitable only for spatial databases.

Are there any general query languages for multimedia databases? How can they be formally defined so that they are independent of particular media and specific applications? These are the questions that this paper addresses. It is well-known that object technology is a promising one for dealing with multimedia data. As a result, most multimedia DBMSs are directly or indirectly (by extending relational models into object-oriented models) based on object-oriented technology. Object Query Language (OQL) [Cat94] has been proposed by Object Database Management Group (ODMG). OQL is currently supported by many Object DBMS (ODBMS) vendors and its popularity should increase as the ODBMS market grows. To the best of our knowledge, no multimedia extensions to OQL exist. In this paper, we propose an object-oriented, general-purpose query language, which we call MOQL, based on OQL. The language includes constructs to deal with spatial, temporal, and presentation properties. These extensions are introduced through predicates and functions. We describe the language features and give examples of their use.

The rest of the paper is organized as follows. Section 2 reviews the related work in multimedia query languages. Section 3 introduces ODMG's OQL and our basic extensions. The extensions include spatial primitives, temporal primitives, and the query presentation primitives. Section 4 briefly discusses the current status of implementing MOQL as part of a full fledged multimedia DBMS. It also summarizes our work and discusses possible future work.

2 Related Work

PSQL (Pictorial SQL) [RFS88] is designed for pictorial databases which require efficient and direct spatial search, based on the geometric form of spatial objects and relationships. One feature of PSQL is the introduction of many spatial operators, such as *nearest* and *furthest* for point objects, *intersect* and *not-intersect* for segment objects, and *cover*, and *overlap* for region objects. Syntactically, there is not much difference from the standard SQL.

EVA [DG92] is, an object-oriented language, based on functional language features with roots in conventional set theory. It is formally defined using the mathematical framework of a many sorted algebra. Although EVA has defined a set of spatio-temporal operators to support query presentation, it lacks some useful presentation features, such as changing display speeds and time constraints (e.g., presenting some object for 20 minutes). Furthermore, EVA does not support spatial queries or video data.

A knowledge-based object-oriented query language, called PICQUERY$^+$, is proposed in [CIT$^+$93]. PICQUERY$^+$ is a high-level domain-independent query language designed for image and alphanumeric database management. It allows users to specify conventional arithmetic queries as well as evolutionary and temporal queries. The main PICQUERY$^+$ operations include panning, rotating, zooming, superimposing, color transforming, edge detecting, similarity retrieving, segmenting, and geometric operations. A template technique has been used in PICQUERY$^+$ to facilitate user queries. Such *query templates* are used in PICQUERY$^+$ to specify predicates to constrain the database view.

SEQL (Spatial Evolutionary Query Language) [CIT94], a direct extension of SQL, is proposed to operate on the spatial evolutionary domains of medical images. In addition to alphanumeric predicates, SEQL contains constructs to specify spatial, temporal, and evolutionary conditions. A **when** clause is added to the language, which selects the appropriate snapshot of the data of interest at a particular point in time. It supports temporal functions which manipulate time points (such as *start time*, *end time* etc.), temporal ordering of an object history (such as *first*, *last*, *next*, etc.), and temporal intervals (such as *before*, *after*, *during*, etc.). Another extension is the addition of a **which** clause which describes various evolutionary processes on a set of evolving objects. Unfortunately, SEQL supports only image databases.

Marcus and Subrahmanian [MS96] have proposed a formal theoretical framework for characterizing multimedia information systems. The framework includes a logical query language that integrates diverse media. This is a first attempt at formally characterizing multimedia database systems. The model is independent of any specific application domain and provides the possibility of uniformly incorporating both query languages and access methods, based on multimedia index structures. The query language is based on logic programming and it makes extensive use of predicates and functions. Such a query language is suitable as an intermediate query language between a higher level language (such as OQL) and a lower level language (such as an object algebra).

Two new query languages, MMQL (Multimedia Query Language) and CVQL (Content-based Video Query Language), for video databases are described in [KC96] and [ATS96] respectively. A major problem with MMQL is that it does not support spatial queries which are fundamental to a multimedia query language. CVQL is defined based on video frame-sequences. Therefore, to query a video database using CVQL, a user must have good knowledge about the video (or frame sequence) being queried. ESQL [AB91] is an image domain query language for the relational model. The query language in [BRG88] is designed for multimedia office documents.

3 Object Query Language and Its Extensions

OQL defines an orthogonal expression language, in the sense that all operators can be composed with each other as long as the types of the operands are correct. It deals with complex objects without changing the set construct and the select-from-where clause. It is close to SQL92 with object-oriented extensions such as complex objects, object identity, path expressions, polymorphism, operation invocation, and late binding. OQL has one basic statement for retrieving information:

select [**distinct**] projection_attributes
from query [[**as**] identifier] {, query [[**as**] identifier] }
[**where** query]

where projection_attributes is a list of attribute names whose values are to be retrieved by the query. In the **from** clause, a variable has to be bound to a set of objects, an extent, or a query. In the **where** clause, the query is a conditional search expression that identifies the objects to be retrieved by the query. However, the conditional search expression can be any OQL query. In OQL, *query* is a very general expression and only the **select** form of a query corresponds to an actual user query. We assume some familiarity with OQL, but a detailed description can be found in [Cat94].

Most of the extensions that we introduce to OQL are in the **where** clause in the form of three new predicate expressions: *spatial_expression*, *temporal_expression*, and *contains_predicate*. The *spatial_expression* is a spatial extension which includes spatial objects (such as points, lines, circles etc.), spatial functions (such as length, area, intersection, etc.), and spatial predicates (such as cover, disjoint, left etc.). A detailed discussion of spatial extensions is given in Section 3.1. The *temporal_expression* deals with temporal objects, temporal functions, and temporal predicates. The *contains_predicate* has the basic form:

contains_predicate ::= media_object **contains** salientObject

where, *media_object* represents an instance of a particular medium type, e.g., an image object or a video object, while salientObject is a *salient object* which is defined as an interesting physical object in some media object. Each media object has many salient objects, e.g. persons, houses, cars, etc. The **contains** predicate checks whether a salient object is in a particular media object or not.

Query 1 Find all images in which a person appears.

select m
from Images m, Persons p
where m **contains** p

This simple query uses the **contains** predicate which checks whether a person p is in image m. The full set of multimedia extensions to OQL that we propose is specified in [LÖS97]. In the following sections, we discuss these extensions and give examples to demonstrate them.

3.1 Spatial Primitives

Spatial data pertains to the space occupied by objects and includes points, lines, squares, regions, volumes, etc. The special requirements of multimedia query languages in supporting spatial relationships have been investigated. From a user's point of view, the following requirements are necessary for supporting spatial queries in a multimedia information system:

- Support should be provided for object domains which consist of *complex* (structured) spatial objects in addition to simple (unstructured) points and alphanumeric domains. These spatial objects must be accessible by pointing to them or describing the space they occupy, and not just by referencing their encodings.
- Support should exist for *direct spatial searches*, which locate the spatial objects in given areas of images.
- It should be possible to perform *hybrid spatial searches*, which select objects based on some attributes and some associations between attributes and spatial objects.
- Support should exist for *complex spatial searches*, which locate spatial objects across the database by using set-theoretic operations over spatial attributes.
- Support should be provided to perform *direct spatial computations*, which compute specialized simple and aggregate functions from images.

3.1.1 Spatial Predicates

A spatial predicate compares the spatial properties of spatial objects and returns a boolean value as the result. We define only three spatial primitives: *point*, *line*, and *region*. Although other constructors, such as circle, rectangle etc., are provided, they are all special cases of *region*. A region may be represented by a set of points, a set of lines, a set of polygons, or other forms (e.g. a point and a radius) in some universe. The functions *nearest* and *farthest* are defined with respect to the set of points defined in a particular media object, such as an image or map. Table 1 shows basic spatial predicates defined in MOQL.

	point	line	region
point	nearest, farthest	within, midpoint	centroid, inside
line	cross	intersect	inside, cross
region	cover	cover, cross	topological_predicate, directional_predicate

Table 1: Spatial Predicates

The operands of the spatial predicates must be the same or compatible object types. For example, predicates *nearest* and *farthest* can apply only to two point objects, predicates *within* and *midpoint* can apply only to a point and a line, and predicate *cover* may apply to a region and a point or to a region and a line. We do not give exact definitions of spatial predicates (or for the temporal predicates in the next subsection) since they are self-explanatory. The directional relations include *left, right, above, below, front, back, south, north, west, east, northwest, northeast, southwest, southwest*, as well as the combinations of *front* and *back* with other directional relations. For example we can have *front_left, front_northwest*, etc. Precise definitions of directional relations are given in [LÖS96]. The topological predicates include *inside, covers, touch, overlap, disjoint, equal, coveredBy*, and *contains* which are specified in [EF91] as eight fundamental topological relations. However, two pairs of predicates are inverses: *cover* vs *coveredBy* and *inside* vs *contains*.

Query 2 Select all the cities, from a map of Canada, which are within a 500km range of the longitude 60 and latitude 105 with populations in excess of 50000:

select *c*
from Maps *m*, *m*.cities *c*
where *m*.name="Canada" **and** *c*.location **inside circle**(**point**(60,105), 500) **and** *c*.population>50000

For each map, the method *cities* retrieves all the cities in this map. Then, a city's location is checked to see if it is within the required range. **point** is a constructor which accepts two values or three values to create a 2D point or 3D point respectively. Here, **point**(60, 105) represents a 2D point; **circle** is a circle object constructor which accepts a spatial point acting as the center of the circle and a radius; **inside** is one of the spatial predicates from Table 1.

3.1.2 Spatial Functions

A spatial function computes attributes of an object or a set of spatial objects. The spatial functions are shown in Table 2. The return type refers to the type of the object returned by a spatial function. The *value* column contains some functions that return scalar values. Function *mbr* stands for minimum bounding rectangle. In addition to these, there is a universal function *distance*, which returns a scale value when applied to any two spatial objects. Function *region* for both point and line objects allows a point or line to be converted to a region. Hence, all the predicates and functions for regions are applicable to points and lines. For example, directly checking a directional relation between a line and a region is not allowed. However, after the conversion of a line to a region, such a check can be made.

Return Type	point	line	region	value
point	nearest, farthest		region	
line	intersect	intersect	region	length, slope
region	centroid		interior, exterior, mbr	area, perimeter

Table 2: Spatial Functions

Query 3 Find the forests and their areas from the maritime region where each forest is covered by a single province.

select *forest*, **area**(*forest*.region)
from Forests *forest*
where *forest*.region **coveredBy any**
  select *p*.region
  from Provinces *p*
  where *p*.region **coveredBy** *maritimeRegion*

The above query illustrates the binding of two nested mappings combined with the spatial function **area** and spatial predicate **coveredBy**. The provincial region is passed from the interior level and used to direct the search in the exterior, to produce those forests in the maritime provinces which are completely covered by individual provinces.

3.2 Temporal Primitives

The inclusion of temporal data in a multimedia query language is an essential requirement. Research in temporal queries has focused more on historical (discrete) databases rather than on databases of temporal media (e.g., [Sno95]). Thus, the focus has been on the reflections of changes of the representation of real world objects in a database (e.g., President Clinton gave a speech at 2:00pm on July 4, 1996), rather than changes in continuous and dynamic media action. Our interest is in temporal relationships among salient objects in multimedia data, not the real world historical relationships. A typical temporal multimedia query is "*Find the last clip in which person A appears*". The specification of the temporal relationship *last* needs special support from query languages to process this query. A *time interval* is identified as the basic anchored specification of time. Allen [All83] introduces a set of 13 temporal interval relations which have been widely accepted. The 13 relations are *equal, before, after, meet, metBy, overlap, overlapedBy, during, include, start, startedBy, finish, finishedBy.*

3.2.1 Temporal Functions

Our choice of functional abstractions for temporal objects is influenced by the work of [GLÖS96]. Interval unary functions which return the lower bound, upper bound and length of the time interval are defined, while binary functions contain set-theoretic operations viz *union*, *intersection* and *difference*. A time interval can be expanded or shrunk by a specified time duration. A *time instant* is a specific anchored moment in time. A time instant is modeled as a special case of a (closed) time interval which has the same lower and upper bound, e.g., $Jan\ 24, 1996\ =\ [Jan\ 24, 1996,\ Jan\ 24, 1996]$. A wide range of operations can be performed on time instants. A time instant can be compared with another time instant with the transitive comparison operators $<$ and $>$. A time span can be *added* to or *subtracted* from a time instant to return another time instant. A time instant can be compared with a time interval to check if it falls before, within or after the time interval. A *time span* is an unanchored relative duration of time. A time span can be compared with another time span using the transitive comparison operators $<$ and $>$. A time span can be subtracted from or added to another time span to return a third time span. We consider the following temporal granularity: *year, month, day, hour, minute, second, ms* (millisecond).

3.2.2 Continuous Media Functions

For continuous media, we consider only video data while audio will be considered in the future. We model a *video* as a sequence of clips and a *clip* as a sequence of frames. A *frame*, the smallest unit of a video object, can be treated as an image. Each frame is associated with a timestamp or time instant while a clip or a video is associated with a time interval. This implies that frames, clips, and videos can be ordered. Therefore, we can ask for the previous frame to a given frame or the last frame of a clip or a video. The continuous media functions are shown in Table 3. A universal function *timeStamp* applies to frames, clips, and videos and returns a time instant.

Return Type	frame	clip	video
frame	prior, next	clip	
clip	firstFrame, lastFrame, nth	prior, next	video
video		firstClip, lastClip, nth	

Table 3: Continuous Media Functions

Since video data consists of sequences of images, they share all the attributes of image data such as color, shape, objects, and texture. Unlike images, videos have temporal relations. Such temporal relations introduce dynamicity, (e.g. *motion*) which does not exist in image data. The implied motion in video data can be attributed to a camera (global) motion and an object (local) motion [ABL95]. In MOQL, an object motion is modeled by the multimedia database and then queried by using temporal predicates or functions. The definition of the abstract camera actions are based on [HK95]. A camera has six degrees of freedom, representing translation along each axis (x: track, y: boom, z: dolly) and rotation about each axis (x: tilt, y: pan, z: rotate). In addition, a change in the camera's focal length produces scaling or magnification of the image plane (zoom in and zoom out). To extract these features, each video stream should be first segmented into logical units by locating *cuts* (camera breaks). Cuts can be classified into different categories, such as fade, wipe, dissolve, etc. We define the following camera motion boolean functions: *zoomIn, zoomOut, panLeft, panRight, tiltUp, tiltDown, cut, fade, wipe*, and *dissolve*.

We assume that each continuous media object has a time interval associated with it that can be accessed through a method, *timestamp*. Furthermore, we assume that each salient object has a set of *timestamped physical representations*. The timestamped physical representation of a salient object indicates the physical characteristics of the salient object at different times. Typical physical characteristics of a salient object include geometric region, color, region approximation, etc. The set of physical representations of a salient object is accessible through method *prSet*.

Query 4 Find the last clip in which person p appears in the video *myVideo*:

select **lastClip**(**select** c **from** *myVideo*.clips c
  where c **contains** p
  order by **upperBound**(c.timestamp))

or

select c
from *myVideo*.clips c
where c **contains** p **and** (**upperBound**(c.timestamp) >= **all**
  select **upperBound**(d.timestamp)
  from *myVideo*.clips d **where** d **contains** p)

The first solution uses the features of the video function **lastClip** and OQL's **order by** clause. It is simpler than the second one. We assume that each video object has a method *clips* which returns a sequence of clips and each clip has a method *timestamp* which returns the time interval associated with this clip. Since two clips' intervals may overlap, we cannot simply rely on temporal predicates **after** or **meet** to perform the query. However, if we are sure that the upper bound of an interval is greater than or equal to all others, then its associated clip must be the last clip in a video. Therefore, we used a nested MOQL statement to express Query 4.

Query 5 List clips where person p_1 is at the left of person p_2 and later the two exchange their positions:

select c
from Clips c, p_1.prSet pr_{11}, p_1.prSet pr_{12}, p_2.prSet pr_{21}, p_2.prSet pr_{22}
where c **contains** p_1 **and** c **contains** p_2 **and** pr_{11}.region **left** pr_{21}.region **and**
  intersection(pr_{11}.timestamp, pr_{21}.timestamp) **during** pr_{11}.timestamp **and**
  pr_{12}.region **right** pr_{22}.region **and**
  intersection(pr_{12}.timestamp, pr_{22}.timestamp) **during** pr_{12}.timestamp **and**
  (pr_{11}.timestamp **before** pr_{12}.timestamp **or** pr_{11}.timestamp **meet** pr_{12}.timestamp)

Suppose clip c is the one we are looking for, then it contains both p_1 and p_2. In this case, both p_1 and p_2 must have at least two different physical representations respectively: one is p_1 at the left of p_2 and another one is p_1 at the right of p_2. We use pr_{11} and pr_{12} to represent the two states of p_1, and pr_{21} and pr_{22} to represent the two states of p_2. The spatial constraints bind p_1 to the left of p_2 and p_1 to the right of p_2 while the temporal constraints bind the intersection of pr_{11} and pr_{21} (as well as pr_{12} and pr_{22}) to be not empty. Such temporal constraints guarantee that p_1 and p_2 appear together sometime in this clip. Certainly, the timestamp of the relation p_{11} at the left of pr_{21} must be previous to the timestamp of the relation pr_{21} at the right of pr_{22}.

3.3 Presentation Functions

The query language has to deal with the integration of all retrieved objects of different media types in a synchronized way. For example, consider displaying a sequence of video frames in which someone is speaking, and playing a sequence of speech samples in a news-on-demand video system. The final presentation makes sense only if the speaking person's lip movement is synchronized with the starting time and the playing speed of audio data. Because of the importance of delivering the output of query results, query presentation has become one of the most important functions in a multimedia query system. Both spatial and temporal information must be used to present query results for multimedia data. The spatial information will tell a query system what the layout of the presentation is on physical output devices, and the temporal information will tell a query system the sequence of the presentation along a time line (either absolute time or relative time). We support presentation by adding a **present** clause as a direct extension to OQL:

present layout { **and** layout }

where, *layout* consists of three components:

- spatial layout which specifies the spatial relationships of the presentation, such as the number of windows, sizes and locations of the windows, etc.
- temporal layout which specifies the temporal relationships of the presentation, such as which media objects should start first, how long the presentation should last, etc.
- scenario layout which allows a user to specify both spatial and temporal layout using other presentation models or languages.

Detailed description of the presentation extensions can be found in [LÖS97]. We give one example to illustrate some of the features.

Query 6 Find all the image and video pairs such that the video contains all the cars in the image, show the image in a window at ((0, 0), (300, 400)) and the video in a window at ((301, 401), (500, 700)), and start the video 10 seconds after displaying the image; display the images for 20 seconds, but play the video for 30 minutes:

select	m, v
from	Images m, Videos v
where	**for all** c **in** (**select** r **from** Cars r **where** m **contains** r)
	v **contains** c
present	**atWindow**(m, (0, 0), (300, 400)) **and** **atWindow**(v, (301, 401), (500, 700)) **and**
	play(v, 10, **normal**, 30*60) **parStart** **display**(m, 0, 20)

Operator **parStart** starts both video and image media objects simultaneously. Therefore, the image object will be displayed immediately. However, since the start offset time for the video is 10 seconds, the video object will start 10 seconds after the image object starts. We use a default value (**normal**) for video playing. This can be changed for faster or slower playing by choosing a number either bigger than one or less than one respectively.

4 Conclusion

In this paper we describe a general-purpose multimedia query language called MOQL. Our approach is to extend the current de facto standard query language, OQL, to facilitate the incorporation of MOQL into existing ODBMSs. Users who are already familiar with OQL do not have to learn a new language. MOQL extends OQL by including extensions related to spatial properties, temporal properties, and presentation properties. We have implemented a proof-of-concept prototype of MOQL. The language is being implemented for a multimedia DBMS [ÖEMI+97] that is developed on top of ObjectStore [LLOW91]. A MOQL interpreter which parses MOQL queries and generates an algebraic tree is implemented in C while the rest is implemented based on ObjectStore using C++. This prototype is able to handle all the query examples given in this paper. The query processing approach is depicted in Figure 1 where the components that have been completed are shaded. Following the syntactic check, a semantic check is performed on the query to validate types, classes and class extents. At this point, it is possible to perform some semantic query optimization; however, our research has not yet fully addressed this issue. The correct MOQL queries are translated into an object algebra [ÖPS+95] that is used as the basis of optimization. The target object algebra that we use for this purpose is one that we developed for another project. This algebra is sufficiently powerful to support OQL queries.

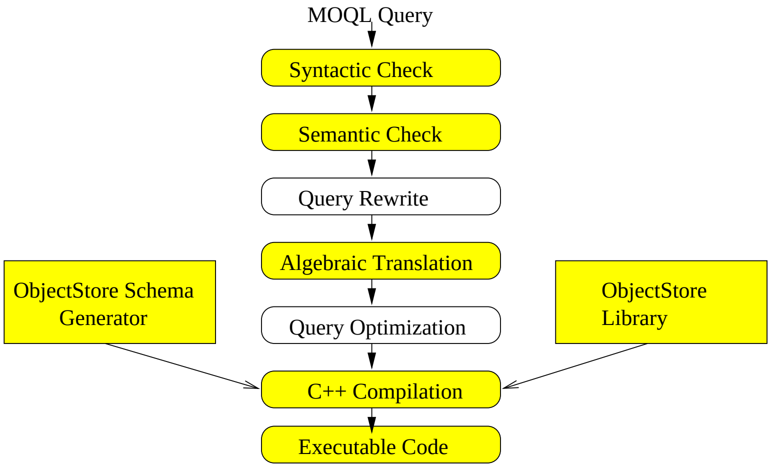

Figure 1: MOQL Query Processing

One of our research objectives is to study algebraic primitives to support optimization of multimedia queries. The target algebra that we currently use does not yet include these primitives so query optimization will be a topic of future research. In our current prototype each algebraic operator is implemented in terms of ObjectStore functions. This establishes a link between the query processor and the underlying object repository while enabling independent development of the query processor.

Further work needs to be done to investigate the support for audio media and to establish the the expressiveness of MOQL. Another important issue we are studying is the optimization of MOQL queries. This work is ongoing and will be reported separately. A parallel ongoing work is the development of a visual query interface built on top of MOQL. Even though MOQL provides powerful predicates, some multimedia queries are easier to specify visually. In an ideal environment, MOQL will establish the basis of a visual query interface and serve as the embedded query language for application development.

References

[AB91] R. Ahad and A. Basu. ESQL: A query language for the relational model supporting image domains. In *Proceedings of the 7th Int'l Conference on Data Engineering*, pages 550—559, Kobe, Japan, 1991.

[ABL95] G. Ahanger, D. Benson, and T. D. C. Little. Video query formulation. In *Proceedings of Storage and Retrieval for Images and Video Databases II, IS&T/SPIE Symposium on Electronic Imaging Science and Technology*, pages 280—291, San Jose, CA, February 1995.

[All83] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of ACM*, 26(11):832—843, 1983.

[ATS96] H. Arisawa, T. Tomii, and K. Salev. Design of multimedia database and a query language for video image data. In *Proceedings of IEEE Int'l Conference on Multimedia Computing and Systems*, pages 462—467, Hiroshima, Japan, June 1996.

[BRG88] E. Bertino, F. Rabitti, and S. Gibbs. Query processing in a multimedia document system. *ACM Transactions on Office Information Systems*, 6(1):1—41, January 1988.

[Cat94] R. Cattell, editor. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, San Francisco, CA, 1994.

[CIT+93] A. F. Cardenas, I. T. Ieong, R. K. Taira, R. Barker, and C. M. Breant. The knowledge-based object-oriented PICQUERY+ language. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):644—657, August 1993.

[CIT94] W. W. Chu, I. T. Ieong, and R. K. Taira. A semantic modeling approach for image retrieval by content. *The VLDB Journal*, 3:445—477, 1994.

[DG92] N. Dimitrova and F. Golshani. EVA: A query language for multimedia information systems. In *Proc. of Multimedia Information Systems*, Tempe, Arizona, February 1992.

[EF91] M. Egenhofer and R. Franzosa. Point-set topological spatial relations. *Int'l Journal of Geographical Information Systems*, 5(2):161—174, 1991.

[Ege94] M. Egenhofer. Spatial SQL: A query and presentation language. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):86—95, January 1994.

[GLÖS96] I. A. Goralwalla, Y. Leontiev, M. T. Özsu, and D. Szafron. Modeling time: Back to basics. TR-96-03, Department of Computing Science, University of Alberta, February 1996.

[HK95] N. Hirzalla and A. Karmouch. A multimedia query specification language. In *Proc. of Int'l Workshop on Multimedia Database Management Systems*, pages 73—81, Blue Mountain Lake, New York, August 1995.

[KC96] T. C. T. Kuo and A. L. P. Chen. A content-based query language for video databases. In *Proc. of IEEE Int'l Conference on Multimedia Computing and Systems*, pages 456—461, Hiroshima, Japan, June 1996.

[LLOW91] C. Lamb, G. Landis, J. Orenstein, and D. Weinreb. The ObjectStore database system. *Communications of ACM*, 34(10):19—20, 1991.

[LÖS96] J. Z. Li, M. T. Özsu, and D. Szafron. Modeling of video spatial relationships in an object database management system. In *Proc. of the Int'l Workshop on Multimedia Database Management Systems*, pages 124—133, Blue Mountain Lake, NY, August 1996.

[LÖS97] J. Z. Li, M. T. Özsu, and D. Szafron. Moql: A multimedia object query language. Technical Report TR-97-01, Department of Computing Science, University of Alberta, January 1997.

[MHM96] Elina Megalou, Thanasis Hadzilacos, and Nikos Mamoulis. Conceptual title abstractions: Modeling and querying very large interactive multimedia repositories. In *Proc. of the Int'l Conference on Multimedia Modeling*, pages 323—338, Toulouse, France, November 1996.

[MS96] S. Marcus and V. S. Subrahmanian. Foundations of multimedia database systems. *Journal of ACM*, 43(3):474—523, 1996.

[ÖEMI+97] M. T. Özsu, S. El-Medani, P. Iglinski, M. Schone, and D. Szafron. An object-oriented SGML/HyTime compliant multimedia database management system. accepted by ACM Multimedia, 1997.

[OM88] J. A. Orenstein and F. A. Manola. PROBE spatial data modeling and query processing in an image database application. *IEEE Transactions on Software Engineering*, 14(5):611—629, May 1988.

[ÖPS+95] M. T. Özsu, R. J. Peters, D. Szafron, B. Irani, A. Lipka, and A. Munoz. TIGUKAT: A uniform behavioral objectbase management system. *The VLDB Journal*, 4:100—147, 1995.

[OT93] E. Oomoto and K. Tanaka. OVID: Design and implementation of a video-object database system. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):629—643, August 1993.

[RFS88] N. Roussopoulos, C. Faloutsos, and T. Sellis. An efficient pictorial database system for PSQL. *IEEE Transactions on Software Engineering*, 14(5):639—650, May 1988.

[Sno95] R. T. Snodgrass. Temporal object-oriented databases: A critical comparison. In W. Kim, editor, *Modern Database Systems*, pages 386—408. Addison-Wesley, 1995.