MQuery: A Visual Query Language for Multimedia, Timeline, and Simulation Data

John David N. Dionisio   Alfonso F. Cárdenas

Computer Science Department

University of California, Los Angeles

July 25, 1996

Please direct all correspondence to:

Professor Alfonso F. Cárdenas
3731 Boelter Hall
University of California, Los Angeles
Los Angeles, CA 90024
U.S.A.

Phone: (310) 825-7550
Fax: (310) 825-2273
e-mail: `cardenas@cs.ucla.edu`

Abstract

This paper describes a visual query language that can express questions over multimedia, timeline, and simulation data using a single set of related query constructs. A uniform model for multimedia types organizes image, sound, video, and long text data in a consistent manner, giving multimedia schemas and queries a degree of data independence even for these complex data types.

Information that possesses an intrinsic temporal element can all be queried using a construct called a stream. Streams can be aggregated into parallel multistreams, thus providing a structure for querying and retrieving multiple sets of time-based information. The unified stream construct permits real-time measurements, numerical simulation data, and visualizations of that data to be aggregated and manipulated using the same set of operators.

Contents

1 Introduction

This paper describes the visual query language MQuery that can fulfill the requirements of such diverse domains as simulation and validation, medical timelines, and multimedia visualization. Our current research work in the Knowledge-Based Multimedia Medical Distributed Database (KMeD) project at UCLA has identified a number of query language needs that come from different application domains.

Background Recent developments in scientific databases have identified new data modeling requirements for next-generation scientific databases in areas such as:

- *Multimedia.* Scientific data is multimedia in nature: fully visual, frequently multi-dimensional, and spanning the dimension of time. Our work proceeds in a similar direction as the ones defined or explored in [1, 2].
- *Simulation and validation.* There is evidence of a greater need to integrate simulation and database technologies [3]. One benefit of this integration is the capability to validate simulation data by comparing it to real-world, measured data using the same query language.
- *Timelines.* Temporal, evolutionary, and process data models [4, 5, 6] in medicine track the progress and history of a patient over time, and a large element of science is the study of processes and their effects over time. Scientific and medical *timelines* present the progress of a tumor, hand bone growth, or other natural process as a series of frames that, properly registered, may be viewed as a movie or animation clip.

The areas addressed by these application domains are linked by a number of common threads:

- *The element of time.* Data that change over time — digital video, simulation data, timelines, etc. — require a database that can represent, query, and visualize this element of time. Better yet, time must be captured in a uniform construct regardless of the time scale or data type.
- *Complex data structures.* Objects having complex structures and interrelationships with other objects are needed in the medical domain. For example, the Unified Medical Language System (UMLS) is a large and complex semantic network of objects, processes, subjects, synonyms, and relationships [7]. If current relational data models are used, queries over UMLS are difficult to express without using artificial or arbitrary keys.
- *Multiple representations of objects.* Representation of an individual object within a wide variety of contexts arises for objects like tumors, which may be visible in multiple CT (computed tomog-

raphy) scans and MR (magnetic resonance) images. Tumors may be mentioned in lab reports or voice transcriptions, or represented by tabular simulation data. A conventional database is capable of storing most of these data types, but much work is required to ensure that a search for the general concept of a "tumor" leads to every context in which it appears. MQuery attempts to resolve this issue by making multiple representations a fundamental element of the language.

Contents of the Paper After summarizing MQuery's primary contributions to the field, we discuss the current state of research in this area (Section 2). Then, we proceed with an overview of the data model upon which MQuery is based (Section 3). Section 4 outlines the application domains for which we intend to test the functionality of our language. Section 5 discusses the language, including example queries of various types. Section 6 concludes the paper.

Primary Contributions

- *Generalized multimedia queries.* We define a general framework for querying all kinds of multimedia data, including images, sounds, long text, digital video, and integrated timelines. When applicable, a non-textual approach is employed to make predicates more natural — for example, drawing an example to retrieve an image or recording a sample to retrieve sounds.
- *Generalized time-based queries.* MQuery provides a visual interface to the querying of data stored as time-based streams. In particular, we use time-based streams to query multimedia, evolutionary, simulation, or timeline data.

2 Previous Work

Visual and multimedia query languages are heavily researched, and they span a wide variety of needs, objectives, and paradigms. This very high degree of diversity is acknowledged by many researchers in the field, including [8] and [9].

Image Query Languages This category of languages examined includes languages whose primary goal is to *query and process image or multimedia data.* Current research shows that only image data is robustly supported. Databases for querying audio and video data remain the subject of further research work.

PICQUERY emphasizes image retrieval [10] and processing while PICQUERY+ adds object orientation and temporal or evolutionary operators [11]. In both systems, the interface is strictly table-oriented, and full multimedia data types such as audio and video are not explicitly supported. Our proposed MQuery language is a logical step forward from PICQUERY+ and the work in [12], an initial exploration of purely visual image querying.

MQuery differs from other systems such as VIMSYS [13, 14], the 3D image work of del Bimbo et. al. [15], Query-By-Visual-Example (QVE) [16], and Query By Image Content (QBIC) [17]. MQuery emphasizes the *presentation and expression*, and not necessarily the processing, of image content queries. In addition, we focus on the integration of conventional alphanumeric databases and newer data types, such as simulation and video data.

Visual Query Languages Most query languages that are labeled as "visual" trace their heritage back to Zloof's Query-By-Example, developed in the early 1970's [18].

Pasta-3 [9] and the Visual Query Language (VQL) [8] are notable examples of current visual query languages. However, these systems are geared toward alphanumeric databases, and do not provide constructs for multimedia or time-based data.

The field of visual query languages is the focus of much research [19, 20, 21]. The primary distinction between MQuery and all of these systems is MQuery's full support for visual querying of *multimedia* and *temporal or stream data*, in addition to the alphanumeric domain.

Database User Interfaces *Database user interfaces* (DBUIs) unify, into a single interface, nearly all aspects of database management. DBUIs can be characterized as easier to use and better integrated because schema viewing, browsing, and querying are seamlessly viewed as a single set of activities. However, they do not have the same querying power as more narrowly-focused systems.

The MOOSE [22] and ER-DRAW [23] systems represent one aspect of an overall DBUI: schema design, browsing, and presentation. Although full DBUIs also possess schema facilities, they are, in general, less powerful than those designed specifically for schema interaction. The functionality found in MOOSE and ER-DRAW is best viewed as a subset of the functionality offered by MQuery.

SNAP has perhaps the best overall DBUI functionality [24]. MQuery adds to SNAP's functionality with extensions such as more complex queries and multimedia and temporal support.

Other DBUIs are present in earlier literature [25, 26]. The DBUI field provides strong user interface elements for databases, but they lack functions that can be found in their more narrowly-focused

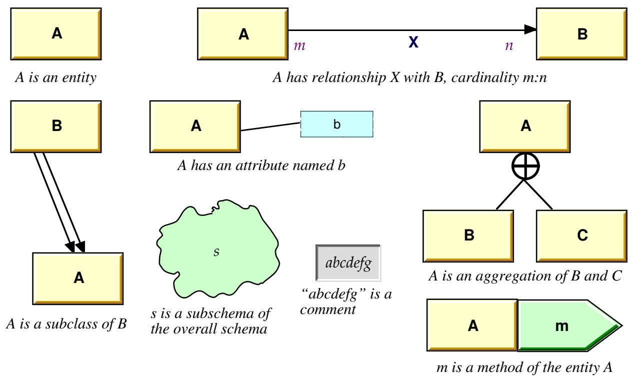

Figure 1: Notational conventions for basic data model constructs.

cousins, such as image and pure visual query languages. We feel that it is easier to integrate image and visual query functionality into a database user interface than the other way around, so we have chosen to take this approach in designing MQuery.

User Interface The current windows-icons-menus-pointer interface is being expanded by work on new paradigms for user interfaces. These include noncommand or active user interfaces [27], gestural user interfaces [28], voice or audio interfaces [29], and new ways to provide user feedback [30].

3D technology may help database administrators and users in visualizing and navigating large, complex data models [31, 32]. MQuery was designed with the awareness that this technology will increase in popularity when it matures.

3 Data Model Overview

We have designed a multimedia data model called M that supports the new and unique concepts in our query language [33]. We provide only an overview of M herein.

Basic Concepts Our basic data model framework is a synthesis of entity-relationship (ER) and object-oriented (OO) data models. We have tried to combine the diagrammatic simplicity of ER with the richer semantics of OO. Figure 1 summarizes the basic ideas and notation behind the model. More details appear in [33].

Multimedia Types The foundation for the multimedia functionalities in this data model is the *multimedia type*. Multimedia types are designed to behave like atomic data types within the system:

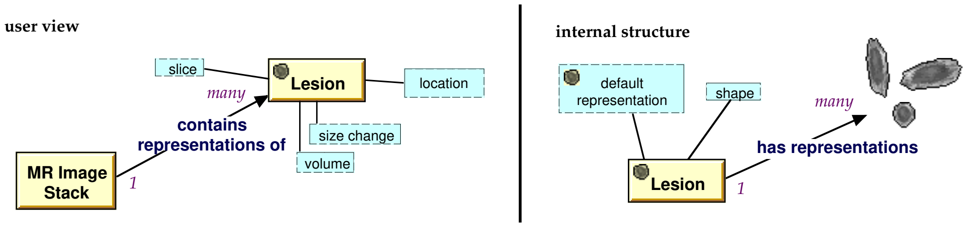

Figure 2: Structure of an intelligent visual entity.

the database provides all of the operations necessary to transparently manipulate them, just as a user can manipulate numbers, strings, or characters.

Multimedia types are viewed as encapsulated "black boxes" by the user. They can be presented on a screen or a speaker, assigned to attributes, compared to other values, or manipulated through pre-defined operations and/or methods. Although the actual implementation of such black box functionality is, internally, much more complex than with numbers or characters, the user-level view of these operations remains the same.

Intelligent Visual Entities An *intelligent visual entity* (IVE) is an entity that stores image representations of itself in the database schema. These image representations are chosen by the database administrator as "typical" for instances of a particular kind of IVE. We assume that images in the database have been segmented, either manually or automatically, or preprocessed or on-the-fly, to identify the queryable objects in the image. Visual querying is achieved by comparing an IVE's library of standard representations to the segmented image regions in the database.

Subclasses of the abstract IVE class may be defined for specific domains, such as lesions, anatomical regions, etc. These subclasses override or specialize the way an IVE's visual representations are used to perform image comparisons. For example, IVEs found in medical images might ignore color information when performing similarity comparisons, since most medical images are entirely in grayscale. Instead, they may focus purely on size (area) and location in relation to each other. Figure 2 illustrates the special structure and appearance of an intelligent visual entity.

An IVE has two appearances: as a standard entity (with an icon to show that it is an IVE), and as a graphic "representative" of that entity's instances. An IVE also has defaults for: its representation, relevant set of image features (i.e. brightness, texture, area, etc.), and, if necessary, methods that determine similarity with other image objects. The latter are set by the database administrator or designer when defining the IVE, and come into play during image query processing.

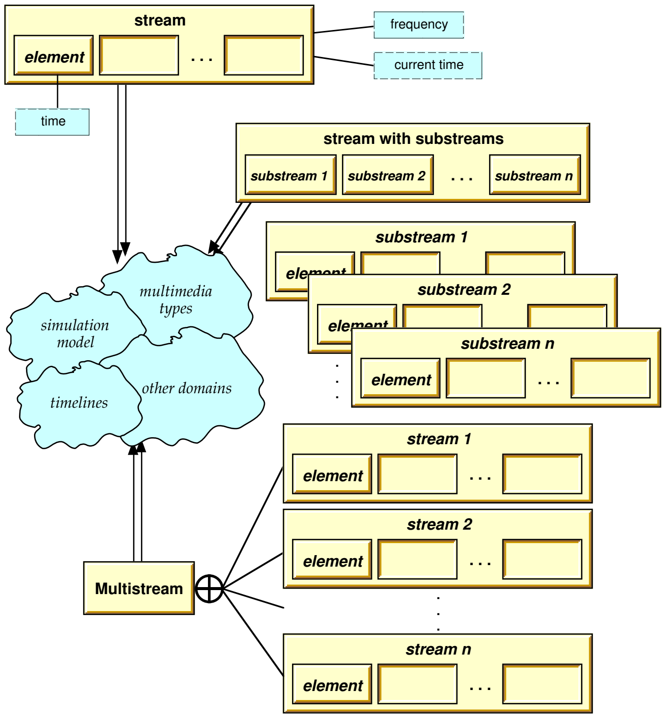

Figure 3: Graphical diagram of abstract stream entity types and their internal structures.

For example, a tumor IVE may define itself as being similar to another tumor if their areas and positions are within some threshold of each other. The user may override this default similarity measure at query time.

Streams A stream is *an ordered, finite sequence of entities or values*. These entities or values are called *elements* of the stream. The sequencing is temporal: in other words, elements e_i and e_j of a stream entity S are distinguished by i and j being different instances in time. This temporal ordering is further described by a *frequency*, indicating the speed by which the elements of the stream travel through time, along a time scale that can be different for different streams. Streams also have the notion of a *current time* from which the *current element* in the sequence can be determined — particularly useful when iterating over a stream's elements or when the passage of time is halted within a stream (i.e. viewing individual frames of a movie loop, for example).

Figure 3 illustrates the notation for some of our basic stream-related constructs.

4 Sample Application and Prototype

We now describe a sample application to show how the new constructs improve the usability and clarity of a database schema. The application is taken from the medical domain; other work that uses our new modeling concepts can be found in the literature [34, 35].

4.1 Domain and Requirements

The example domain presented here is based on a multimedia database for thermal ablation therapy of brain tumors that has been developed by our group.

Thermal ablation therapy is the use of focal heating for the treatment of tumors. Techniques for thermal ablation of brain tumors were pioneered in the 1960's, and have been further refined since then [3]. The procedure is particularly important in the treatment of brain tumors, where invasive surgery is either impossible or poses the risk of severe brain damage. Using specially designed interventional magnetic resonance instruments, a radiofrequency (RF) electrode is directed into the tumor with MR guidance. Instead of the usual surgical craniotomy exposure, a 2mm twist drill hole is used for access in the skull of the patient, who remains awake during the procedure.

The sample schema presented in this section maintains the data that are relevant to the thermal ablation therapy application domain. The database can store patient records and images, associate them with each other, and perform queries based on features such as tumor volume or histology. The schema also supports simulations of the heat transfer process that occurs during therapy.

4.2 Discussion of Schema

Figure 4 shows the overall thermal ablation therapy schema that we have developed as a testbed. The schema herein is actually a subset of a larger project between the UCLA Computer Science and Radiological Sciences Departments; a larger, broader data model spanning many other areas of medical imaging is currently under development.

4.2.1 Patients and Health Care Activities

The standard representation of a patient is shown in Figure 4. An individual `Patient` *undergoes* zero or more `Health Care Activity` instances. `Patient`s have attributes such as an ID, name, and many others.

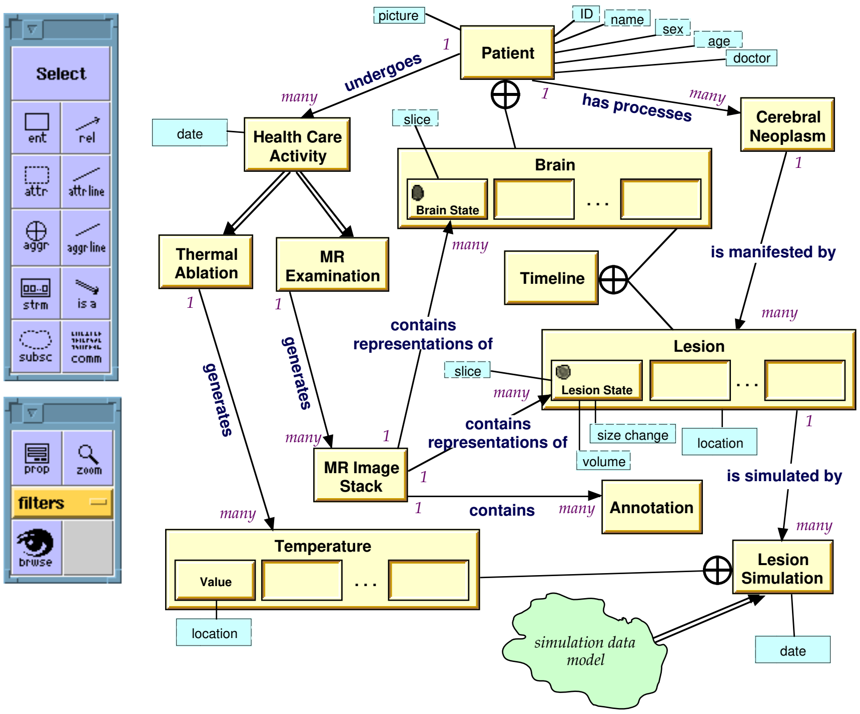

Figure 4: Sample schema using the M data model for thermal ablation therapy data management, showing tool palettes for drawing the schema on the left.

The sample database keeps track of two types of health care activities: `MR Examination`s and `Thermal Ablation` treatments. `MR Examination`s *generate* a set of MRI (magnetic resonance imaging) scans of the patient's brain and tumor. Thus, an `MR Image Stack` *contains representations of* the `Patient`'s `Brain State` and any `Lesion State`s at the time of the examination. This relationship shows an application of our multimedia type model. In addition, `Brain State`s and `Lesion State`s are modeled as intelligent visual entities (IVEs), because they directly correspond to some visible region of interest in the `MR Image Stack`.

`Thermal Ablation`s represent instances of thermal ablation procedures performed on the patient. They include: the number of doses applied, whether or not a biopsy was taken, etc. Measurements tracking the brain's temperature are also taken during the procedure, and so a `Thermal Ablation` *contains* a stream of temperature values.

4.2.2 Brain, Lesion, and Temperature Streams

Figure 4 illustrates the use of our stream model [35]. For one, as examinations accumulate over time, individual `Brain` and `Lesion State`s (essentially snapshots at a particular moment in time) are collected into streams that fully represent, within the requirements of the application domain, the `Patient`'s `Brain` and particular `Lesion`s within the `Brain`.

The `Brain` entity belongs to an overall aggregation that represents the `Patient`'s anatomical systems (other anatomical systems are not shown in the Figure 4, but have been modeled by our group). The `Lesion` entity belongs under one of the pathologic functions for which a `Patient` *has processes*. In this case, it is a `Cerebral Neoplasm` disease process which *is manifested by* one or more `Lesion`s.

The third use of the stream construct lies in our representation of the `Temperature` entity as a stream of individual temperature values. `Temperature` is used in two places: in the first case, a `Thermal Ablation` procedure *generates* a real-world stream of measurements, thus tracking the overall temperature of the tissue undergoing thermal ablation as it changes over time. Second, `Temperature` is one of the data streams of a `Lesion Simulation`. `Lesion Simulation`s follow our simulation model, capturing the heat transfer equations that theoretically govern the thermal ablation process. Thus, instances of `Temperature` may be directly compared to determine the accuracy of simulated treatments against measurements taken during actual treatments.

5 Query Language

MQuery is derived from a rich heritage of previous query language work, and thus many of its more fundamental notions do not radically deviate from current multimedia and visual query languages. MQuery functions as a superset of these languages, as well as a superset of other query language work performed by our group.

5.1 Overview

As discussed in Section 2, current research into query languages has expanded into a broad range of metaphors and paradigms: image query languages, visual query languages, and database user interfaces. Section 2 also pointed out that this diversity has resulted in systems that excel in specific areas but do not handle others at all.

One of the goals in defining MQuery is to integrate the diverse technologies that have been developed thus far into a single comprehensive system. MQuery's other goals are to design:

- a system that sufficiently supports the M data model and all of its constructs,

- a comprehensive database interface that captures the functionality of schema design, browsing, querying, and output within a unified environment,
- a system that is capable of storing, querying, and presenting all forms of data, particularly multimedia such as images, audio, and digital video, and
- a system using a visual and user-friendly metaphor, and to support and validate that design using structured usability testing techniques.

The last three goals correspond precisely to the specific goals of database user interfaces, image or multimedia query languages, and visual query languages respectively. Most systems in the literature (see Section 2) focus primarily on one of these goals at a time; our intent in MQuery is to apply a wholistic approach to satisfy all of these requirements.

The general sequence of activities performed by an MQuery user flows in this manner:

1. The user is presented with the overall data model. This represents the data that is available to the user, as stored in the database's dictionary/directory.
2. When the user is ready to ask a query, a query window is created. The user can drag or copy elements from the schema diagram to the query window. The query diagram that is built by the user forms the actual query statement for processing.
3. The user asks the system to insert, retrieve, delete, or update any matching data. The system responds by fetching the data (sometimes invoking certain specialized modules to accomplish this task), then presenting the results.

At any point in time, the user may edit the query diagram, invoke another query, or explore the schema further. The user can use any query result that is currently on display to perform further queries, perhaps to retrieve similar objects or objects within a narrower scope.

5.2 Schema Design

All schema-related activities in MQuery are initiated through the main *schema window*, along with one or more *schema tool palettes* (as shown in Figure 4). Each tool palette concentrates on a specific set of schema activities, including schema design, browsing, modification, and querying.

The *schema window* is an interactive representation of the underlying database schema. The representation is faithful to the notation adopted by the M data model (Section 3). The components of the schema window may be copied into query windows in order to specify the objects to be queried.

Schema design tools "understand" the semantics of the constructs that they draw onto the schema window. For example, when a relationship is being drawn, the relationship tool does not permit the user to begin drawing the relationship until the cursor is on the border of an entity. Similar semantic restrictions apply to other tools.

5.3 Query Formulation

MQuery's most significant contributions include:

- *Full-function query language integrated with a full-function schema display:* most works on the literature (see Section 2) focus primarily on one or the other, but not both.
- *Full support for entire range of query operations (insert, retrieve, delete, update):* most languages today focus primarily on information retrieval. In reality, this is only a subset of the entire range of query functionalities; little work has been done on giving operations such as insertion, deletion, and modification as friendly or visual an interface as the retrieval side. By defining a single construct (the query specification) that is usable by all query operations, MQuery creates a single interface that is capable of performing all four standard query operations.
- *Direct visual support for new data types such as images, sounds, and video:* most visual interface systems currently concentrate on alphanumeric data, while the few systems supporting multimedia or image data have narrowly-scoped query languages.

MQuery also inherits the complete feature set of its predecessor, PICQUERY+, including a hierarchical knowledge base and use of fuzzy operators [11].

To perform a query, the user defines a query specification (examples of which shall be illustrated throughout this section), then specifies whether MQuery is to perform an insertion, retrieval, deletion, or updating on that specification. The same query specification may be used as a basis for any of the four query actions; of course, the *interpretation* of the query specification differs, depending on the action.

After MQuery processes the query spec based on the user's desired action, it responds with a set of output windows. The types of windows displayed as output depend on the query action that was performed.

While forming a query spec, the user may go back and forth from the schema window to the query window, or the user may also examine output windows that are already on display due to previously

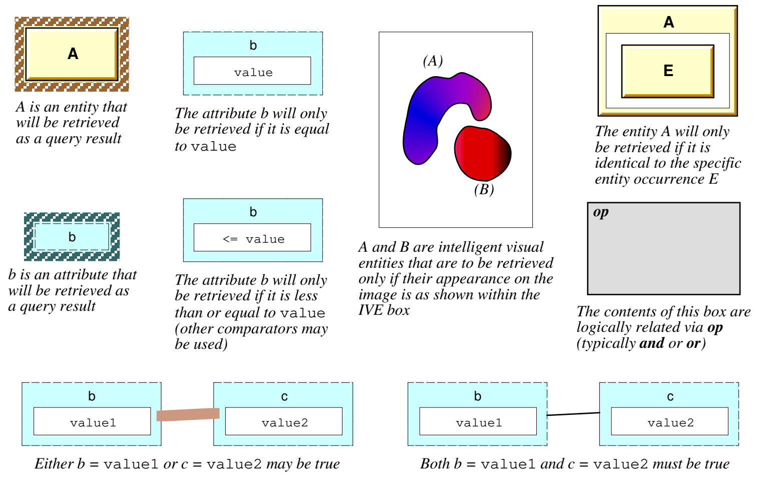

Figure 5: Query-specific notation employed by MQuery when forming visual query specifications.

processed queries. The notion of user freedom is central in current user interface work [36, 37], and is also a fundamental design principle in MQuery.

5.3.1 MQuery Visual Notation

The notation or graphical syntax of an MQuery query specification is nearly identical to that of the M data model that MQuery supports (see Figure 1). However, some notation must be added in order to support certain query-specific constructs, as shown in Figure 5.

The most basic of these constructs is the *predicate box*, which can be invoked from any MQuery object. The contents of an object's predicate box determines a constraint on whether or not a particular instance of that object is to be retrieved. The absence of a predicate box means that no restriction applies — in other words, "retrieve all."

Though the most apparent use of a predicate box is to specify alphanumeric constraints, we do not make any such restrictions. The contents of a predicate box are arbitrarily complex — in fact, it may be an entire instance of the entity or object being constrained. This capability is used by nested queries, described in Section 5.3.3.

Another fundamental query construct, the *result border*, can be seen in Figure 5. The result border is applied to the objects in the query which the user wants returned as the query result. It is analogous to the SELECT clause of an SQL query.

A final query construct that is not present in the schema design portion of MQuery is the *IVE box*. An IVE box is used when invoking the visual query capabilities of our model's intelligent visual

INSERT

Figure 6: MQuery for "Manually add new `Patient`s to the database."

entities (IVEs). When the user wishes to ask a "show me objects that look like this"-type of query, the IVEs involved are selected, their visual appearances are activated, and they are placed within an IVE box to show their relative positioning and sizing. The query processor interprets this box as a query predicate which constrains n-tuples of IVEs (where n is the number of IVEs placed in the box) to those n-tuples that are visually similar to the arrangement in the box.

5.3.2 Insertion Queries

MQuery responds to an insertion query by invoking an `insert` method for the objects to be added. By default, this method displays data entry forms to be filled out by the user. However, `insert` methods may be overridden (or alternatives may be defined) to customize the insertion process for different data types.

In general, the literature does not pay much attention to the act of inserting data to a database; most of the systems in Section 2 assume that the data is already stored in the database, and do not apply visual interface concepts to this phase of database management. However, some work has acknowledged that the latest developments in data modeling have resulted in complicated structures for which insertion is no longer as straightforward [14].

Query 1 *Manually add new* `Patient`*s to the database.*

The query window for Query 1 is shown in Figure 6. This is the query used when a user wishes to add new occurrences of an entity to the system. MQuery responds to this insertion query by presenting the user with a blank `Patient` form, containing fields to be entered by the user.

Query 2 *Manually add a new* `Thermal Ablation` *treatment to the database; the treatment was performed on August 5, 1995 on the* `Patient` *with ID 555-12-12.*

Note in Figure 7 that the constraints in the query spec are specified as if a retrieval query is about to be performed; however, because the user issues an *insert* command instead of a *retrieve* command,

INSERT

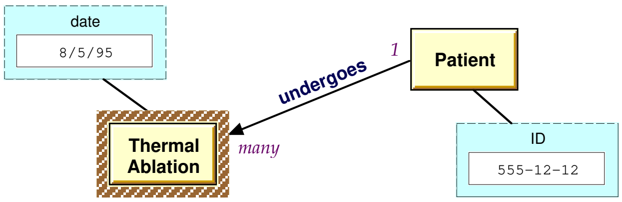

Figure 7: MQuery for "Manually add a new `Thermal Ablation` treatment to the database; the treatment was performed on August 5, 1995 on the `Patient` with ID 555-12-12."

INSERT

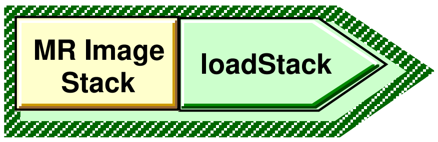

Figure 8: MQuery for "Use the MR Image Stack entity's `loadStack` method to add some image files into the database."

the system responds by adding new data to the database with the given attributes.

Query 3 *Use the MR (magnetic resonance) Image Stack entity's* `loadStack` *method to add some image files into the database.*

Figure 8 illustrates a query where data insertion is not performed manually, as in Query 1, but automatically, using a method that is defined within the entity whose occurrences are to be inserted. In this case, the `loadStack` method requires a filename or a list of filenames from which the `MR Image Stack` entity is to read and insert new data. In the case of our sample medical data model, these files are likely to be PACS (picture archiving and communication system) images from the UCLA Department of Radiological Sciences. When Query 3 is run, the user informs the system of the filenames of the images to be added. The `MR Image Stack`'s `loadStack` method takes care of the data from that point on.

5.3.3 Retrieval Queries

MQuery responds to a retrieval query by retrieving then presenting the requested data to the user. The query results are reusable: they may be placed in another query or used to modify the current query. This nesting of queries and results is applicable to all data types, including complex stream or

Figure 9: MQuery for "Display a list of all the patients who are currently in the database."

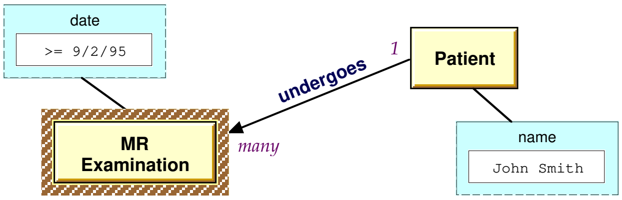

Figure 10: MQuery for "Display all MR exams concerning patient John Smith that are dated September 2, 1995 or later."

multistream structures.

Alphanumeric Queries MQuery contributes to this query category by providing a visual interface that simplifies operations including the equivalents of joins and subqueries.

Query 4 *Display a list of all the patients who are currently in the database.*

Query 5 *Display all MR exams concerning patient John Smith that are dated September 2, 1995 or later.*

Query 4 looks exactly like the insertion Query 1 in its graphical form; the only difference is that in Query 1, the request is "insert this," while in Query 4, the request is "retrieve this." Query 5 appears in Figure 10.

Queries with Multimedia Results

Query 6 *Retrieve radiologic images which contain objects similar to the ones that I will place onscreen, based on size and shape.*

Of note in Query 6, Figure 11: (a) the request to "retrieve images" translates visually into requesting for a set of `MR Image Stacks`, and (b) the actual predicate used to select the desired images is *purely visual* in nature — it is formed by moving and arranging shapes on the screen.

The shapes shown in Figure 11 illustrate how intelligent visual entities (IVEs) are used in practice. These icons represent the `Brain State` and `Lesion State` entities seen in our sample application

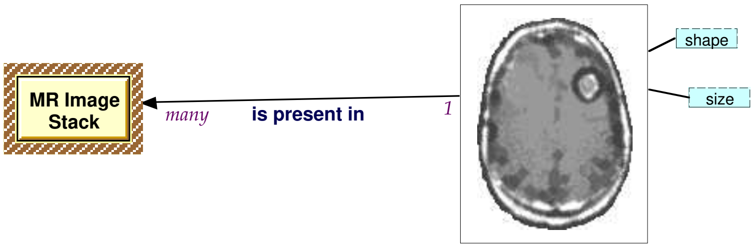

Figure 11: MQuery for "Retrieve radiologic images which contain objects similar to the ones that I will place onscreen, based on size and shape."

schema (Figure 4). In Figure 11, their fully-visual appearance has been activated. The white rectangular area where they reside represents a single visual predicate; in English, that section of the query specifies "an individual brain entity and an individual lesion entity whose appearance is as shown on the screen."

The meaning of "appearance" is determined by the IVEs involved, `Brain State` and `Lesion State`. Each IVE compares the drawn icon to the instances of that IVE in the database by using the image features defined as relevant for similarity comparisons. In Query 6, these features are explicitly set by the user as size and shape (where shape is based on the characteristics of an icon's contour or outline). These attributes are chosen from a menu provided by the IVE. Approaches for processing the query based on different visual attributes are being examined by our group.

When similarity is not explicitly defined by the user, IVEs use defaults set by the database designer. For example, `Brain State` may determine similarity by looking at the area of the brain's outline, while `Lesion State` determines similarity by geometrically comparing the lesion's contours. The distance between the contours' centroids is used as the default metric for their spatial relationship, but this metric, like image features, can be customized for different combinations of IVEs.

Query 7 *Play back voice recordings in images where Dr. Chan recommends chemotherapy.*

Query 8 *What are the radiologic/imaging appearances of a particular pathology?*

For space reasons, we refrain from including full MQuery examples for Queries 7 and 8. In the context of the example queries that have already been illustrated, translating these queries to MQuery query specifications is not difficult, given the appropriate schema.

Noteworthy in Queries 7 and 8 are the use of voice objects as transparently as image objects in Query 7, and, in Query 8, the association of a very abstract object (a given pathology) with images in the database. These show the generality and applicability of the multimedia type model that we

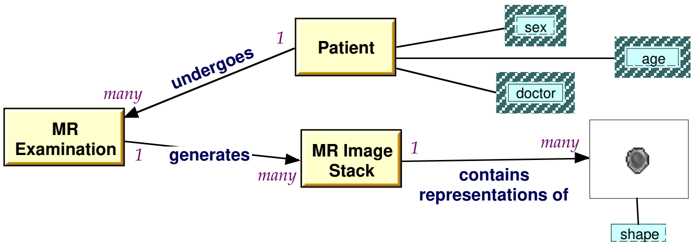

Figure 12: MQuery for "Obtain the sex, age, and doctor of all patients with tumors similar in shape to the tumor currently being viewed."

have defined [33]. A special icon, perhaps for a hypothetical "intelligent audio entity" similar to the IVE defined previously, may be used to visualize voice and sound data. This icon can interact with the user by playing back its encoded sound when double-clicked.

Queries with Multimedia Predicates Multimedia data can be used in MQuery not only as query results but also as participants in the actual predicates.

Query 9 *Obtain the sex, age, and doctor of all patients with tumors similar in shape to the tumor currently being viewed.*

Query 10 *Locate other treated lesions in the database similar with respect to size, shape, intensity, and growth or shrink rate of the current case.*

Query 11 *Does the lesion overlap any of the activated areas from the functional MRI study?*

Figures 12 illustrates the MQuery expression for Queries 9. Query 10 is like Query 9, but takes more image features into account when performing similarity comparisons.

We have implemented Query 9 and only need more knowledge about lesions to implement Query 10. Query 11 is quite forward-looking and requires a thorough model of the brain when examined via functional MRI.

Queries Over Time-Based Data Query 12 does not get "into" the stream structure yet but accesses it as a single overall entity.

Query 12 *Retrieve and play back the thermal ablation simulation results for patient John Smith for any simulation runs performed after January 10, 1996.*

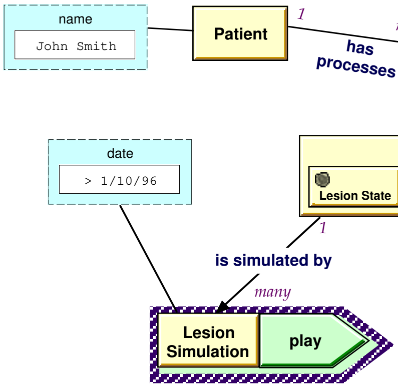

Figure 13: MQuery for “Retrieve and play back the thermal ablation simulation results for patient John Smith for any simulation runs performed after January 10, 1996.”

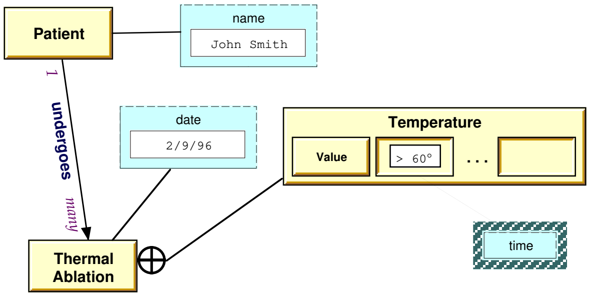

Figure 14: MQuery for “When does the tissue in the lesion being treated for John Smith on February 9, 1996 become greater than 60° C?”

Query 12, Figure 13, illustrates how streams may be accessed as self-contained entities. In Figure 4, `Lesion Simulation` is derived from our simulation model, where a simulation is defined as a set of parallel streams called a *multistream* [33]. The method `play` is invoked from these retrieved simulations, and this is the output that is presented to the user.

Query 13 *When does the tissue in the lesion being treated for John Smith on February 9, 1996 become greater than 60° C?*

The condition (> 60°) in the central element of `Temperature` seeks the stream elements that satisfy this condition; the `time`s attached to those elements are returned. The attribute `time`, by definition, is an attribute of any entity that serves as a stream element (as seen in Figure 3), and so it is not explicitly mentioned in the sample schema in Figure 4.

Query 14 illustrates the use of streams for making comparisons over time. In Figure 4, the `Brain` and `Lesion` streams have internal stream elements called `Brain State` and `Lesion State`, respec-

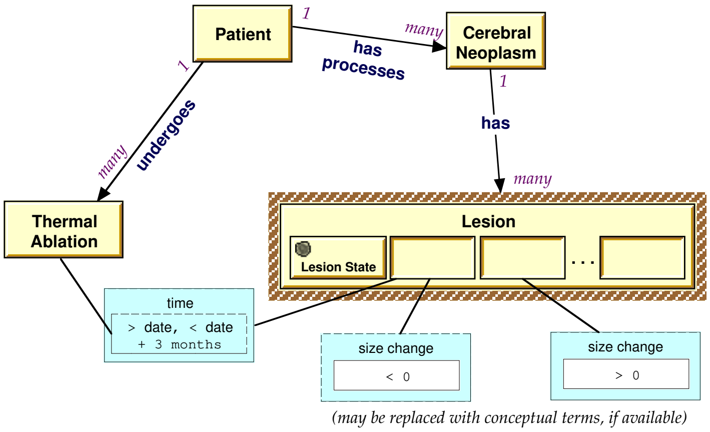

Figure 15: MQuery for "Find all cases in which a tumor decreased in size for less than three months post treatment, then resumed a growth pattern after that period."

tively. MQuery's time-based querying is based on this relationship between the overall stream and its individual elements.

Query 14 does not call upon the IVEs' visual characteristics, but is more complex in terms of the way it uses time in its predicate. Figure 15 illustrates the MQuery expression for Query 14.

Query 14 *Find all cases in which a tumor decreased in size for less than three months post treatment, then resumed a growth pattern after that period.*

Specifying the change in size is done by specifying the desired range on the size change attribute[1], as seen in Figure 15. To express the time constraints, we call upon the built-in time attribute of the `Lesion States`: we assign, to one box, the constraint that its time stamp must be within three (3) months of the date of the patient's thermal ablation therapy. The inclusion of `Patient` assures that the thermal ablation treatments and lesions that are linked all belong to the same patient; without the links provided by `Patient`, any thermal ablation treatment may be matched with the lesion, regardless of their respective patients.

There is no need for an additional time predicate for the second box — by its position, it is *implicitly expected to occur after any elements that satisfy the first box*. Thus, only the size change > 0 is required; the condition that this growth pattern occurs after the three-month period is already implied by the positioning of the second box to the right of the first one. The time constraint is

[1]Although this particular value is modeled as an attribute, it may internally be implemented as a method; however the user does not need to be aware of this.

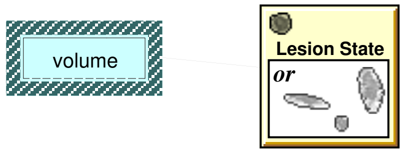

Figure 16: MQuery for "What are the volumes of the tumors that were retrieved in the previous query?"

therefore translated from an alphanumeric predicate (such as "time $<$ date $+$ 3 months") to a visual one, communicated entirely by the relationship between the stream elements.

Alternatively, size change may be queried on a conceptual level (i.e. size change is `stable` or `shows little change`) if a knowledge base mapping these concepts to values or value ranges is a part of the system.

Nested Queries These queries show how MQuery's integrated modules make it simple to pass the results of one query into another. This capability is made possible by integrating output and visualization as a component of the overall MQuery system. Thus, MQuery is "aware" of the windows within which query results are displayed, and can copy or retrieve the objects from those windows.

Of the queries listed below, we provide MQuery expressions for Queries 15 and 16. Queries 17 does not have an MQuery expression because it is not based on the sample schema presented in Figure 4.

Query 15 *What are the volumes of the tumors that were retrieved in the previous query?*

Query 16 *Where and when does maximum tissue heating take place for the simulation run that is currently on display?*

Query 17 *List other cases that have a meningioma of similar size to the case currently being viewed.*

Figures 16 and 17 present MQuery expressions for Queries 15 and 16. Query results in other windows are re-used in new queries by using copy-paste or drag-and-drop. The operation is analogous to query construction, where objects from a *schema* window are copied then pasted into a query window.

As can be seen in the figures, nested queries are achieved by replacing the contents of an entity's predicate box with one or more specific entity occurrences, thus naturally extending the more familiar functionality of placing an alphanumeric constant or comparison in an attribute's box. Figure 16 is particularly interesting because it shows how IVEs can also be used in the same manner; note the

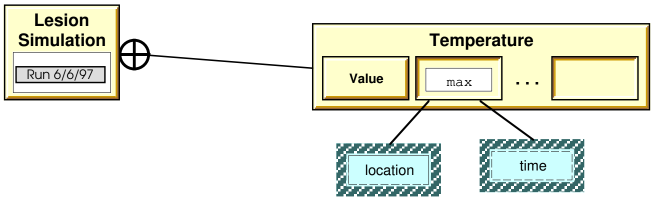

Figure 17: MQuery for "Where and when does maximum tissue heating take place for the simulation run that is currently on display?"

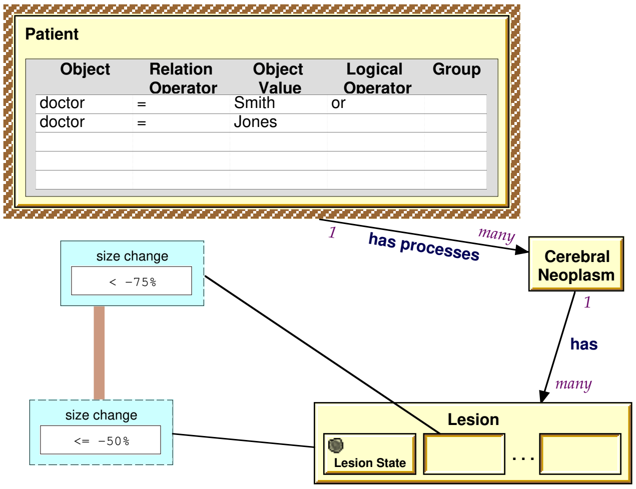

Figure 18: MQuery for "Find patients treated either by Dr. Smith or Dr. Jones whose primary lesions exhibit a decrease in size by at least 50% for every examination since baseline, or have at least one examination that exhibits a decrease in size by greater that 75%."

multiple `Lesion State` objects inside the predicate box, indicating a set of `Lesion State`s that have been copied from query results presumably on display elsewhere on the screen.

Queries With Multiple Predicates

Query 18 *Find patients treated either by Dr. Smith or Dr. Jones whose primary lesions exhibit a decrease in size by at least 50% for every examination since baseline, or have at least one examination that exhibits a decrease in size by greater that 75%.*

Figure 18 shows how compatibility with PICQUERY+ is achieved in MQuery. A PICQUERY+ table is used for `Patient` instead of an entity rectangle; this is done because a table interface permits a clear, line-by-line listing of the predicates to be applied to its associated entity.

DELETE

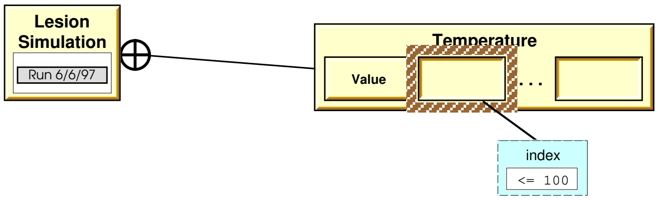

Figure 19: MQuery for "Delete the first 100 simulation data points on the simulation currently on display."

5.3.4 Deletion Queries

Entity deletion deletes all of its attributes and any relationships or aggregations pointing to that entity. MQuery does not recursively delete an object if it participates in chains of relationships. Instead, the deletion stops at "level one" — the entity itself, and the association, are eliminated, but not the node on the other end of the association.

Query 19 *Delete the first 100 simulation data points on the simulation currently on display.*

Query 19, Figure 19, may come into play if the validity of a simulation model is being tested, and the first 100 points were discovered to have anomalies that would skew the results of the simulation. The index attribute, which is used to determine a stream element's position within the overall stream, is a built-in attribute of any object that participates as a stream element [33].

5.3.5 Update Queries

For update queries, entry forms are opened by MQuery showing the current values of the objects designated as a "query result." The user then enters the new value for that object. MQuery permits a minor variant that increases the degree of automation in a modification query: if the object with a result border is also a *predicate box* then the value within the predicate box is used as the new, modified value for that object.

Query 20 *Change the ID photograph of Mr. Jones to the one currently shown onscreen, and replace his timeline image slices for October 10 and November 10 with slice numbers 12 and 14, respectively.*

In Figure 20, a photograph already on display is capable of providing an iconic form of itself for use in the query. This iconic form is a built-in function of any entity that is modeled in the M data model as the `Image` multimedia type.

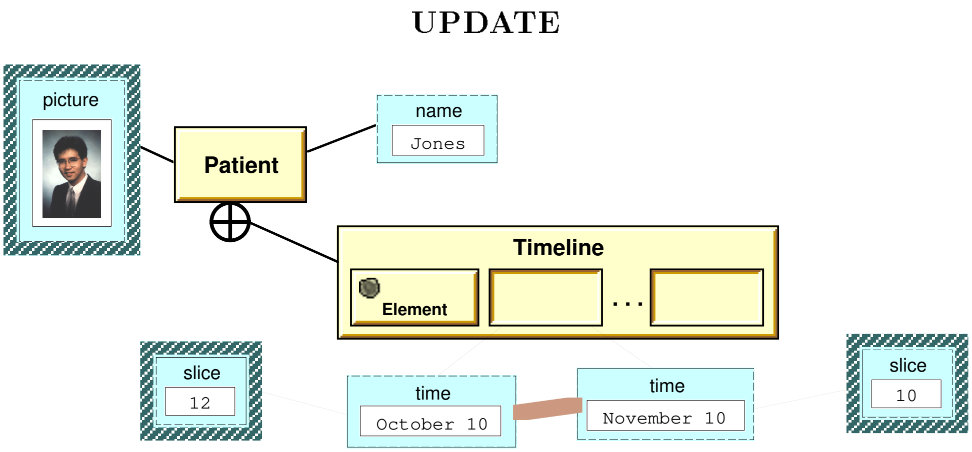

Figure 20: MQuery for "Change the ID photograph of Mr. Jones to the one currently shown onscreen, and replace his timeline image slices for October 10 and November 10 with slice numbers 12 and 14, respectively."

`Timeline` is a multistream — an aggregation of one or more streams, in this case `Brain` and `Lesion`. A multistream can be manipulated as if it were a simple stream whose elements are synchronized aggregations of the component streams. Thus, in Figure 20, `Timeline` looks like a simple stream, and the predicates attached to its elements *are applied to all of the elements of its component streams.*

In addition, an **or** operator is required, because the query is actually performing *two* unrelated modification queries. Thus, we have the dual predicate boxes attached to the same stream element, but marked with the thick line representing an **or** operation.

5.4 Output Presenter & Visualizer

The most significant features of MQuery's presentation and visualization module include the following:

- *Support for a wide range of data types.* MQuery's presentation and visualization module aims to support a wider array of data types than previously found in related work. This includes standard alphanumeric data, complex relationships, multimedia data such as digital audio and video, and scientific data such as simulations, electrocardiograms, Doppler ultrasound, etc. This broad functionality will be achieved through a modular architecture where new display methods can be "plugged in" based on a well-defined specification or protocol.

- *Intelligent visualization of retrieved data.* When data needs to be displayed to the user, the output module first checks to see whether the database's administrator or some other user has designed a custom method for displaying the data. If found, it is used to perform the display; if multiple methods are found, the user is presented with a menu of available choices. If no custom methods are found, a default `display` method dynamically assembles a display window for the query results; the display generated will be based on rules and heuristics that guide the way different types of data are to be visualized.

- *Unified interface for all display needs.* Whether the user examines data from the schema window, inserts new data, or views data from the query window, MQuery's presentation/visualization package uses the same interface. Even data insertion displays the same windows used for data retrieval; the only difference is that insertion queries generate *blank* displays that can be filled out.

- *Full integration with query module.* The interaction between query results and the actual query expression is, unlike most visual query systems, fully bidirectional. Similar to nested queries in SQL, MQuery results can be sent back into other queries to function either as subqueries or new values for comparison.

5.5 Implementation and User Evaluation

A subset of MQuery has been implemented in the KMeD system using VisualWorks [38] and Gemstone [39]. Our testbed database currently resides on a central server, where images, reports, and patient data are copies of data that reside in a distributed, heterogeneous environment consisting of PACS, the Radiology Information System (RIS), and the Hospital Information System (HIS) [40]. A fuller version of MQuery, exhibiting much of the functionality in this paper, is in progress. Image and time-based query processing is planned using knowledge-based techniques developed at a parallel research project at UCLA. Because of the broad functionality introduced in MQuery, execution speed and/or efficiency is not a major focus in the initial implementation.

After a full MQuery prototype has been implemented, we will execute a user testing and validation phase that will include students, professors, and practitioners of computer science and radiology. Our initial experience has shown that, after a brief tutorial describing the model's notation, potential database users have found M's schema diagrams to be very readable and straightforward to understand.

6 Conclusions and Future Work

In this paper, we described MQuery, which provides the following new constructs: an overall multimedia query language that accesses multimedia types in a uniform manner, intelligent visual entity querying on appearance without depending on alphanumeric indexing, and query constructs for time-based information.

All told, the query language concepts introduced in MQuery are applicable to a wide variety of domains: multimedia digital libraries, medical imaging, medical records (via a visual timeline), engineering or scientific simulations, etc. Further, this broad set of domains is efficiently served by a small number of concepts and ideas, particularly multimedia types and streams.

Future work on MQuery includes completion of the implementation and prototype testing of its query language and output modules. Other research directions for M and MQuery include advanced visualization, and the inclusion of hypertext or hypermedia data to the overall model.

A Acknowledgments

The authors would like to thank the many colleagues, collaborators, and consultants whose contributions helped make MQuery into the powerful query language presented in this paper.

- Wesley W. Chu from the Computer Science Department and Ricky K. Taira from the Department of Radiological Sciences are the co-principal investigators of the UCLA KMeD project, which motivated the development of M.

- Denise R. Aberle, Gary R. Duckwiler, Jonathan Goldin, Robert B. Lufkin, Michael F. McNitt-Gray, and Fernando Viñuela have been invaluable in developing the database requirements of real-world medical applications such as cerebral aneurysm embolization, thermal ablation therapy, and thoracic oncology imaging.

References

[1] Ramesh Jain, editor. *NSF Workshop on Visual Information Management Systems*, February 1992.

[2] Wesley W. Chu, Alfonso F. Cardenas, and Ricky K. Taira, editors. *AAAS Workshop on Advances in Data Management for the Scientist and Engineer*, Boston, Massachusetts, February 1993. National Science Foundation.

[3] Y. Anzai, R. B. Lufkin, A. DeSalles, D. R. Hamilton, K. Farahani, and K. L. Black. Preliminary experience with MR-guided thermal ablation of brain tumors. *American Journal of Neuroradiology*, 16(1):39–48, January 1995. Discussion on pp. 49–52.

[4] W. W. Chu, I. T. Ieong, R. K. Taira, and C. M. Breant. A temporal evolutionary object-oriented data model and its query language for medical image management. In Li-Yan Yuan, editor, *Proceedings of the 18th International Conference on Very Large Databases*, pages 53–64, Vancouver, Canada, August 1992. Very Large Data Base Endowment, Morgan Kaufmann Publishers, Inc.

[5] Wesley W. Chu, Alfonso F. Cardenas, and Ricky K. Taira. KMeD: A knowledge-based multimedia medical distributed database system. *Information Systems*, 20(2):75–96, 1995.

[6] J. Michael Pratt and Maxine Cohen. A process-oriented scientific database model. *SIGMOD Record*, 21(3):17–25, September 1992.

[7] Department of Health and Human Services, National Institutes of Health, National Library of Medicine. *UMLS Knowledge Sources*, August 1992.

[8] Lil Mohan and R. L. Kashyap. A visual query language for graphical interaction with schema-intensive databases. *IEEE Transactions on Knowledge and Data Engineering*, 5(5):843–858, October 1993.

[9] Michael Kuntz and Rainer Melchert. Pasta-3's graphical query language: Direct manipulation, cooperative queries, full expressive power. In Peter M. G. Apers and Gio Wiederhold, editors, *Proceedings of the 15th International Conference on Very Large Databases*, pages 97–105, Amsterdam, The Netherlands, August 1989. Very Large Data Base Endowment, Morgan Kaufmann Publishers, Inc.

[10] Thomas Joseph and Alfonso F. Cardenas. PICQUERY: A high level query language for pictorial database management. *IEEE Transactions on Software Engineering*, 14(5):630–638, May 1988.

[11] Alfonso 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.

[12] Arturo Pizano, Allen Klinger, and Alfonso F. Cardenas. Specification of spatial integrity constraints in pictorial databases. *Computer*, pages 59–71, December 1989.

[13] Amarnath Gupta, Terry Weymouth, and Ramesh Jain. Semantic queries with pictures: the VIMSYS model. In Guy M. Lohman, Amilcar Sernadas, and Rafael Camps, editors, *Proceedings of the 17th International Conference on Very Large Databases*, pages 69–79, Barcelona, Spain, September 1991. Very Large Data Base Endowment, Morgan Kaufman.

[14] Deborah Swanberg, Chiao-Fe Shu, and Ramesh Jain. Knowledge guided parsing in video databases. In Keith T. Knox and Edward Granger, editors, *IS&T/SPIE's Symposium on Electronic Imaging: Science & Technology*, San Jose, California, USA, January–February 1993. The Society for Imaging Science and Technology (IS&T) and The International Society for Optical Engineering (SPIE).

[15] Alberto Del Bimbo, Maurizio Campanai, and Paolo Nesi. A three-dimensional iconic environment for image database querying. *IEEE Transactions on Software Engineering*, 19(10), October 1993.

[16] Toshikazu Kato. Database architecture for content-based image retrieval. In Albert A. Jamberdino and Wayne Niblack, editors, *Image Storage and Retrieval Systems*, pages 112–123, San Jose, California, February 1992. SPIE — The International Society for Optical Engineering; IS&T — The Society for Imaging Science and Technology, Proc. SPIE.

[17] Wayne Niblack, R. Barber, W. Equitz, Myron Flickner, E. Glasman, D. Petkovic, P. Yanker, C. Faloustos, and G. Taubin. The QBIC project: Querying image by content using color, texture, and shape. In Wayne Niblack, editor, *Storage and Retrieval for Image and Video Databases*, pages 173–187, San Jose, California, 1993. SPIE.

[18] M. M. Zloof. Query-by-example. In *Proceedings of the National Computer Conference*, pages 431–437, Arlington, VA, May 1975.

[19] Jan Paredaens, Jan Van den Bussche, Marc Andries, Marc Gemis, Marc Gyssens, Inge Thyssens, Dirk Van Gucht, Vijay Sarathy, and Lawrence Saxton. An overview of GOOD. *SIGMOD Record*, 21(1):25–31, March 1992.

[20] Michele Angelaccio, Tiziana Catarci, and Giuseppe Santucci. QBD*: A graphical query language with recursion. *IEEE Transactions on Software Engineering*, 16(10):1150–1163, October 1990.

[21] Mariano P. Consens and Alberto O. Mendelzon. GraphLog: A visual formalism for real life recursion. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 404–416, Nashville, April 1990. Association for Computing Machinery, ACM Press.

[22] Yannis E. Ioannidis, Miron Livny, and Eben M. Haber. Graphical user interfaces for the management of scientific experiments and data. *SIGMOD Record*, 21(1):47–53, March 1992.

[23] Victor M. Markowitz and Arie Shoshani. Data management tools for scientific applications: An overview. Technical report, Lawrence Berkeley Laboratory, February 1993.

[24] Daniel Bryce and Richard Hull. SNAP: A graphics-based schema manager. In *IEEE International Conference on Data Engineering*, pages 151–164, Washington, D.C., February 1986. IEEE Computer Society, IEEE Computer Society Press.

[25] Harry K. T. Wong and Ivy Kuo. GUIDE: a graphical user interface for database exploration. In *Proceedings of the Eighth International Conference on Very Large Data Bases*, pages 22–32, Mexico City, September 1982. Very Large Database Endowment.

[26] Roger King and Stephen Melville. Ski: A semantics-knowledgeable interface. In Umeshwar Dayal, G. Schlageter, and Lim Huat Seng, editors, *Proceedings of the Tenth International Conference on Very Large Data bases*, pages 30–33, Singapore, August 1984. Very Large Database Endowment.

[27] Jakob Nielsen. Noncommand user interfaces. *Communications of the ACM*, 36(4):57–71, April 1993.

[28] Rui Zhao. Incremental recognition in gesture-based and syntax-directed diagram editors. In Stacey Ashlund, Kevin Mullet, Austin Henderson, Erik Hollnagel, and Ted White, editors, *Proceedings of INTERCHI 1993*, pages 95–100, Amsterdam, The Netherlands, April 1993. Association for Computing Machinery.

[29] Lisa J. Stifelman, Barry Arons, Chris Schmandt, and Eric A. Hulteen. VoiceNotes: A speech interface for a hand-held voice notetaker. In Stacey Ashlund, Kevin Mullet, Austin Henderson, Erik Hollnagel, and Ted White, editors, *Proceedings of INTERCHI 1993*, pages 179–186, Amsterdam, The Netherlands, April 1993. Association for Computing Machinery.

[30] Didier Bardon. Management of color usage in dynamic mapping environments: Balancing semantics, visual ordering, and discernability. In T. Catarci, M. F. Costabile, and S. Levialdi, editors, *Proceedings of the International Workshop Advanced Visual Interfaces*, pages 50–67, Rome, Italy, May 1992. World Scientific Publishing Co.

[31] George G. Robertson, Stuart K. Card, and Jock D. Mackinlay. Information visualization using 3D interactive animation. *Communications of the ACM*, 36(4):57–71, April 1993.

[32] Hideki Koike. An application of three-dimensional visualization to object-oriented programming. In T. Catarci, M. F. Costabile, and S. Levialdi, editors, *Proceedings of the International Workshop Advanced Visual Interfaces*, pages 180–192, Rome, Italy, May 1992. World Scientific Publishing Co.

[33] John David N. Dionisio and Alfonso F. Cardenas. A unified data model for representing multimedia, timeline, and simulation data. To appear, 1996.

[34] Denise R. Aberle, John David N. Dionisio, Alfonso F. Cardenas, Ricky K. Taira, Wesley W. Chu, Michael F. McNitt-Gray, and Jonathan Goldin. A unified timeline model for multimedia medical databases. *Radiographics*, 16(3):669–681, May 1996.

[35] John David N. Dionisio, Alfonso F. Cardenas, Ricky K. Taira, Denise R. Aberle, Wesley W. Chu, Michael F. McNitt-Gray, Jonathan Goldin, and Robert B. Lufkin. A unified timeline model and user interface for multimedia medical databases. *Computerized Medical Imaging and Graphics*, 1996. To appear.

[36] Ben Shneiderman. *Designing the User Interface*. Addison Wesley, Reading, Massachusetts, second edition, 1992.

[37] Donald A. Norman. *The Design of Everyday Things*. Doubleday Currency, New York, paperback edition, 1988. Previously published as *The Psychology of Everyday Things*.

[38] ParcPlace Systems, Inc., Sunnyvale, CA. *VisualWorks Release 2.0 User's Guide, Cookbook, and Object Reference*, 1994.

[39] Servio Corporation. *GemStone Programming Guide, GemStone Version 4.0*, June 1994.

[40] A. F. Cardenas, R. K. Taira, and W. W. Chu. Integration and interoperability of a multimedia medical distributed database system. *IEEE Data Engineering*, 16:43–47, March 1993.