A Unified Data Model for Representing Multimedia, Timeline, and Simulation Data

John David N. Dionisio and Alfonso F. Cárdenas

Abstract—This paper describes a unified data model that represents multimedia, timeline, and simulation data utilizing a single set of related data modeling constructs. A uniform model for multimedia types structures image, sound, video, and long text data in a consistent way, 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 represented using a construct called a stream. Streams can be aggregated into parallel multistreams, thus providing a structure for viewing 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. Prototypes based on the model have been implemented for two medical application domains: thoracic oncology and thermal ablation therapy of brain tumors. Sample schemas, queries, and screenshots from these domains are provided. Finally, a set of examples is included for an accompanying visual query language discussed in detail in another document.

Index Terms—Multimedia database management, multimedia data model, timeline, simulation, visual data modeling, multimedia data streams, temporal databases, visual querying, multimedia querying, medical multimedia databases.

1 Introduction

This paper describes a unified data model that we shall call M for representing multimedia, timeline, and simulation data. Our current research work in the Knowledge-Based Multimedia Medical Distributed Database (KMeD) project at UCLA has identified a number of data modeling needs that come from different application domains. Our investigation concluded that they can all be served using a single construct called a *stream*. Streams have been described in the literature, although within a narrower scope [25], [4], [39]. This paper describes how we have been able to generalize the *time-based stream* model so that it can fulfill the requirements of such diverse domains as simulation and validation, medical timelines, and multimedia visualization.

1.1 Background

Recent developments in scientific databases have identified new data modeling requirements for next-generation scientific databases. These developments involve areas which at first glance seem to be distinct and disjoint:

- *Multimedia.* In its fullest and most complete form, scientific data is multimedia in nature. It is fully visual, frequently three-dimensional, and spans the dimension of time [32]. Much work has focused in this area, and our work proceeds in a similar direction as in [31], [38], [12], [27].
- *Simulation and validation.* Harreld [27] and Anzai et al. [3] are evidence of an increasing need to integrate simulation and database technologies. One potential benefit of this integration is the capability to validate simulation data by direct comparison to real-world, measured data.
- *Timelines.* Temporal, evolutionary, and process data models [15], [13], [35] have many potential applications in science and medicine. Much of medicine involves tracking the progress and history of a patient over time, while 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, skeletal development, or other natural process as a series of still frames. If properly registered, timelines may be viewed as a short movie or animation clip.

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

- *The element of time.* The above application domains all require data that changes over time: digital video, simulation data, timelines, etc. A data model supporting these application domains must be able to represent, query, and visualize this element of time. Better yet, this element of time must be captured in a uniform construct regardless of the time scale or data type.
- *Complex data structures.* Scientific data domains involve objects with complex structures and interrelationships with other objects. In the medical domain, for example, the Unified Medical Language System [23] defines a large and complex semantic network of objects, processes, subjects, synonyms, and relationships. Current relational data models do not easily capture and present such complex data without using artificial or arbitrary keys. A data model that

• *The authors are with the Computer Science Department, School of Engineering and Applied Science, University of California at Los Angeles, 3732 Boelter Hall, Los Angeles, CA 90095-1596. E-mail: dondi@itmedicine.net, cardenas@cs.ucla.edu.*

Manuscript received 27 Dec. 1995; revised 7 Aug. 1996.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 104493.

takes this structural complexity to a higher level is thus required.

- *Multiple representations of objects.* The multimedia data generated by a scientific application domain results in the representation of any individual object in multiple data areas. For example, a tumor mass may be visible in multiple CAT scans and MRI scans. It may be mentioned in multiple lab or radiological reports, annotations, or voice transcriptions. Finally, it may be represented by numerical or tabular simulation data. A conventional database is capable of storing most of these data types, but a great deal of work is required to ensure that the word "tumor" or any concepts relating to it lead to any or all of these means of storage, and vice versa. Our data model attempts to resolve these multiple representations by making this concept a fundamental element of the model.

We address these needs by extending and integrating a number of data models already in the literature, as discussed in Section 2.

1.2 Contents of the Paper

After summarizing our model's primary contributions to the field in Section 1.3 and comparing related work in Section 2, we proceed with a review of the basic data modeling concepts upon which our model builds in Section 3.1.

Section 3.2, Section 3.3, Section 3.4, and Section 3.5 describe the focus of our new model: generalized constructs for representing various kinds of multimedia and time-based data. Section 4 outlines the application domains for which we intend to test its functionality. Section 5 and Section 7 provide a hint of things to come by giving an overview of the accompanying visual query language which we are designing.

1.3 Primary Contributions

The primary contributions of our data modeling work to the field can be described as follows:

- *Generalized multimedia types.* We define a general framework for modeling and accessing various kinds of multimedia data, including images, sounds, long text, digital video, and integrated timelines.
- *Generalized model for representing most kinds of time-based data.* In our data model, we have applied a time-based stream model that is common in multimedia work to other application domains. In particular, we use time-based streams to model evolutionary, simulation, and/or timeline data. Collaborative work with radiologists and clinicians, where prototype software using our generalized model was developed, has shown that this unified approach to representing time-based data results in greater ease of comprehension and specification of data models containing time-based entities [2].

Section 3.2, Section 3.3, Section 3.4, and Section 3.5 describe our data model, focusing on the constructs that specifically represent these contributions. We have asked users to both interpret and create different schema diagrams using our data model. These tests have shown that the new constructs make it easier for users to specify and express the multimedia and time-based components of a database.

2 Previous Work

We compare our data model to three kinds of data models that exist today:

1) multimedia data models
2) general data models, and
3) multimedia file formats.

For each data model category, major models have been selected for comparison with our data model.

2.1 Multimedia Data Models

A number of data models that address the structure and semantics of multimedia data have been developed. These multimedia data models emphasize the modeling of images, sound, or video data. This approach simplifies the overall problem, because it assumes that a database consists only of this type of data. To our knowledge, only VIMSYS provides for a system that exists within the context of a broader database application, although emphasis remains within the realm of image data modeling [26].

Our work extends the modeling approach of the University of Geneva [25] with other general database modeling concepts, and also adds a query language. In addition, the major application of simulation and scientific computing extends the timed stream model to cover new areas such as interpolation and simulation validation. The Amsterdam Hypermedia Model [28] is extended in our model by adding a general query language and explicit support for simulation and scientific data. In addition, our work focuses on multimedia databases, and not on multimedia hypertext documents.

The M data model is also a superset of the capabilities of spatio-temporal logic [7] because it integrates spatial and temporal semantics with more general database constructs. Support for sound and simulation data is also added. Finally, we combine the multimedia modeling of time-based streams with the higher database functionality of VIMSYS [26], and add explicit support for simulation and scientific computing to both areas.

A number of more recent models have begun to tackle the modeling of digital video data. OVID [34], VideoMAP [40], and the concept of moving icons or "micons" [41] focus on digitized video and their semantics. The M data model applies the video-oriented ideas of OVID, VideoMAP, and micons within its definitions wherever they are appropriate.

2.2 General Data Models

Many current data models, though not explicitly directed toward multimedia applications, have a number of constructs that will nonetheless support current multimedia requirements. M uses the extended entity-relationship data model (EER) [33] as a basis for basic database concepts such as entities, relationships, and attributes. In addition, it adds new modeling constructs such as temporal and

evolutionary relationships, a more explicit implementation of aggregation, and methods. Most importantly, multimedia modeling is incorporated into EER by this work.

The spatial temporal-evolutionary data model (SEDM) [15], [29] focuses primarily on semantic constructs over still images. Our model adds a stream construct that models time *within* an entity. Further, support for simulation data models is also added.

The object-oriented predicate calculus (OOPC) [6] provides a comprehensive model that is applicable to most of the object-oriented data models and database management systems that are present today. The M data model combines the visual effectiveness of EER with the rich constructs of OOPC to achieve a hybrid model with the advantages of both EER and OOPC. M's multimedia types, streams, and simulation modeling also serve as further enhancements to the basic framework established by OOPC.

The M data model provides the multimedia data integration in Jasmine [30] in addition to a rich multimedia model on a par with stand-alone models. Sound and video are modeled and integrated, and simulation and scientific data modeling constructs are added.

2.3 Multimedia File Formats

Multimedia file formats, though not formal data models, remain important to the field because they represent the most widespread and standardized means of distributing multimedia information. The steep requirements of multimedia data storage and management have given these file formats a degree of sophistication not previously found with simpler data types such as relational tables or text files.

The Moving Picture Experts Group (MPEG) [24] and QuickTime [4] formats are widely used for storing video data. They can be used as underlying storage formats for our higher-level models of digital video.

3 Data Model

Because M is derived from a rich heritage of previous data modeling work, many of its fundamental notions correspond to current entity-relationship and object-oriented data models.

It should be noted that the M data model is an evolution of previous data modeling work by our research group. It functions as a superset of the constructs described before. This work includes the image stack data model, presented in [10].

Further, our stream data model interacts with the temporal evolutionary data model, or TEDM [15]. As will be discussed in Section 3.4 and Section 3.5, the combination of TEDM and our multimedia and stream models results in a set of constructs that can represent almost any type of temporal data, at varying logical levels.

3.1 Basic Concepts

The basic constructs of our model in graphical notation are shown in Fig. 1. They define the overall framework of the data model, synthesized from various data models such as those in [16], EER [33], OOPC [6], and Jasmine [30].

In general, our basic 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. In later sections, we extend the model further with our own new constructs.

Databases and Knowledge Bases. A *database* is *a set of entities*. Each entity consists of *attributes*, *relationships*, and *methods*. Attributes and relationships describe the structure, content, and connectivity of an entity, while methods describe the behavior of that entity.

A *knowledge base* is *a structured repository of information, similar to a database in function but differing in content*. In general, knowledge bases contain stable or constant information that can be used as reference or resource material. It may be linked to the entities and other components of a database. Knowledge bases may be viewed as information resources that describe or characterize the information stored in a database. A knowledge base can take on many representations and formalisms. In the specific context of our research, we have focused on a *type abstraction hierarchy* representation of knowledge (see below).

Entities. An *entity* is *the database representation for a particular object in the real world, generally viewed as the "nouns" of a database*. It is notated as a labeled rectangle. If a data model diagram contains nonspecific entities (i.e., rectangles that may represent different kinds of entities), the label within the entity rectangle is italicized.

Inheritance is *the derivation of an entity from another entity*. The inheriting entity (called a *subclass*) possesses the attributes, relationships, and methods of its *superclass*(*es*). Note that inheritance is really a special relationship called *is a* between the superclass and subclass entities. Our version of inheritance is semantically identical to the notion of inheritance found in today's object-oriented systems. It is notated as a double arrow, with arrowheads pointing toward the *subclass* entity.

Aggregation is *the composition of one or more entities as components of another entity*. It is generally used to express a *consists of* relationship. In Fig. 1, we may interpret A as being made up of an occurrence of B and C. A is the *aggregate* and B and C are its *components*. Aggregation is notated with a line that is connected to a circumscribed cross (⊕) at the aggregate. No arrowhead is drawn at either the aggregate or the components.

Methods are *encapsulated modules of code that operate within the context of a particular entity*. Our model supports the specification of methods (or *functions*) for the different entities in the database. They are defined and used in this system in the same way that they are used in most object-oriented paradigms and can be applied in ways that will be familiar to those who use such systems:

- derivation of on-the-fly, calculated attributes,
- encapsulation of complex, highly domain-specific predicates (i.e., a *similarity* function that is tailored to a particular entity), and
- performing of nontrivial processing operations that require a general programming language.

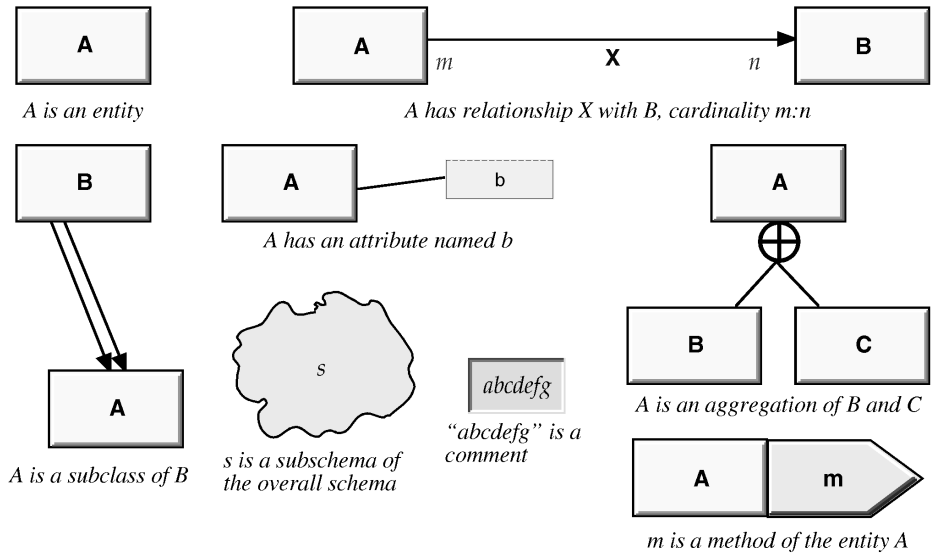


Fig. 1. Notational conventions for basic data model constructs.

In M, methods consist of a name, a set of named parameters and, if defined, a return value. Methods with or without an associated entity are supported. Methods that are associated with an entity can only be invoked from within the context of such an entity. This entity can be viewed as the *performer* of that method.

Relationships. *Relationships* are *constructs that define how different types of entities interact with each other*. They are named using verb expressions (i.e., *is a*, *contains objects*, etc.), and are notated as arrows between the participating entities. The diagram is read such that the arrowhead points to the receiver of the relationship: when "A contains objects B," the arrowhead points to the entity represented by B.

The degree or cardinality of the relationship among entities may be *one-to-one*, *one-to-many*, or *many-to-many*, and may involve any number of entities.

Attributes. An *attribute* is *a unit of information that qualifies or describes the entity to which it is attached*. It is notated as a dotted rectangle whose label states the attribute's name.

Attributes may be *atomic* or *complex*. Atomic attributes are thought of as the "basic data types" within the system: numbers, characters, strings, etc. In addition, images and audio data are also considered atomic in this work.

Complex attributes are attributes whose values are, in themselves, also entities within the database. They can also be thought of as nested entities. Thus, these attributes may have attributes, relationships, and methods of their own.

Attribute atomicity or complexity is not explicitly expressed by our notation. Further, our notation does not require that the data type of an attribute be set by the schema, nor does it indicate whether the attribute is stored or calculated by a method. Our intention in this design choice is to provide the user with as simple a view of the data as possible, without burdening him or her with details or restrictions such as data types, means of storage or calculation, etc.

Subschemas. A *subschema* is *a logical subset of a database's components that can be encapsulated as a single database node*. The size of a database schema may exceed the size of the medium on which it is presented (a piece of paper, a window on the screen, etc.). In this case, the schema may be broken up into logical pieces. Connections across schema portions may be indicated by a dotted curve, with the appropriate arrows or lines indicating how the current schema is connected to schemas on another page or screen.

Knowledge. *Knowledge* is *information that is stored not in the database, but in the knowledge base*. Thus, knowledge is precisely the information that was previously described as stable or constant, and usable as reference or resource material. For this work in particular, knowledge is used to describe or organize the data present in the system.

Within the specific context of our group's research, we focus on *type abstraction hierarchies* as our primary knowledge base construct. Full details on the use of type abstraction hierarchies and their role in cooperative query processing may be found in other publications [11], [17], [8]. More advanced forms of type abstraction hierarchies, called *multidimensional type abstraction hierarchies*, are used to perform query processing of spatial and similarity predicates. These are explored in [9].

Comments. A *comment* is *descriptive text that does not affect the structure or content of the database*. Comments are generally used to explain or clarify, for a human reader, certain portions of a database schema. They are notated as graphical boxes of italicized text. In an actual schema design environment, it may be useful to make the contents of these comments searchable by the user, to pinpoint sections of interest in the schema.

3.2 Multimedia Types

A *multimedia type* is defined as *an entity whose content is most meaningful when it is displayed or presented in a way that is directly perceivable by the human senses*. Images, for example, are seen; audio is heard; video is both seen and heard. These

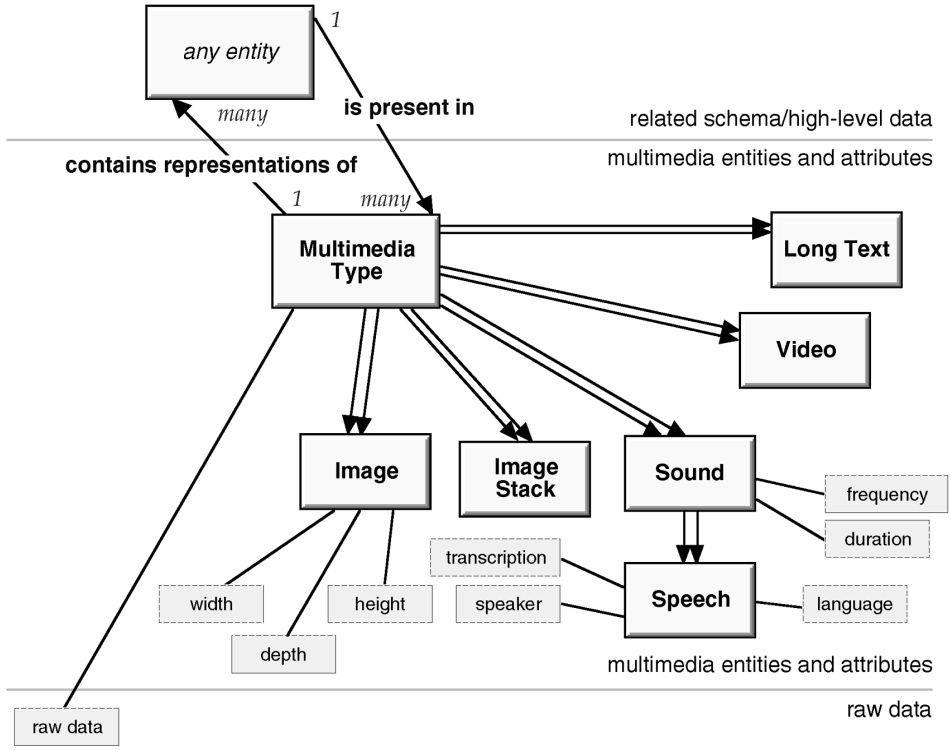


Fig. 2. Overall structure of a multimedia type, showing the layers which they occupy.

special entities form the foundation for the multimedia functionalities in this data model.

3.2.1 Structure of Multimedia Types

Multimedia types can be 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 predefined 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.

If desired, multimedia types also provide a visible internal structure, illustrated in Fig. 2. This is analogous to our treatment of the string data type: we can assign, manipulate, and work with strings as if they were self-contained objects, but we can also access individual characters within a string. Thus, in our model, images and audio data may be displayed, compared, or manipulated as a whole, while supplementary information such as image dimensions or audio frequency are also available.

The structure that we define over a multimedia type consists of three layers. This structure is similar to the models that are found in [25], [28], [5], [4]. In our case, the multimedia types are integrated into the overall ER/OO framework described in Section 3.1 by placing all multimedia types under an entity **`Multimedia Type`**. The aforementioned structural layers are translated into attributes or relationships connected to this entity. Our discussion begins at the lowest, most physical level and focuses on increasingly logical or high-level constructs.

Raw Data. The *raw data* layer is the low-level, binary representation (i.e., file, byte stream, etc.) of a multimedia type: a raw sequence of bytes without any translation or interpretation. In Fig. 2, the raw data portion of **`Multimedia Type`** is notated as the attribute **`raw data`**.

Multimedia Entities and Attributes. The next layer, called the *multimedia entities* and *attributes* layer, describes the multimedia types such that they may be accessed as actual database entities. All entities representing a multimedia type are subclasses of **`Multimedia Type`**. Images are modeled either as **`Image`** or **`Image Stack`** entities,[1] audio samples are **`Sound`** entities, digitized video are **`Video`** entities, etc.

The attributes of a multimedia entity are frequently read-only, in the same way that a text string's length is read-only. For instance, you cannot directly assign a text string's length, but you can change it indirectly by modifying the contents of the string.

Fig. 2 presents some common multimedia entities and their attributes: width, height, and depth for images, or a transcription attribute for digitized speech.

Related Schema/High Level Data. Our *related schema/high level data* layer is similar to the semantic layers found in VIMSYS [26] and SEDM [29]. Most multimedia types need to be referenced by content or in terms of other entities (i.e., "Find images with the following tumor," "Find speeches that talk about the Gulf War," etc.). This higher level, semantic information is captured in logical entities

1. The difference between an image and an image stack is primarily cardinality, in that a single image can be thought of as a one-slice image stack.

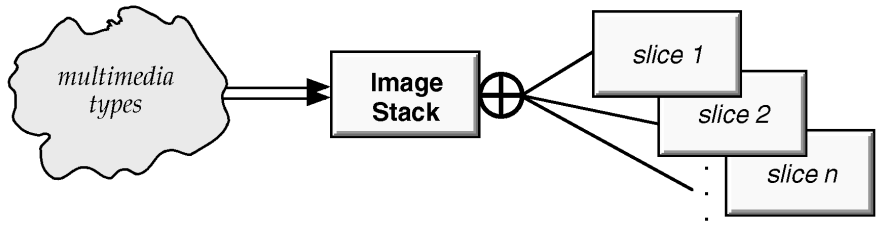


Fig. 3. The dynamic image stack model as represented in M.

that participate in specific relationships with the entity representing a multimedia type.

These entities are collectively labeled as "any entity" in Fig. 2. "Any entity" represents a set of any related database entities. These entities are identical to standard database entities, except that *they map to a particular portion of their underlying multimedia type*. For example, a tumor entity, with standard attributes such as a volume, type, and others, may also be connected to a region of an MRI image. Note that in order to be fully effective, this link is two-way—database entities point to their representative multimedia types, while multimedia types can list the database entities that are associated with them.

In Fig. 2, this bidirectional relationship is notated as the pair of one-to-many relationships *contains representations of* and *is present in*. Any entity in the database may participate in these relationships. A multimedia type itself may be viewed as a representation of another multimedia type. An example would be a thumbnail image which represents a preview version of a larger picture.

The high-level layer permits us to attach meaningful constructs to raw images and audio data, and thus permits us to query multimedia types on a level beyond just pixels or amplitudes. The use of this more logical level of access also permits integration with a knowledge base, thus providing even more contextual information for a user and serving as a guide when relaxing or modifying a query.

3.2.2 Basic Multimedia Types

Fig. 2 shows examples of the multimedia types that are common today. Fig. 2 can be expanded in one of two ways:

1) the addition of new, innovative multimedia types not currently envisioned by today's technology, and
2) the extension of a basic multimedia type (such as any of those shown in Fig. 2) to satisfy a highly specific application domain.

We now review the basic multimedia types that we see as forming the foundation of a multimedia database system.

Long Text. We have found that long text data, such as medical reports, book or article transcriptions, unstructured legacy data, etc., is very well-modeled if viewed as a multimedia type. This is due to a number of characteristics that long text shares with conventional multimedia types: high information content and/or volume, need for sophisticated search methods, and frequent interconnection with higher-level entities. Thus, long text fits very well within the three-layer structure that is described in the prior section.

Images. Images are the most ubiquitous and deeply researched multimedia types in the literature today. In our model, images are represented as entities in their own right. Structurally, they are two-dimensional arrays of picture elements or *pixels*, where each dimension is bounded by the attributes width and height. Each pixel takes up a particular number of bits, stored in the depth attribute. The `Image` entity defines methods for accessing the individual pixels of the 2D array. Other methods may also include filters or transformations, or operations for combination and convolution.

Subclasses of the `Image` entity may be defined for specific purposes or application domains. For example, a `Radiologic Image` may be defined, indicating specifically that a particular image was captured using a radiologic imaging device.

Image Stacks. An *image stack* is a logically related set of images. The model is based on previous work on pictorial database systems [10] and is ported into the data model as shown in Fig. 3. We represent an image stack as an aggregation of *slices*, each of which is given a unique name or numerical index within the stack. For example, slices of a geographical stack may represent elevation, amount of rainfall, etc., over the region covered by the stack. Alternatively, an image stack representing a magnetic resonance imaging (MRI) study may have slices numbered from 1 to 40.

Sound. Sounds are sequences of amplitude values played at a particular frequency. In addition to its actual data (a one-dimensional array of values), a Sound entity's primary attributes are its frequency and duration.[2]

Speech. Speech is a specific type of sound, restricted to digitized recordings of spoken language. Thus, speech inherits the structure and attributes of sound and adds a number of attributes specific to speech: `language`, `speaker`, and `transcription`. The `transcription` attribute to store a textual version of the speech object. The text may be derived either through manual interpretation (by a secretary, for example), or through automated speech recognition.

Video. Digital video is frequently called a *composite* multimedia type because it is made of other, simpler multimedia types. Digital video consists primarily of two *tracks*: a video track and a sound track. The video track is a sequence of images that is played back at a high rate to simulate

2. In this document, the term *frequency* refers not to the specific pitch or tone of a sound, but the rate at which a sound is digitally sampled.

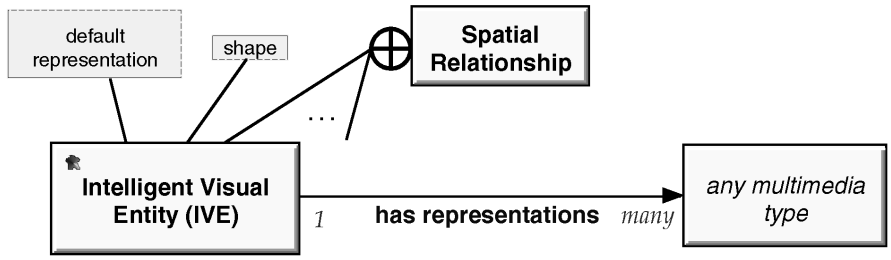


Fig. 4. Structure of an intelligent visual entity.

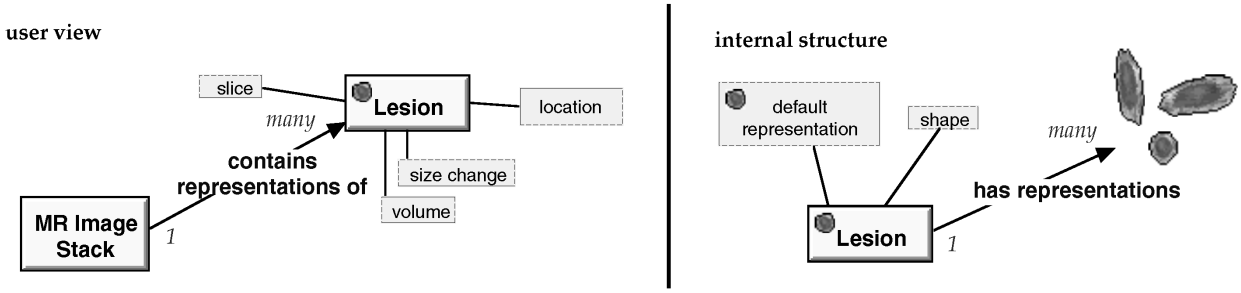


Fig. 5. Modeling a lesion entity in a database as an IVE.

motion. The sound track is sound data that is synchronized to produce audio that corresponds to the images on the video track.

3.3 Intelligent Visual Entities

In this section, we describe a construct that enables a user to ask questions about data in the most intuitive way possible—by "showing" the computer what he or she wants to retrieve.

3.3.1 IVE Definition and Semantics

An *intelligent visual entity* (IVE) is *an entity that stores visual information about itself in the database schema*. This information is then used to answer queries that are expressed as diagrams, sketches, or pictorial examples. Encapsulation is used to permit different IVEs to employ processing that is suited specifically to the application domain of each IVE.

Fig. 4 illustrates the special structure and appearance of an intelligent visual entity. An IVE has two appearances:

1) as a standard entity (with a special icon to show that it is an IVE), and
2) as a graphic "representative" of the instances of that entity.

In addition, an IVE has further properties that are typically not seen by a regular user. These properties are set by the database administrator or designer when defining the IVE.

As an example, consider a medical database that tracks various lesions within patients suffering from lung cancer. These lesions come in a wide variety of shapes and sizes, and may appear in many different locations. Thus, it is often insufficient for a radiologist to store these lesions based on artificial attributes such as an ID (i.e., "lesion 1") or oversimplified attributes such as a diameter or general location (i.e., "right lung," "left lung," etc.). Instead, a lesion can be modeled as an IVE—it is characterized not by alpha numeric descriptors but by its visual appearance and spatial location. The radiologist is thus freed from remembering these sometimes unintuitive alphanumeric attributes and can, instead, refer to lesions as "lesions that look like the one in the current image." Fig. 5 illustrates how such a lesion entity may be modeled as an IVE. The left side of Fig. 5 shows how a database user perceives the IVE, while the right side shows the internal structure that the IVE encapsulates.

Internally, an IVE is linked via the relationship *has representations* to a set of instances of multimedia types, the most common of which would be images. These images are selected from the database as sufficient visual representations of the IVE. It is from these images that an IVE can choose its graphic appearance, as opposed to the generic entity-box look. A default representation indicates the type of display to which the IVE defaults when first presented to the user. This multiple representation approach is somewhat analogous to the views in [37], but applied to multimedia (primarily image) objects as opposed to alphanumeric data.

IVEs may also be aggregated into spatial relationship entities. Placing multiple IVEs under an entity specifically defined for a spatial relationship permits the database to store specific query processing methods into that relationship, so that interpreting or comparing relationships can be tailored to the actual application domain.

Note that the IVE structure complies with the multimedia type structure defined in Section 3.2.1. The IVE functions as the high-level, highly logical object, while its spatial relationships and multimedia types (as accessed by *has representations*) comprise the IVE's multimedia entities, attributes, and raw data.

3.3.2 Usage in Queries

The IVE model was designed specifically to support visual or by-example queries on objects that can be found in images, videos, and other multimedia data. Instead of translating spatial notions such as "looks like," "near to," or "touching" into textual operators (as some other multimedia databases do), the user is permitted to express a multimedia query *as is*—drawing a desired arrangement or appearance of objects retrieves the images or other data which look like that drawing.

Within the MQuery language, visual querying is triggered when one of the images for which the IVE *has representations* is selected as the current representation. The selected image now represents the IVE onscreen, and is used to make similarity comparisons with other images at query processing time. When multiple IVEs have been set to use images as their current representation, MQuery takes into account their positions relative to each other, and interprets this position as a desired set of spatial relationships come query time.

3.4 Streams

The term "stream" is generally used in the multimedia field to represent time-based data [25]. However, because our model generalizes this concept to include more that just multimedia information, the term "sequence" may be used instead. Typically, data are *temporally atomic*—viewing or perceiving them does not take up any time. However, present applications are increasingly producing data items that occupy an interval of time. Obvious examples include digitized sound and video. This section introduces our stream data model, followed by more detailed semantics in the next section.

3.4.1 Definitions

A *stream* is *an entity representing an ordered, finite sequence of entities, or values.* These entities or values are called *elements* of the stream. The sequencing is temporal: 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.

Streams also have the notion of a *current time* from which the *current element* in the sequence can be determined. This is 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).

Substreams. A *substream* is a *stream that is itself an element of another stream*. They break up an overall sequence into semantically meaningful parts. For example, a stream that tracks changes in a tumor over time may group the first five tumor instances as a substream called "Phase A." The next seven instances may then be grouped into "Phase B," and so forth. Substreams permit our stream data model to interact with the temporal evolutionary data model previously defined by our research group [15]. In terms of digital video, a substream may be thought of as an individual *scene* out of an entire video sequence, and has been referred to as such in the literature [39]. The term "substream" was chosen over "scene" for its greater generality, as the term "scene" is too tightly connected to the area of multimedia and video.

Multistreams. A *multistream* is *an aggregation of streams combined and synchronized to form a new composite stream*. The canonical example for a multistream entity is digitized video with an associated soundtrack, which is an aggregation of synchronized video and audio streams. No limitation is placed on the type, frequency, or size of these component streams. However, the multistream that aggregates these streams must take care of managing and synchronizing their differing sizes, types, and frequencies.

3.4.2 Stream Notation

Fig. 6 illustrates the stream modeling concepts that have been defined so far. Due to the fact that sequencing or ordering is an intrinsic aspect of the data model, a new notation is introduced for representing a basic stream. Modeling the component entities as a 1-to-n relationship is not sufficient, because neither of these concepts have a sense of time or sequence.

To minimize any further notational complexity, the concepts of a substream and multistream build upon the new stream notation using known constructs. To express a substream, the elements of a stream are given unique names, and streams with these names are defined elsewhere in the diagram. Multistreams are expressed by aggregating separate streams into an entity. Multistreams may contain streams with varying frequencies or element durations, although this variation is not necessarily seen in the formal schema notation.

Streams, substreams, and multistreams are all special kinds of entities that, like standard entities, have attributes, have relationships with other entities, or participate in an inheritance ("is-a") hierarchy. In addition, the elements of a stream, when they are not substreams, are thought of as entities also. This permits the elements of a stream to have attributes, relationships, etc., that are separate from the attributes and relationships of the stream itself.

3.4.3 Practical Applications for Streams

In this section, we discuss the specific application of streams to a variety of domains. Previous work on this type of stream data structure has been, in general, applied only to multimedia data such as digitized video or music [25], [4], [39], [28]. It has not been framed within the greater context of a generalized database management system. The key idea that differentiates our stream work from the stream work of others in the field is our *application of the same general stream model to a diverse range of domains* that previously have been developed either in isolation of other subject areas or have not yet been sufficiently modeled in a database management system.

One of the most significant areas to which we are applying streams is to data resulting from *simulations.* The representation of simulation data into our stream structure has great potential for scientific database applications, which need to correlate *multiple* sets of simulated and real-world data in many domains.

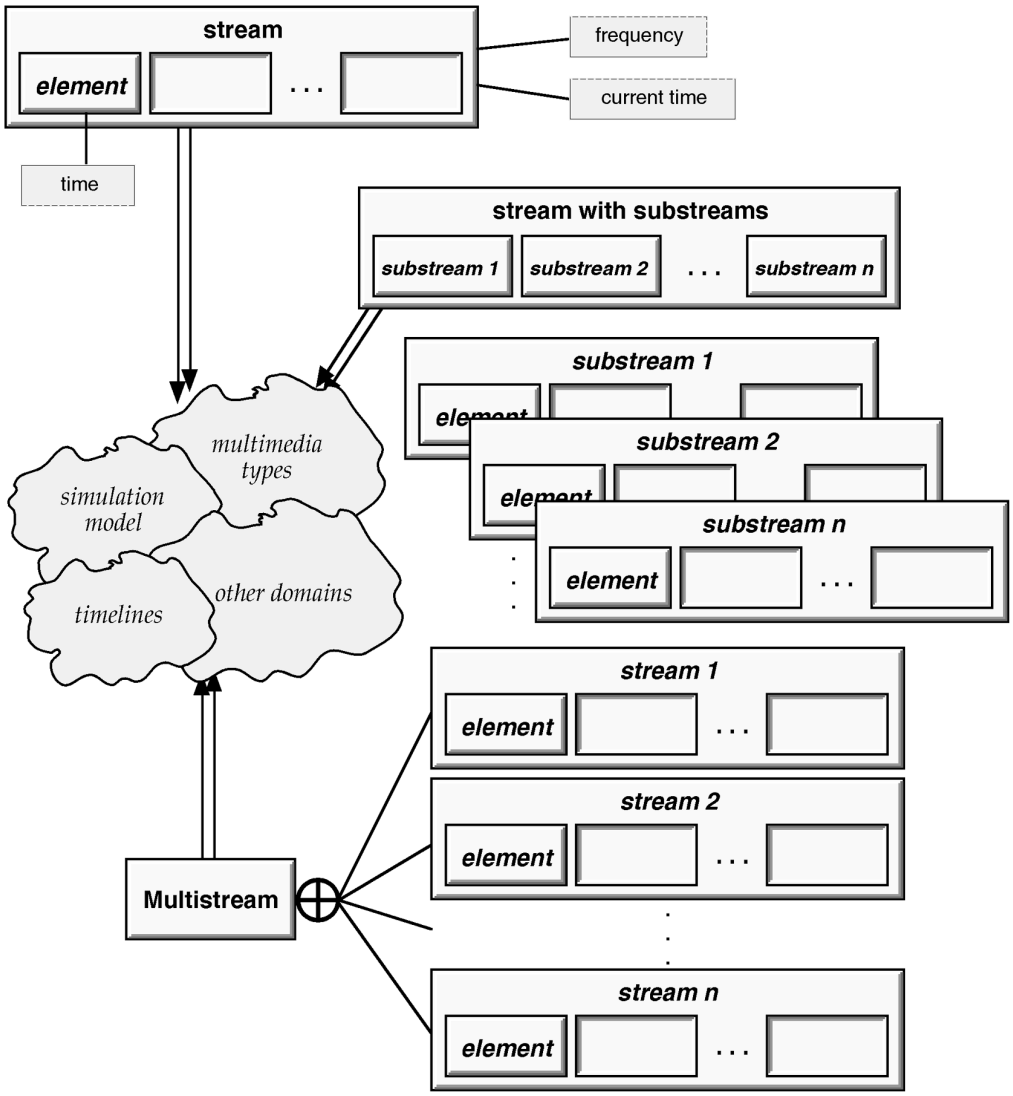


Fig. 6. Graphical diagram of abstract stream entity types and their internal structures.

Multimedia Types. Streams are highly applicable to multimedia data models. This is due primarily to the intrinsic temporal element behind such multimedia types as sound and video.

Sound, for example, has an intrinsic temporal quality. Sounds cannot be perceived instantaneously—listening to them *always* takes some finite amount of time. Thus, they are well-modeled as streams of samples at a given frequency. This approach permits sound data to be accessed and presented using the same notation and semantics as other higher-level streams (timeline histories, simulations, etc.), and is in fact an illustration of one of the significant contributions of this work.

Speech, since it is a subclass of sound, is also modeled as a stream. In addition, its transcription attribute may also be modeled as a stream, where individual words or even phonemes are sequenced over time. This makes possible queries such as "Retrieve dictations where the speaker mentions the word 'cancer' within five minutes of the recording." Although this particular avenue of research is not explored in depth by our work, it certainly holds interesting possibilities, and further illustrates the usefulness and versatility of our stream model.

Digital video, like sound and speech, is intrinsically time-based. An additional level of sophistication is necessary, however, due to the existence of multiple tracks (i.e., audio and video) that need to be synchronized during presentation. Thus, digital video is modeled as a multistream.

Simulations. Simulation data are easily represented using multistreams, managing the many parameters, mathematical models, geometries, and output sets of a simulated system. They also support relationships with real-world measurements and datasets that will aid the user in validating the accuracy of a simulation.

Simulations generally aggregate two kinds of streams:

1) a main data stream manages the actual, numeric output of a simulation, while
2) multiple presentation streams take care of any visualization or presentation of that data.

Thus, the elements of data stream are conventional entities with alphanumeric attributes, while the elements of the

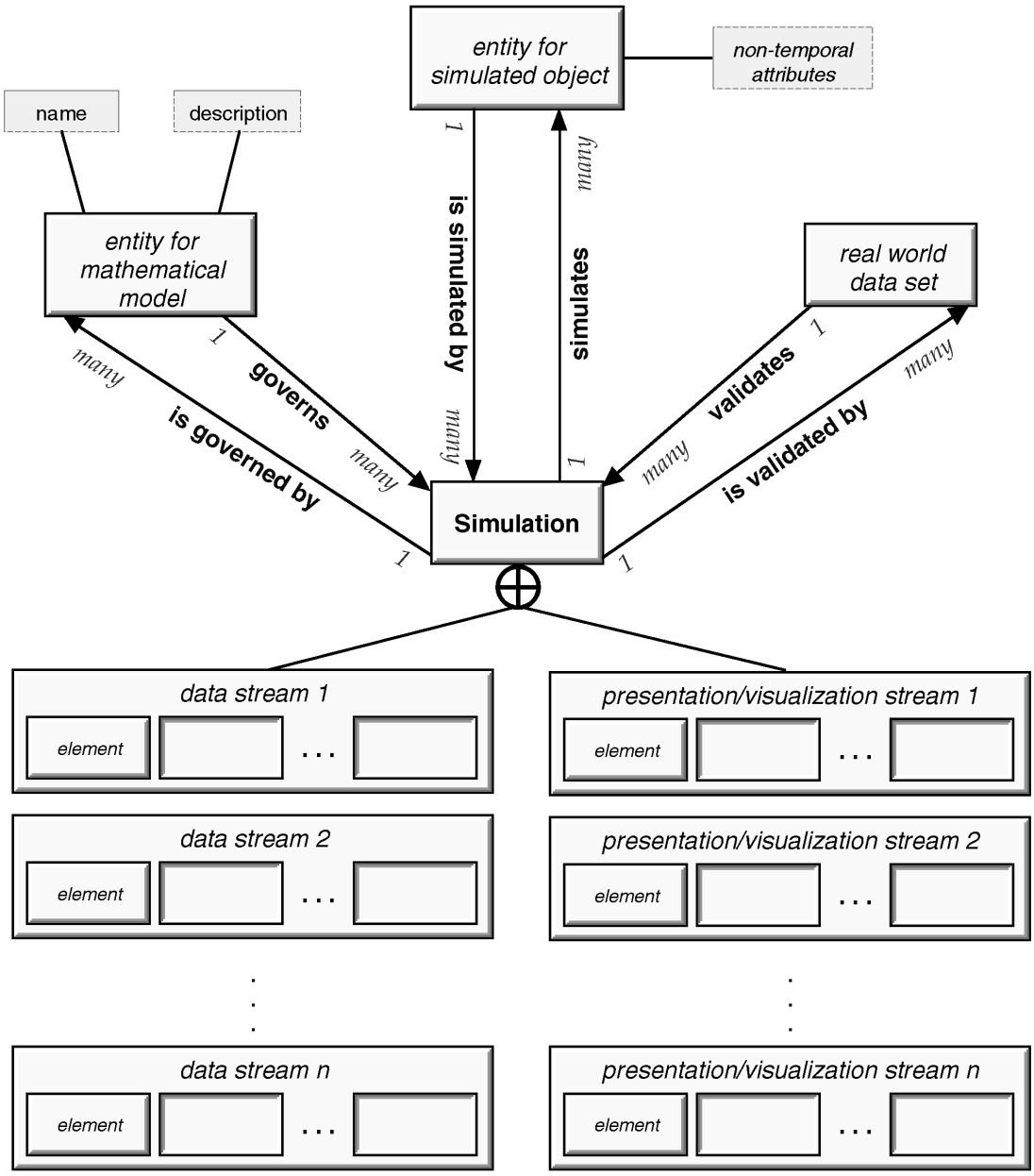


Fig. 7. Graphical data model for a simulation entity.

presentation streams may include images, three-dimensional meshes, graphs, or other presentation-oriented objects.

In our model, simulations participate in bidirectional relationships with two other kinds of entities: logical-level entities of the objects being simulated, and real-world measurements or datasets for those simulated objects. Fig. 7 illustrates the many relationships involved among these three entity types.

For example, a doctor may wish to simulate the behavior of a cerebral aneurysm. This system involves three types of database constructs: an entity representing the aneurysm and its attributes (such as patient ID, aneurysm type, etc.), a simulation entity that captures the mathematical model and other data relevant to the aneurysm's simulation, and a set of angiograms that represent actual, radiographic footage of the aneurysm in question. The aneurysm simulation *simulates* the aneurysm entity, the aneurysm entity *is present in* the set of angiograms, while the angiograms themselves *validate* the accuracy of the aneurysm simulations. The database's knowledge of the semantics of these complex interrelationships will permit the doctor to manage all of this information in a unified and integrated fashion.

A new and significant aspect of using streams to model simulation data is the notion of *countability*. Although actual, stored simulation data are always *discrete* sets of values, the data modeler may choose to present them as if the data points existed along a *continuous* flow of time. The notion of being discrete or continuous does not directly affect the data model, or even the database, since digitally stored information is *always* discrete. However, if a continuous view of data is required, the necessary methods must be written to provide, perhaps dynamically, any data points

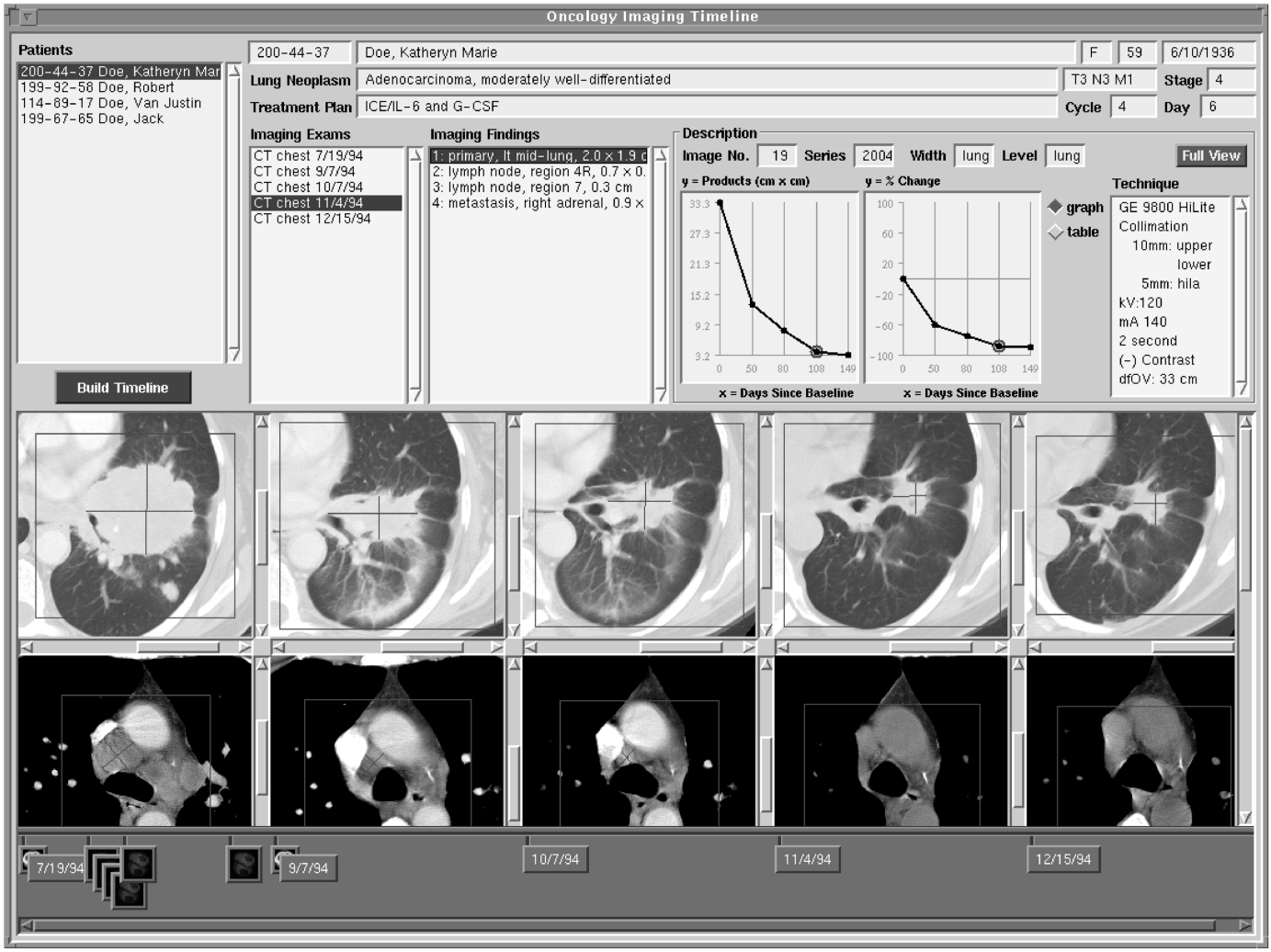


Fig. 8. A sample timeline that merges radiological images, reports, and graphs into a single overall view of a given patient.

that fall between the discrete time slices for which stored data points exist.

Timelines. Another effective application of streams and multistreams can be found in the representation and modeling of *timelines*. In radiology, our group has been exploring the use of timelines to concisely present the overall history of a patient, including images, reports, and graphs, in a single view. Fig. 8 illustrates an example of such a timeline.

Shown is an imaging timeline for thoracic oncology. On the upper-left corner of the window is a list of patients retrieved by a database query. A detailed view of the currently selected patient, including graphs for selected attributes, is shown to the right of this list. Below this alphanumeric data is the timeline multistream itself, divided into two panes. The upper pane is a timeline of images, showing one stream per lesion identified by the radiologist. The lower pane is a stream of other nonimage events in the patient's history. Different icons represent different kinds of events or data, such as a pathology test, radiological exam, etc.

Our stream-as-timeline model is applicable to diverse fields such as thoracic oncology and thermal ablation therapy of brain tumors.[3] We have developed a prototype system for proof-of-concept, reported in [22]. Like an individual simulation data set, medical timelines are modeled as multistreams. Aggregated streams include various image streams, report streams, graph streams, and any number of other data types. Other potential aggregated streams include annotations, spoken recordings made by doctors, etc.

Doctors and radiologists have found timelines to be highly useful at presenting as much information about a patient as possible with a high degree of organization and clarity (as reported in [2]). Previous reporting methods either lacked the "big picture" view enabled by a timeline or lacked the amount of detail that most doctors require of their data.

Temporal Evolutionary Data Model. Temporal and evolutionary constructs, as defined in [15], may also be reformulated using a combination of intelligent visual entities and streams. Fig. 9 summarizes the new notation for the temporal evolutionary data model (TEDM).

3. Some readers may view such data, which are sampled at distant points in time, as versions of the medical image.

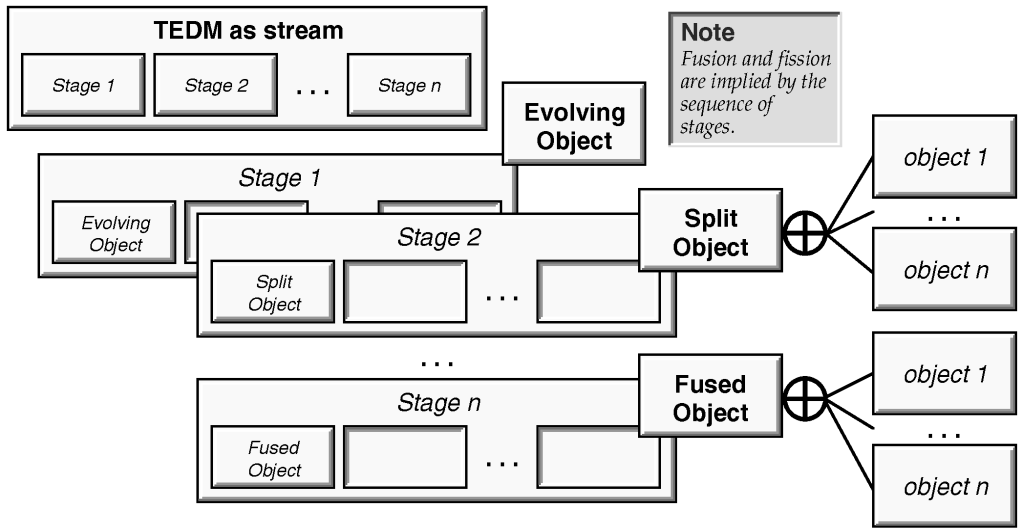


Fig. 9. Notational conventions for TEDM constructs.

The overall evolution of an object in TEDM is now captured in M as a stream or multistream. Stages within the evolution are translated into named substreams. The elements of these substreams represent individual instances of the evolving objects. When the elements of an evolutionary stage consist of multiple entities, the elements of that stage are modeled as aggregations of those entities (as can be seen from the *Split Object* and *Fused Object* entities in Fig. 9). The occurrence of fission and fusion are implied by the ordering of the stages—for example, if the elements of one stage are not aggregated entities and the next stage consists of aggregations, then the transition from one stage to another is fission.

3.5 Semantics of Streams

In this section, we formalize our stream data model and the concepts behind stream time. Our perspective of time views it from *within* an entity, and thus differs from other temporal models and algebras [15], [36] which study time's characteristics *outside of* the objects in the database. These temporal approaches focus on the history, evolution, or versioning of an object as time passes. With our stream model, time does not span over multiple separate objects. Instead, time passes *inside* an individual stream. Although a stream does consist of multiple elements, these objects are encapsulated within a single, larger construct, and can be manipulated collectively as a single entity instead of a linked set of entities. These two approaches to time are not mutually exclusive. They only look at time from different points of view.

Our model of time is based primarily on the models used in timed streams [25] and QuickTime [4]. It has been adapted so that it can be used for new domains such as simulation, medical imaging, and scientific computing.

3.5.1 Formal Definitions

A stream is either *discrete* or *continuous*, depending on the quantifiability of its elements.

DEFINITION 1. *A* discrete stream *S is a 6-tuple* (f, d, t, $\triangle t$, n, E) *where*:

- *f is the* frequency *of the stream.*
- *d is the* duration *of the stream.*
- *t is the* current time *of the stream.*
- *$\triangle t$ is the stream's* time vector.
- *n is the number of elements in S.*
- *E is a sequence e_0 to e_{n-1} of elements e_i.*

In general, f is expressed in hertz (Hz) and its multiples. d, t, and $\triangle t$ may be expressed in either ticks or seconds. n is a finite natural number, and e_i is as defined below.

DEFINITION 2. *A* continuous stream S_∞ *is a 6-tuple* (f, d, t, $\triangle t$, a, E) *where*:

- *f, d, t, and $\triangle t$ are the same as in a discrete stream.*
- *a is a domain over which the elements of the stream may be "indexed."*
- *E is a continuum $e(0)$ to $e(a)$ of elements where $e(x)$ is a function in* [0, a] *whose result is an element as defined below.*

DEFINITION 3. *An element e_i or $e(x)$ is a 4-tuple* (O, T, R, V) *where*:

- *O is the* index *of that element in the stream. In this definition, $O = i$ for discrete streams or $O = x$ for continuous streams.*
- *T is the* tick *of that element in the stream.*
- *R is the* time *of that element in the stream.*
- *V is the value, or data, stored by that element.*

V can be anything from an image or sound sample to a medical report from a timeline. V can also be another stream, in which case it is a substream of its owning stream S.

DEFINIITON 4. *A multistream M is a 7-tuple (f_M, d_M, t_M, $\triangle t_M$, n_M, σ, s) where*:

- *The variables f_M to $\triangle t_M$ correspond to the variables for a stream S, except that they are now applied to the multistream.*

- *n_M is the number of streams in M.*
- *σ is a set S_0 to S_{n_M-1} of streams S_k.*
- *s is an array of* start times *s_k for each stream S_k in σ.*

For multistreams, the expressions f_k, d_k, etc., are used to denote the frequency, duration, or other attribute of the kth stream in the multistream.

The idea of continuous streams does not strictly contradict the earlier, more intuitive definition of streams as finite sequences of elements. Streams, after all, are ultimately stored as digital data, which is necessarily finite as stored. However, this finite data, through interpolation or dynamic calculations, can be used to provide the *appearance* of a continuous stream.

3.5.2 Time in Streams

The temporal basis of a stream entity is its *time scale*. A stream's time scale determines how the stream measures time when compared to our own sense of "real time." The succeeding sections define the concepts that are associated with a stream entity's time scale.

Real time is time as perceived and measured by human beings. In our model, real time is measured in *seconds*. All other temporal units are based on either fractions of or multiples of a real-time second. Real time is a *continuous* time scale, because every point in real time can be defined and isolated with infinite precision.

Streams measure time in terms of *ticks*. The passage of ticks, or *stream time*, represents a *discrete* time scale. Within an individual stream, time cannot be divided into units that are finer than the duration of an individual tick.

Frequency. Stream ticks are mapped into real-time seconds by their *frequency*. A stream's frequency is the *number of ticks per second*. The variable f is used to denote a stream's frequency, which is often measured in hertz (Hz) and its multiples. *Tick length* is the number of seconds between ticks, and is determined by calculating $1/f$. The amount of real time occupied by t ticks is thus t/f.

Duration. A stream occupies a finite span of time. Thus, it has a total *duration*. A stream's duration is the amount of time occupied by a stream, and it can be measured either continuously (in seconds) or discretely (in ticks). As a discrete measure, the duration d is the total number of ticks spanned by the stream. To convert this value into real time, d is divided by the stream's frequency f to arrive at the number of seconds spanned by a stream (d/f).

Current Time. There are instances when a user or system must access a stream during a specific point in time. These instances may occur during the "playback" of a stream, or when querying its data for a specific event or during a particular span of time. During these instances, the time value on which a stream is "focused" is defined as its *current time* or t. During playback, for example, a stream's current time is set to zero, then incremented at some predefined rate until it reaches the stream's duration.

The current time can be expressed either in a stream's ticks or in real-time seconds. Stream methods are available for converting between the two time scales.

Time Vectors. Another concept in time modeling is the notion of a *time vector* or $\triangle t$. Time vectors represent the state of time's movement within a stream: they express the *direction* (forward or backward) and *rate* (in ticks per second) of time's flow. Time vectors +1 and –1 represent playback and reverse playback at the stream's standard frequency. The special time vector 0 represents a state of "frozen time" within the stream entity.

Start Times. For multistreams, an additional time-related value called the *start time* is defined. One start time s_k is defined for each stream in the multistream. The start time denotes the time in a multistream M for which a given stream S_k starts to "play." Thus, if an element in a stream occurs τ seconds from the beginning of that stream, it is then invoked at $s_k + \tau$ seconds *in relation to the entire multistream*.

3.5.3 Precise Stream Element Access

Three options exist for directly accessing or locating an entity within a stream: *access by order*, *access by tick*, and *access by real time*. As we describe these forms of access, let S represent a given stream. For a discrete stream, n is the number of individual elements in S. Accessing methods to continuous streams are analogous to the discrete versions presented here.

Access By Order. The elements of a stream can be accessed in terms of their order within the stream. This order is defined purely by which entity comes before or after another. The length of time that passes *between* entities is not considered.

Access by *order* is notated as $S_O(i)$ for the ith element within the stream. Clearly, i is strictly an integer, and is defined as ranging from 0 to $n - 1$. In a database schema, stream elements have an index attribute that indicates the element's position in the overall stream.

Access By Tick. Elements within S can be accessed in terms of the *tick* at which an element occurs within S. An element that occurs at tick j is written as $S_T(j)$, where j is an integer from 0 to $d - 1$. Because not every tick in S is required to have a corresponding element, it is possible that $S_T(j)$ may be null. For database access, each stream element has a tick attribute that returns the value of j for that element.

Access By Real Time. An element of S can also be accessed in terms of the *seconds elapsed* between the real-time beginning of the stream and the occurrence of the element. An element that is accessed in this way is written as $S_R(\tau)$, where τ is a real number ranging from 0 to $(d-1)/f$.

The tick j represented by time τ is $j = \tau f$. As with access by tick, it is entirely possible that no element occurs precisely at $S_R(\tau)$. In that case, this expression has a value of null. A database or query language can access the time attribute to find out a particular element's timestamp.

3.5.4 Imprecise Stream Element Access

Not every tick (and certainly not every second or fraction thereof) will have a corresponding element. For example, a stream S may have an element occurring at the sixth tick, with the next element not occurring until the ninth tick. Thus, to be precise, no element exists at $S_T(7)$. However, it is

often acceptable to think of an element as "applicable" or "active" until the next element in the stream comes along. One such instance is digital video, where a particular frame remains on display until the next frame comes along. Thus, an element *may* be defined to exist at $S_T(7)$.

In order to differentiate between precise (with nulls) and imprecise (without nulls) stream access, braces ({}) are used to designate imprecise access: for $\{S_T(j)\}$ or $\{S_R(\tau)\}$, if no element exists precisely at $S_T(j)$ or $S_R(\tau)$, then the element returned is the *last* element that occurred within the stream. Thus, in the digital video example, $\{S_T(7)\} = \{S_T(8)\} = S_T(6)$.

A domain for which imprecise stream access may not be acceptable is simulation. For example, if a particular simulation run only solved its equations every 10th of a second and the state of the system at 0.15 seconds is desired, returning the values at time = 0.10 seconds may not be acceptable. In this case, *interpolation* is the more desirable approach to returning the state of the stream at 0.15 seconds.

3.5.5 Stream Quantifiability and Interpolation

The multimedia application of streams can reasonably assume that streams are *discrete*—in other words, a stream's elements are countable, in the mathematical sense. However, when the notion of streams is expanded to cover other time-based data, this assumption becomes invalid. For example, in the case of simulation data, data points theoretically exist at every moment over the course of a simulation. Although it is true that these data points are never all calculated, from a modeling point of view it is incorrect to view the simulation stream as a discrete sequence of elements. We thus permit the notion of a *continuous* stream: streams which can be accessed only by real time or by tick, and not by order.

The entire data set of a continuous stream cannot be stored, so individual elements must be interpolated. The method of interpolation is very dependent on the application domain.

3.5.6 Composition of Multiple Streams

A time scale is only applicable to the specific stream entity for which it has been defined. Certain types of stream entities will share identical time scales: CD audio, for example, or standard NTSC video. However, most streams will have different time scales, particularly when dealing with simulation data, their visualization, and real measurements on the simulated object(s). It is thus necessary to look closely at what is needed in order to properly compose such streams into an integrated multistream entity.

Translating Times Among Streams. The time stream translation problem can be stated in this way:

> *Given streams S and S′ with differing time scales, where the current time in S is given as j_S ticks, find $j_{S'}$ in terms of S′ ticks such that in real time, $\tau_S = \tau_{S'}$. This translation is a fundamental step in performing more advanced operations such as synchronization and interpolation.*

The translation algorithm is simple and universal. To determine the amount of real time (in seconds) represented by a given number of ticks in a given frequency, we have the expression $\tau = j/f$. Therefore, the relationship between j_S and $j_{S'}$ is:

$$\frac{j_S}{f_S} = \frac{j_{S'}}{f_{S'}} \tag{1}$$

Note that when f_S does not divide $j_S f_{S'}$ exactly, this equation will result in a noninteger value for $j_{S'}$. In this case, a ceiling or floor function (i.e., $\lceil j_{S'} \rceil$ and $\lfloor j_{S'} \rfloor$, respectively) may be necessary, depending on the purpose of the time stream translation. Alternatively, the system may *interpolate* the state of stream S' during that noninteger tick.

Synchronization of Time Among Streams. The need for synchronization arises when multiple streams with differing time scales are combined into a single multistream entity. The classic example of such synchronization is digitized video and sound. A video stream with a frequency of around 30 frames per second must be presented alongside an audio stream which, at CD-quality levels, can reach frequencies of up to 44,000 Hz.[4] The problem of synchronization lies in determining what corresponding components of each stream are simultaneously "active" at any given time. Inversely, if two events in two separate streams are required to occur simultaneously when the streams are aggregated into a multistream, the problem is to determine "when" in the overall multistream each local stream should start.

Conversion between the time scales of a stream and its aggregating multistream is straightforward: given S_k with frequency f_k and start time s_k, a tick j_{S_k} is easily translated into a tick j_M of the aggregating multistream M in this manner:

$$j_M = s_k + j_{S_k} \frac{LCM(f_0 \ \ldots \ f_{n_M - 1})}{f_k} \tag{2}$$

Streams and multistreams defer to the approaches used in QuickTime [4] and the Amsterdam Hypermedia Model [28] when handling more complicated synchronization issues.

3.5.7 Interpolation Within Streams

In addition to synchronization, *interpolation* is another requirement that may arise from the aggregation of multiple streams with differing time scales into a single multistream. Interpolation is the *calculation of a new element based on the elements already present within a stream*. Specifically, if the state of a stream is required at a particular time or tick j and no element occurs (or is valid) precisely at that moment, interpolation can automatically generate a new element for j based on the element that precedes or succeeds this instant of time.

Our data model's approach to interpolation is simple: The ability to interpolate components of a stream may or not be present in that particular stream. If a stream entity is capable of interpolation, then it is an *interpolation-capable* stream, and this capability may either be enabled or disabled. Note that a continuous stream

4. Recall here that "frequency" in this document refers not to a sound's pitch but to the rate at which its waveform has been digitally sampled.

should always be interpolation-capable. Streams whose curves are not differentiable in the mathematical sense may define their own interpolation algorithms for dealing with any discontinuities.

4 Sample Application and Prototype

In this section, we describe a sample application that takes advantage of the new constructs presented in this paper, showing how their use greatly improves the usability and clarity of a database schema. The application is taken from the medical domain, and is first defined by its subject matter and requirements. The prototype that we developed using M is then presented.

4.1 Domain and Requirements

The example domain presented here is based on a multimedia database for thermal ablation therapy that has been developed by our group [22], [21]. However, we are also exploring other domains, such as thoracic oncology, though results are reported elsewhere [2].

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 1960s, and have been further refined since then [18], [1], [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 patient records, models, and images that are relevant to the thermal ablation therapy application domain. Using the modeling constructs provided by M, an instantiation of the schema can store patient records and images, associate them with each other, and perform queries on this data based on features such as tumor volume or histology. In addition, the schema supports simulations of the heat transfer process that occurs during thermal ablation therapy, mapping these simulations to the appropriate patients where applicable.

4.2 Discussion of Schema

Fig. 10 shows the overall thermal ablation therapy schema that we have developed as a testbed for the M data model. The schema shown is actually a subset of a larger project between our Computer Science and Radiology Departments. A broader data model spanning many other areas of medical imaging is being developed as a part of that larger project.

4.2.1 Patients and Health Care Activities

The standard representation of a patient is shown in Fig. 10. This representation stores an individual **Patient** as an entity participating in the 1-to-n relationship *undergoes* with a **Health Care Activity** as its destination. **Patients** have attributes such as a patient ID, name, and many others.

The sample database keeps track of two types of health care activities: **MR Examinations** and **Thermal Ablation** treatments. **MR Examinations** *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 States** at the time of the examination. This relationship shows an application of the multimedia type model illustrated in Fig. 2. In addition, **Brain States** and **Lesion States** are modeled as intelligent visual entities (IVEs), because they directly correspond to some visible region of interest in the **MR Image Stack**.

Thermal Ablations represent instances of actual thermal ablation procedures performed on the patient. They include information on the actual ablation procedure, such as 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

Our stream model is called upon frequently in Fig. 10. In one instance, as examinations accumulate over time, individual **Brain** and **Lesion States** (essentially snapshots at a particular moment in time) are collected into streams that fully represent the **Patient's Brain** and particular **Lesions** 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 Fig. 10, but may be explicitly modeled as necessitated by the application domain). 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 *generates* one or more **Lesions**.

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** *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 Simulations** follow our simulation model (as seen in Fig. 7), capturing the heat transfer equations that theoretically characterize 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 Overview

To accompany the data model presented in this paper, we propose a highly visual query language called MQuery that directly supports the new and unique concepts in our data model. MQuery is a next-generation evolution of the language PICQUERY+ [14]. We provide herein only an

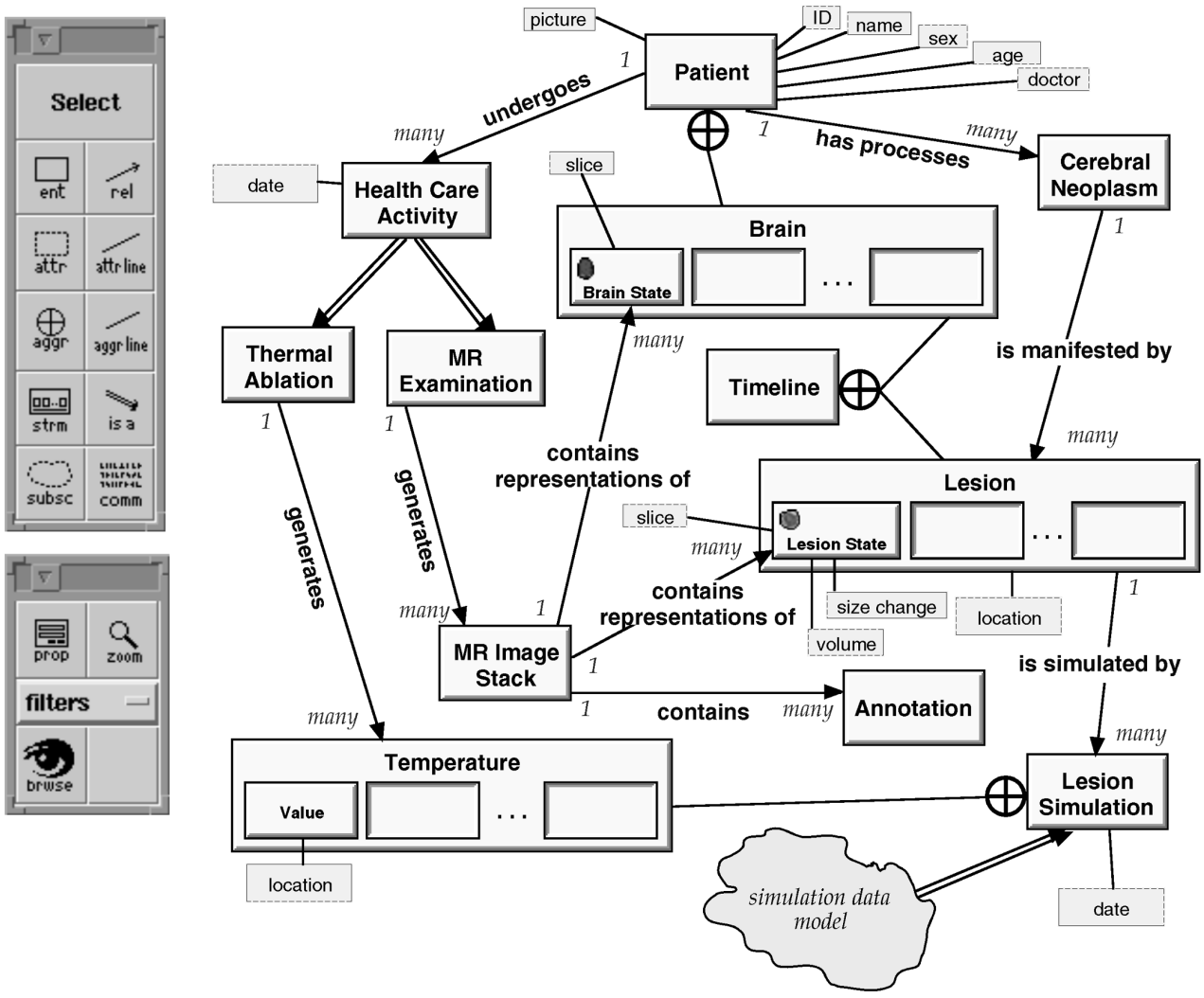


Fig. 10. Sample schema using the M data model for thermal ablation therapy data management.

overview of MQuery; a more detailed discussion of the language can be found in [19].

Queries with complex predicates (WHERE clause in an SQL statement) are more conveniently expressed in a tabular form. Thus, MQuery retains the capability to use PICQUERY+'s tabular format at the same time as its new visual constructs.

5.1 Multimedia Data Access: MQuery and Modules

With MQuery, we expand the idea of a query language beyond the conventional notion of a pure data retrieval system. Instead, the data retrieval functionality is only a module of the overall design. Other modules include:

- Schema designer, browser, and editor (called MSchema).
- Visual interface for inserting, modifying, retrieving, and deleting data.
- Information visualization and presentation.

Thus, MQuery serves as the front end for virtually all database operations. By interacting with the user during all phases of database design and manipulation, a high degree of integration and consistency across all of these stages is achieved. This level of integration permits features such as "feedback" queries—where any query result or set of results may be used as input for new queries—and intensional queries, which permit the user to ask questions about the schema itself, as opposed to merely its data.

5.2 Query Formulation

The user forms a query in MQuery through a series of copy-paste actions. With a schema window on one side and a query window on another side of a display, the user "copies" desired schema elements and "pastes" them onto the query window. These actions result in the formation of what resembles a subschema in the query window.

When schema elements are pasted onto a query window, they can be filled out by the user to satisfy the conditions of the query. In addition, the desired query results are indicated. This is visually represented by an extra-thick border around the schema elements that have been designated as the desired query result. Fig. 11 and Fig. 12 show examples of visual queries formulated in this manner.

5.3 Sample Queries

The sample queries below have been separated into different categories in order to highlight selected features of MQuery.

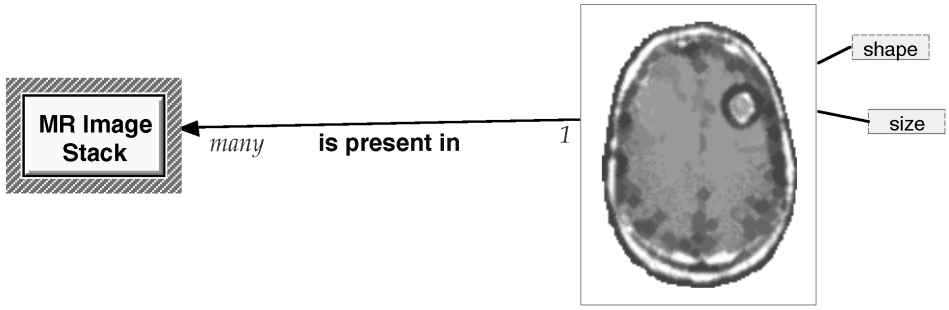


Fig. 11. MQuery expression for the query "Retrieve radiologic images which contain objects similar to the ones that I will place onscreen."

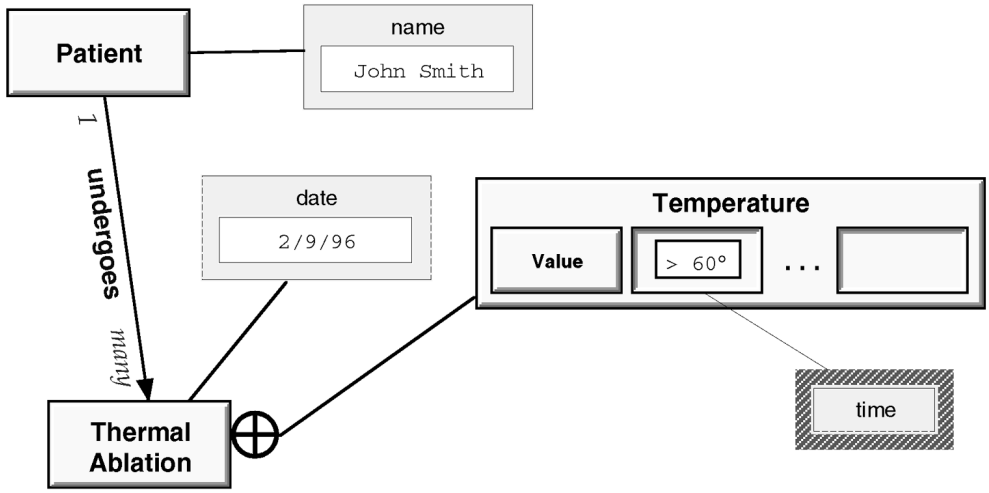


Fig. 12. MQuery expression for the query "When does the tissue in the lesion being treated for John Smith on February 9, 1996, become greater than 60°C?"

Note that these separate query categories can be arbitrarily mixed and matched to form queries that use more than one special MQuery feature at a time.

Alphanumeric Queries. Alphanumeric queries are well-handled by traditional alphanumeric commercial DBMS. MQuery contributes to this query category by providing a visual interface that simplifies operations including the equivalents of joins and subqueries.

Query 1. *Display a list of all the patients who are currently in the database.*

Query 2. *Display all reports generated concerning patient John Smith that are dated September 2, 1995, or later.*

Both of these queries can be expressed and answered by the current prototype.

Queries with Multimedia Results. These queries retrieve data that are not alphanumeric in nature. They highlight the multimedia types of the data model, and how they are closely coupled to the query language. Fig. 11 provides the MQuery diagram for Query 3.

Query 3. *Retrieve radiologic images which contain objects similar to the ones that I will place onscreen.*

Query 4. *Play back the voice recordings for images where Dr. Chan recommends chemotherapy.*

Query 5. *What are the radiologic/imaging appearances of a particular pathology?*

Query 3 can be expressed and answered by the current prototype. Query 4 and Query 5 can be expressed in the language given the right data model, but cannot yet be answered by the current prototype.

Queries with Multimedia Predicates. Multimedia data can be used in MQuery not only as query results but also as participants in the actual predicates. The queries below show predicates that cannot be answered solely from alphanumeric information.

Query 6. *Obtain the sex, age, and doctor of all patients with tumors similar in shape to the tumor currently being viewed.*

Query 7. *Locate other treated lesions in the database similar with respect to size, shape, intensity, and growth or shrink rate of the current case.*

Query 8. *Does the lesion overlap any of the activated areas from the functional MRI study?*

Query 6 and Query 7 can be expressed and answered by the current prototype, although better techniques for answering Query 7 are being investigated. Query 8 can be expressed but not answered by the current prototype.

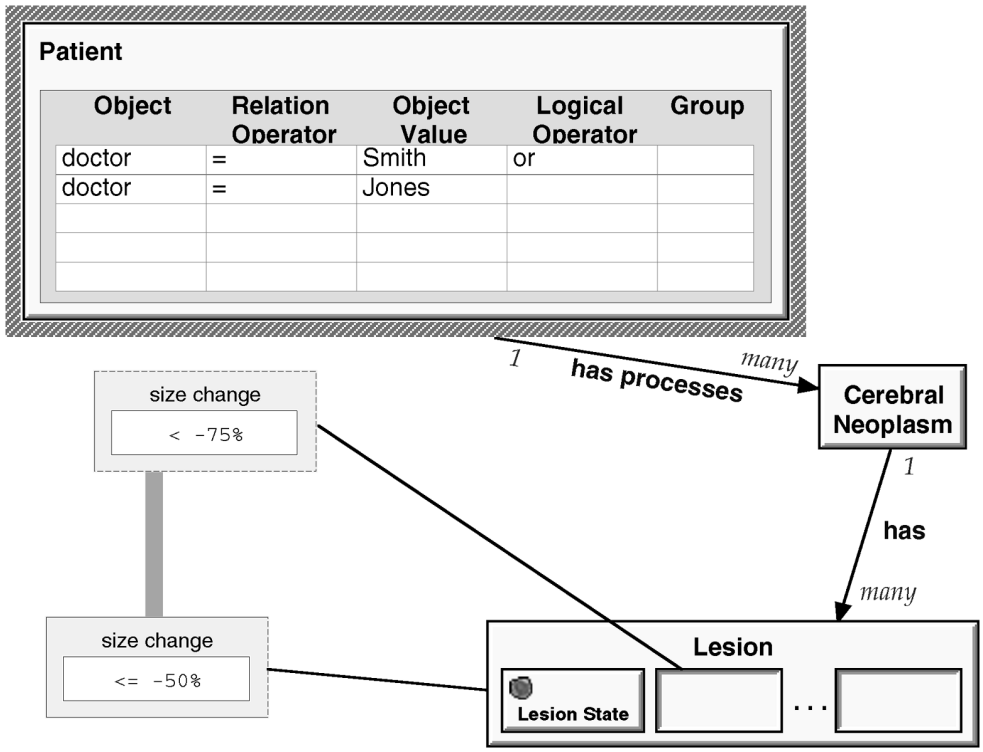


Fig. 13. MQuery expression for the query "Find patients who are currently on treatment protocols X or Y whose primary lesions exhibit a decrease in size by at least 50 percent for every examination since baseline, or have at least one examination that exhibits a decrease in size by greater than 75 percent."

Queries Over Time-Based Data. This category of queries highlights how our stream model makes querying over time-based data simpler and more intuitive. Fig. 12 provides the MQuery diagram for Query 10.

Query 9. *Retrieve and play back the thermal ablation simulation results for patient John Smith for any simulation runs performed after January 10, 1996.*

Query 10. *When does the tissue in the lesion being treated for John Smith on February 9, 1996, become greater than* 60°*C?*

Query 11. *How does the shape and volume of the left frontal lobe tumor of Jane Doe change with respect to time post therapy?*

Query 12. *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.*

All of these queries can be expressed in the current prototype, and all are theoretically answerable. However, lack of data, particularly in the area of simulations, has prevented the full implementation of this answering capability.

Feedback Queries. These queries show how MQuery's integrated modules make it simple to pass the results of one query into another.

Query 13. *What are the volumes of the tumors that were retrieved in the previous query?*

Query 14. *Where and when does maximum tissue heating take place for the simulation run that is currently on display?*

Query 15. *List other cases that have a meningioma of similar size to the case currently being viewed.*

Query 16. *What are the most up-to-date imaging and therapeutic methods for the pathologies currently displayed on the screen?*

These queries have been specified in the language but are not yet implemented in the current prototype. Specifically, linkages among queries and their results have not yet been implemented.

Queries With Multiple Predicates. A key challenge of a general query language is to permit the user to express complex Boolean predicates without detracting from the languages intuitiveness or usability. The following queries are intended to test the usability of MQuery's approach in expressing Boolean predicates. Fig. 13 provides the MQuery diagram for Query 17.

Query 17. *Find patients who are currently on treatment protocols X or Y whose primary lesions exhibit a decrease in size by at least 50 percent for every examination since baseline, or have at least one examination that exhibits a decrease in size by greater than 75 percent.*

Query 18. *Find cases which demonstrate a tumor in the posterior fossa adjacent to the dura or next to the fourth ventricle.*

Queries 17 and 18 can currently be expressed in the older PICQUERY+ language [14]. A PICQUERY+ compatibility module has been partially implemented in the current prototype. This module can express these queries within the newer MQuery environment, but cannot yet answer them.

5.4 Stream Query Processing

Stream predicates define logical conditions over streams. There are two types of stream predicates:

1) *Stream entity predicates* apply to an overall stream, such as "streams with more than 100 elements," or "streams over tumors of type X." These predicates are no different than standard predicates, in that they treat streams as straightforward, self-contained entities. For example, the predicate "performed after January 10, 1996," in Query 9 is a stream entity predicate.
2) *Stream element predicates* are stated in terms of a stream's elements. These predicates are repeatedly applied to the elements of a stream, returning either a **true** or **false** for each element. The predicate's result is a new stream consisting only of those elements that returned a **true** value for that predicate. For example, "greater than 60°C" in Query 10 is a stream element predicate.

As stream entity predicates are semantically no different from standard predicates applied to conventional entities, the discussion in this paper focuses on stream element predicates. In addition, the following definitions and results lead to indices over stored streams in a database, and thus focus on discrete streams or continuous streams with digitally stored representations (and are therefore discrete streams at that level).

5.4.1 Streams of Element Index Ranges

Queries 10, 11, 12, and 14 contain examples of stream element predicates. The formal definition of a stream element predicate is given below.

DEFINITION 5. *A* stream element predicate P_S *is a Boolean expression that can be applied to the elements* $S_O(i)$ *accessed by order from a stream* S. $P_S(S_O(i))$ *is the value of that predicate for a particular stream element* $S_O(i)$, *which is either* **true** *or* **false**.

DEFINITION 6. *Given a stream* $S = (f, d, t, \triangle t, n, E)$ *and a stream element predicate* P_S, *the* stream predicate result $P_S\{S\}$ *is a stream* $(f_P, d_P, t_P, \triangle t_P, n_P, E_P)$ *where* $E_P = \{e \mid (e \in E) \wedge P_S(e)\}$.

The query "Retrieve lesions whose tissue temperature is greater than 60°C," which is a slight variation of Query 10, produces a stream predicate result consisting of lesions satisfying the stream element predicate "greater than 60°C." A stream of element indices can be derived from every stream predicate result (as in the above example). Further, consecutive indices can be concatenated into element index ranges, particularly when a stream element predicate holds true over an extensive subsequence of stream elements. This "stream of ranges" thus constitutes a storable index over a set of streams. Additional conventional indexing may then be performed over these sets of ranges, which are, at this point, composed of scalar values.

DEFINITION 7. *Given a stream predicate result* $P_S\{S\}$ *with element set* E_P *as defined previously, a* stream of element indices S_I *is a stream* $(f_I, d_I, t_I, \triangle t_I, n_I, E_I)$ *whose element set* $E_I = \{e \mid e = O\forall(O, T, R, V) \in E_P\}$.

DEFINITION 8. *The stream of element indices* S_I *constructed from a given stream* S *produces a* stream of ranges $S_{[I]}$ *such that the elements* (O, T, R, V) *of* $S_{[I]}$ *have* $V = [O_j \ldots O_k]$ *where* $P_S(S_O(i))$ *is* **true** $\forall i \in [O_j \ldots O_k]$.

Given $S_{[I]}$ for a particular predicate P_S, queries of the following forms may be answered without having to perform a complete scan of every instance of stream S in the database:

- Retrieve streams with elements that satisfy P_S (such as the modified Query 10).
- Retrieve the elements of some stream (perhaps specified by a stream entity predicate) that satisfy P_S (such as Query 12 and Query 14).
- Any other query that builds upon elements of a stream that satisfy the predicate P_S (such as Query 17).

The index $S_{[I]}$ can be constructed and maintained as stream instances are added to the database or deleted from it. If higher level indices have been constructed over $S_{[I]}$, these indices must also be maintained as well.

5.4.2 Streams of Satisfied Stream Element Predicates

The creation of streams of ranges $S_{[I]}$ facilitates the querying of streams based on the truth or falsehood of a single stream element predicate P_S. This approach results in one index for each predicate P_S of interest to the database's designers and target users. An alternative approach which constructs one index for a set of predicates is discussed in this section.

DEFINITION 9. *A* stream element predicate partition φ *for some stream* S *is a set of stream element predicates* P_S *such that* $\forall$ *elements* e *of* S, $P_S(e)$ *is* **true** *for no more than one* $P_S \in \varphi$.

Examples of these predicate partitions include $\{< x, > x, = x\}$ where x is some pivotal value (frequently zero, for streams that track rates of growth) and $\{= v_1, = v_2, \ldots, = v_n\}$ where $\{v_1, v_2, \ldots, v_n\}$ constitutes a finite range of values that can be taken on by some attribute of a stream's element. Query 11 and Query 12, which are interested in size changes, would benefit from such a partition. To index the growth or shrinkage of a tumor, the predicate partition $\{< 0, > 0, = 0\}$ may be used for that tumor's volume or size attributes.

The idea behind stream element partitions is to construct, for each stream instance, a sequence of element ranges for which some predicate in the partition is true. As with the previous approach, a higher-level index can then be constructed over these smaller streams of element ranges to streamline query processing even further.

DEFINITION 10. *Given a stream element predicate partition* φ *for some stream* S, *a* stream of satisfied stream element predicates *is a stream whose element set* E_φ

consists of element tuples $(O_\varphi, T_\varphi, R_\varphi, V_\varphi)$ *where* $V_\varphi = (P_S, [O_j \ldots O_k])$, *with* $P_S \in \varphi$ *and* $[O_j \ldots O_k]$ *is a range of indices for accesses by order in stream S such that* $\forall S_O(o)$, $o \in [O_j \ldots O_k]$, $P_S(S_O(o))$ *is* `true`.

Query results from Query 12 would form a set of streams of satisfied stream element predicates. In this case, the partition φ is $\{\leq 0, > 0\}$ and the ranges $[O_j \ldots O_k]$ consist of elements that satisfy the ≤ 0 predicate prior to three months post treatment, then satisfy > 0 after that.

Note that our definitions permit the condition where *no* predicate in φ is true for a given element. This possibility is eliminated by choosing the predicates in φ such that $P_S(e)$ is `true` for one and only one $P_S \in \varphi$.

We observe, without presenting the complete (and lengthy) details, that satisfied stream element predicate streams are semantically related to the range streams defined in the previous section. Simply put, predicate streams can be constructed from a set of range streams and vice versa, as long as the stream element predicates involved satisfy the conditions in Definition 9. Though this may suggest, conceptually, that both stream indexing approaches require the same amount of storage and maintenance, differences in the actual data content of stream instances may make one approach better than the other.

6 Prototype Implementation

We have implemented the data model in this document with an interactive, visual schema design tool designed to support the entire M data model specification. The VisualWorks Smalltalk development environment from ParcPlace / Digitalk Inc. was used for this prototype.

6.1 Implementation Details

Schema diagrams are committed to a Gemstone database system, which is being extended in order to accommodate the more advanced data modeling constructs presented in this paper. Visualization and presentation can be done on VisualWorks, but can also use specialized packages such as IDL (Interactive Data Language) from Research Systems Inc.

An MQuery prototype has also been developed on the same platforms. At this point, MQuery can answer simple visual queries, including the brain and lesion objects for Query 3 and illustrated in Fig. 11. In PICQUERY+, which is a tabular and less visual subset of MQuery, we can answer thus far Query 1, Query 2, Query 12, Query 13, and Query 15.

The thermal ablation therapy application presented herein is discussed in depth in [21]. The initial data model and prototype use a subset of the M data model, and can perform queries on various brain tumors based on size and rate of growth. Query results are linked to an IDL 3D image visualization tool. Another major and concurrent effort is in thoracic oncology, including the necessary data model, timeline interface, and visualization facilities [2], [22].

6.2 Data Model Evaluation

The data model notation, and software implemented thus far, has been tested and evaluated by users with varied levels of formal database training (sometimes none at all). This section provides highlights of the testing process that was conducted. A full account, including more details and graphs of the testing results, is provided in a separate document [20].

Methodology. Users were tested in the areas of schema comprehension and definition or design. To separate the data model's usability from the software's usability, these tests were each conducted twice: once using pencil and paper, and again using the software prototype. Hands-on sessions were videotaped for later analysis.

After objective testing concluded, users were given a questionnaire where they provided subjective reactions to the data model and software. The questionnaire covered topics ranging from the usability of the data model to its potential application to real-world problem domains. Free-text comments were also solicited.

Results. In general, user feedback was very positive, particularly with regard to the ease of use of the model and notation and the ease by which multimedia types can be visualized and understood in schemas diagrammed with the model. On an objective level, users performed well in accomplishing the tasks given to them. Interpretations of schema diagrams were generally accurate and complete, as were the designs generated by the users with the data model notation. Subjectively, user responses in the questionnaire indicated a high degree of satisfaction with the data model.

7 Conclusions and Future Work

We described the M data model which provides the following new constructs: an overall multimedia type model that organizes all multimedia types under a single hierarchy and layered structure, intelligent visual entities that are capable of answering queries on their appearance without depending on alphanumeric indexing, a stream construct that unifies data modeling requirements for time-based information, and a data model for simulation and visualization that takes advantage of our multimedia and stream models. Logical schemas defined in M are capable of capturing multimedia, simulation, temporal, and evolutionary data so that they may be represented, queried, and visualized.

M has an accompanying query language, MQuery, that takes advantage of the data model's special constructs. We showed examples of the query requirements in a number of application domains.

We indicate that prototype databases have been designed and implemented for thoracic oncology [2] and brain tumor thermal ablation therapy [21], illustrating a number of highlights. We are pursuing further implementation and use of the features presented in this article; modules that have been implemented thus far include schema design and basic querying and visualization. The schema design module already supports the complete data

model, including multimedia types, intelligent visual entities, streams, and simulation.

All told, the data modeling concepts introduced herein 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 includes finalizing the detailed specification, implementation, and prototype testing of several parts of *MQuery*. MQuery will be visual in nature, and it will take full advantage of the special constructs defined by our model. Other research directions include advanced visualization, new interface paradigms (i.e., gestures, virtual reality, etc.), and the inclusion of hypertext or hypermedia data into the overall model.

ACKNOWLEDGMENTS

The authors thank the many colleagues, collaborators, and consultants who have motivated, inspired, and contributed to this work. Dr. Wesley W. Chu from the Computer Science Department and Dr. Ricky K. Taira from the Department of Radiological Sciences are the coprincipal investigators with Alfonso F. Cárdenas of the UCLA KMeD project. Drs. Denise R. Aberle, Gary R. Duckwiler, Jonathan Goldin, Robert B. Lufkin, Michael F. McNitt-Gray, and Fernando Viñuela of the UCLA School of Medicine have been invaluable in developing and evaluating the database requirements and proof-of-concept prototypes of real-world medical applications such as cerebral aneurysm embolization, thermal ablation therapy, and thoracic oncology imaging. Portions of this work were supported by grants from the National Science Foundation.

REFERENCES

[1] Y. Anzai, A. DeSalles, K. Black, K. Farahani, S. Sinha, D. Castro, and R.B. Lufkin, "Interventional MRI," *Radiographics*, 1993.

[2] D.R. Aberle, J.D.N. Dionisio, M.F. McNitt-Gray, R.K. Taira, A.F. Cárdenas, J.G. Goldin, K. Brown, R.A. Figlin, and W.W. Chu, "Integrated Multimedia Timeline of Medical Images and Data for Thoracic Oncology Patients," *Radiographics*, vol. 16, no. 3, pp. 669-681, May 1996.

[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," *Am. J. Neuroradiology*, vol. 16, no. 1, pp. 39-48 (discussion, pp. 49–52), Jan. 1995.

[4] Apple Computer, "QuickTime," *Inside Macintosh*, Addison-Wesley, 1993.

[5] E. Bertino, M. Damiani, and P. Randi, "An Approach to Integrate Multimedia Data in a Knowledge Representation System," T. Catarci, M.F. Costabile, and S. Levialdi, eds., *Proc. Int'l Workshop Advanced Visual Interfaces*, pp. 109-123, Rome, World Scientific, May 1992.

[6] E. Bertino, M. Negri, G. Pelagatti, and L. Sbattella, "Object-Oriented Query Languages: The Notion and the Issues," *IEEE Trans. Knowledge and Data Eng.*, vol. 4, no. 3, pp. 223-237, June 1992.

[7] A. Del Bimbo, E. Vicario, and D. Zingoni, "Symbolic Description and Visual Querying of Image Sequences Using Spatio-Temporal Logic," *IEEE Trans. Knowledge Data Eng.*, vol. 7, no. 4, pp. 609-622, 1995.

[8] W.W. Chu and Q. Chen, "A Structured Approach for Cooperative Query Answering," *IEEE Trans. Knowledge and Data Eng*, vol. 6, no. 5, pp. 738-749, Oct. 1994.

[9] W.W. Chu and K. Chiang, "Abstraction of High Level Concepts from Numerical Values in Databases," *Proc. AAAI Workshop Knowledge Discovery in Databases*, 1994.

[10] M. Chock, A.F. Cárdenas, and A. Klinger, "Manipulating Data Structures in Pictorial Information Systems," *Computer*, pp. 43-50, Nov. 1981.

[11] W.W. Chu, Q. Chen, R.-C. Lee, "Cooperative Query Answering via Type Abstraction Hierarchy," S.M. Deen, ed., *Cooperating Knowledge Based Systems*, Elsevier, 1990.

[12] *Workshop Advances in Data Management for the Scientist and Engineer*, W.W. Chu, A.F. Cárdenas and R.K. Taira, eds., National Science Foundation, Boston, Feb. 1993.

[13] W.W. Chu, A.F. Cárdenas, and R.K. Taira, "KMeD: A Knowledge-Based Multimedia Medical Distributed Database System," *Information Systems*, vol. 20, no. 2, pp. 75-96, 1995.

[14] A.F. Cárdenas, I.T. Ieong, R.K. Taira, R. Barker, and C.M. Breant, "The Knowledge-Based Object-Oriented PICQUERY+ Language," *IEEE Trans. Knowledge and Data Eng.*, vol. 5, no. 4, pp. 644-657, Aug. 1993.

[15] 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," L.-Y. Yuan, ed., *Proc. 18th Int'l Conf. Very Large Databases*, pp. 53-64, Vancouver, Canada, Morgan Kaufmann, Aug. 1992.

[16] *Research Foundations in Object–Oriented and Semantic Database Systems*, A.F. Cárdenas and Dennis McLeod, eds., Prentice Hall, 1990.

[17] W.W. Chu, M.A. Merzbacher, and L. Berkovich, "The Design and Implementation of CoBase," *Proc. ACM SIGMOD*, pp. 517-522, Washington, D.C., 1993.

[18] D. Castro, R.E. Saxton, and R.B. Lufkin, "Interstitial Photoablative Laser Therapy Guided by Magnetic Resonance Imaging for the Treatment of Deep Tumors," *Seminars of Surgical Oncology*, vol. 8, pp. 233–241, 1992.

[19] J.D.N. Dionisio and A.F. Cárdenas, "MQuery: A Visual Query Language for Multimedia, Timeline, and Simulation Data," *J. Visual Languages and Computing*, vol. 7, pp. 377-401, 1996.

[20] J.D.N. Dionisio and A.F. Cárdenas, *A Methodology for User Evaluation of Visual Schema Designers and Query Languages*, under review, 1998.

[21] J.D.N. Dionisio, A.F. Cárdenas, R.B. Lufkin, K.L. Black, R.K. Taira, and W.W. Chu, "A Multimedia Database System for Thermal Ablation Therapy of Brain Tumors," *J. Digital Imaging*, vol. 10, no. 1, pp. 21-26, Feb. 1997.

[22] J.D.N. Dionisio, A.F. Cárdenas, R.K. Taira, D.R. Aberle, W.W. Chu, M.F. McNitt-Gray, J.G. Goldin, and R.B. Lufkin, "A Unified Timeline Model and User Interface for Multimedia Medical Databases," *Computerized Medical Imaging and Graphics*, vol. 20, no. 4, 1996.

[23] Dept. of Health and Human Services, National Institutes of Health, National Library of Medicine, *UMLS Knowledge Sources*, Aug. 1992.

[24] D. Le Gall, "MPEG: A Video Compression Standard for Multimedia Applications," *Comm. ACM*, vol. 34, no. 4, pp. 46-58, Apr. 1991.

[25] S. Gibbs, C. Breiteneder, and D. Tsichritzis, "Data Modeling of Time-Based Media," D. Tsichritzis, ed., *Visual Objects*, pp. 1-22, Centre Universitaire d'Informatique, Univ. of Geneva, 1993.

[26] A. Gupta, T. Weymouth, and R. Jain, "Semantic Queries with Pictures: The VIMSYS Model," G.M. Lohman, A. Sernadas, and R. Camps, eds., *Proc. 17th Int'l Conf. Very Large Databases*, pp. 69-79, Barcelona, Spain, Very Large Data Base Endowment, Morgan Kaufman, Sept. 1991.

[27] M.R. Harreld, "Brain Aneurysm Blood Flow: Modeling, Simulation, VR Visualization," PhD thesis, Univ. of California, Los Angeles, 1996.

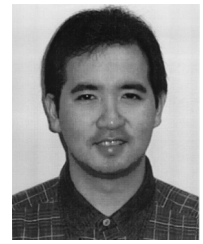
[28] L. Hardman, D.C.A. Bulterman, and G. van Rossum, "The Amsterdam Hypermedia Model: Adding Time and Context to the Dexter Model," *Comm. ACM*, vol. 37, no. 2, pp. 50-63, Feb. 1994.

[29] I.T. Ieong, "Data Modeling and Query Processing for Image Management," PhD thesis, Univ. of California, Los Angeles, 1993.

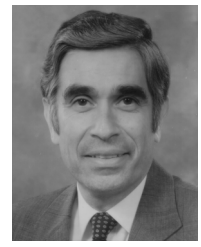
[30] H. Ishikawa, F. Suzuki, F. Kozakura, A. Makinouchi, M. Miyagishima, Y. Izumida, M. Aoshima, and Y. Yamane, "The Model, Language, and Implementation of an Object-Oriented Multimedia Knowledge Base Management System," *ACM Trans. Database System*, vol. 18, no. 1, pp. 1-50, Mar. 1993.

[31] *Proc. NSF Workshop Visual Information Management Systems*, R. Jain, ed., Feb. 1992.

[32] P.R. Keller and M.M. Keller, *Visual Cues: Practical Data Visualization*, IEEE CS Press, Los Alamitos, Calif., 1993.
[33] V.M. Markowitz and A. Shoshani, "Representing Extended Entity-Relationship Structures in Relational Databases: A Modular Approach," *ACM Trans. Database Systems*, vol. 17, no. 3, pp. 423-464, Sept. 1992.
[34] E. Oomoto and K. Tanaka, "OVID: Design and Implementation of a Video-Object Database System," *IEEE Trans. Knowledge and Data Eng.*, vol. 5, no. 4, pp. 629-643, Aug. 1993.
[35] J.M. Pratt and M. Cohen, "A Process-Oriented Scientific Database Model," *SIGMOD Record*, vol. 21, no. 3, pp. 17-25, Sept. 1992.
[36] E. Rose and A. Segev, "A Temporal Object-Oriented Algebra and Data Model," technical report, Lawrence Berkeley Lab., June 1992.
[37] E. Sciore, "Object Specialization," *ACM Trans. Information Systems*, vol. 7, no. 2, pp. 103-122, Apr. 1989.
[38] M. Stonebraker, J. Chen, N. Nathan, C. Paxson, and J. Wu, "Tioga: Providing Data Management Support for Scientific Visualization Applications," *Proc. 19th Int'l Conf. Very Large Databases*, R. Agrawal, S. Baker, and D. Bell, eds., pp. 25–38, Dublin, Ireland, Very Large Data Base Endowment, Morgan Kaufmann, Sept. 1993.
[39] D. Swanberg, C.-F. Shu, and R. Jain, "Knowledge Guided Parsing in Video Databases, *Proc. Symp. Electronic Imaging*: *Science and Technology*, K.T. Knox and E. Granger, eds., San Jose, Calif., Soc. for Imaging Science and Technology and International Soc. for Optical Engineering, Jan.-Feb. 1993.
[40] Y. Tonomura, A. Akutsu, K. Otsuji, and T. Sadakata, "VideoMAP and VideoSpaceIcon: Tools for Anatomizing Video Content," *Proc. INTERCHI*, S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel, and T. White, eds., pp. 131-136, Amsterdam, ACM, Apr. 1993.
[41] H. Ueda, T. Miyatake, S. Sumino, and A. Nagasaka, "Automatic Structure Visualization for Video Editing," *Proc. INTERCHI*, S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel, and T. White, eds., pp. 137-141, Amsterdam, ACM, Apr. 1993.



John David N. Dionisio received his PhD degree in computer science at the University of California at Los Angeles in 1996. His dissertation, "An Integrated Data Model, Language, and User Interface for Knowledge, Multimedia, and Simulations," is based on multimedia database modeling and query languages, with particular interest in the medical domain. As a graduate student researcher, he was a member of the Knowledge-Based Multimedia Medical Distributed Database (KMeD) development team headed by Wesley W. Chu, Alfonso F. Cárdenas, and Ricky K. Taira. He is now the director of technology of the Telemedicine/ITMedicine Division of the Department of Radiological Sciences at UCLA. He is a member of the ACM, and has been inducted into the Pi Mu Epsilon mathematics honor society, the Alpha Sigma Nu Jesuit honor society, and the Tau Beta Pi engineering honor society.



Alfonso F. Cárdenas received the BS degree from San Diego State University, and the MS and PhD degrees in computer science from the University of California at Los Angeles in 1969. He is now a professor in the Computer Science Department of the School of Engineering and Applied Sciences at UCLA, and a consultant in computer science and management for the Computomata International Corporation. His major areas of research interest include database management, distributed multimedia (text, image/picture, voice) systems, information systems planning and development methodologies, and software engineering automation. He has been a consultant to users and vendors of hardware and software technology. He has served as chair and a member of organizational and program committees for many conferences, and has led many seminars and spoken before audiences in various countries. He is past-president of the Board of Trustees of the Very Large Data Base Endowment. He has been a member of review boards for the National Science Foundation, the National Institutes of Health, and various other institutions. He has authored numerous articles, and authored and/or edited three books.