Knowledge-Based Image Retrieval with Spatial and Temporal Constructs*

Wesley W. Chu, Chih-Cheng Hsu, Alfonso F. Cárdenas, and Ricky K. Taira
University of California, Los Angeles, CA 90095

wwc@cs.ucla.edu, ccheng@us.ibm.com, cardenas@cs.ucla.edu, rtaira@mail.rad.ucla.edu

Abstract

A knowledge-based approach to retrieve medical images by feature and content with spatial and temporal constructs is developed. Selected objects of interest in a medical image (e.g. x-ray, MR image) are segmented, and contours are generated from these objects. Features (e.g. shape, size, texture) and content (e.g. spatial relationships among objects) are extracted and stored in a feature and content database. Knowledge about image features can be expressed as a hierarchical structure called a Type Abstraction Hierarchy (TAH). The high-level nodes in the TAH represent more general concepts than low-level nodes. Thus, traversing along TAH nodes allows approximate matching by feature and content if an exact match is not available. TAHs can be generated automatically by clustering algorithms based on feature values in the databases and hence are scalable to large collections of image features. Further, since TAHs are generated based on user classes and applications, they are context- and user-sensitive.

A knowledge-based semantic image model is proposed which consists of four layers (raw data layer, feature and content layer, schema layer, and knowledge layer) to represent the various aspects of an image objects' characteristics. The model provides a mechanism for accessing and processing spatial, evolutionary, and temporal queries.

A knowledge-based spatial temporal query language (KSTL) has developed which extends ODMG's OQL and supports approximate matching of feature and content, conceptual terms, and temporal logic predicates. Further, a visual query language has been developed that accept point-click-and-drag visual iconic input on the screen which is then translated into KSTL. User models are introduced to provide default parameter values for specifying query conditions.

We have implemented a Knowledge-Based Medical Database System (KMeD) at UCLA, and currently under evaluation by the medical staff. The results from this research should be applicable to other multimedia information systems as well.

*This work is supported in part by the National Science Foundation Scientific Database Initiative Grant IRI9116849, and also in part by ARPA contract F30602-94-C-0207

1 Introduction

A new generation of intelligent database systems must emerge, in order to effectively disseminate, integrate, retrieve, correlate, and visualize multimedia medical information. Many medical Picture Archiving and Communication Systems (PACS) [HT92] have several terabytes of image data on-line, yet utilization of data for research and teaching is very limited. Search engines for medical images are needed that can support content-based image retrieval. Exactly matched content-based retrieval will almost never yield any answer; further, it is especially difficult when the task involves searching for image data that contain imprecise descriptors. To illustrate some of these difficulties, consider the following queries:

Query 1 Find patients with similar lesions to that of patient with ID "P000-01" based on their shape and locations.

Query 2 Find African-American patients with similar post-therapy lesion development to the image sequence shown on the screen.

Query 3 Find patients treated either by Dr. Smith or Jones whose lesions exhibit a decrease in size by at least 50% for every scheduled examination.

Query 4 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.

Query 5 Retrieve the image frames in which a micro-lesion is nearby the lateral ventricle and approximately 9 mm in diameter. The micro-lesion evolves into a macro-lesion with diameter equal or larger than 25 mm and invades the lateral ventricle in approximate one year.

Some of the problems that must be addressed include: 1) How do we communicate these natural language queries to a computer system? Conventional database query languages are limited in their expressibility, especially concerning fuzzy correlations (e.g. in the vicinity of), spatial concepts (e.g. INSIDE, NEARBY, FAR AWAY), and temporal concepts (e.g. size doubled since initial diagnosis); 2) What methods are used to match cases stored within the raw image repository (e.g. PACS) to the conditions specified in the user query? The system needs to be able to mimic the response of a trained expert in the medical field. That is, the system must know of all subclasses of a specified object (e.g. a malignant tumor[1] could be an adenocarcinoma, or any number of other cancerous growths), reference variations (e.g. synonyms, lexical variants, abbreviations, etc.), conceptual terms (e.g. large, small), and context sensitive terms (e.g. SIMILAR TO); 3) How does the system intelligently relax certain constraints when no exact query solutions exist? For example, in order for the system to find a solution, it may need to relax the shape of the mass from spherical to elliptical; 4) What kind of user interface is suitable for users? A flexible user interface is required to allow the user to express the query and navigate through the presented query solutions. The user requires tools to visualize the presented answers in a specific format.

In this paper, Section 2 presents the four-layered knowledge-based semantic image model for representing medical images with spatial and temporal features. Section 3 presents the language for specifying visual query with temporal and spatial constructs. Section 4 discusses the translation of the visual query expression into the knowledge-based spatial and temporal query language (KSTL) which can be implemented as extension of ODMG's OQL. Section 5 discusses the knowledge-based query answering for approximate matching images with features and contents via relaxation. The data flow and system architecture is presented in Section 6.

2 Knowledge-Based Semantic Temporal Image Model

Currently, images cannot be easily or effectively retrieved due to the lack of a comprehensive data model that captures the structured abstractions and knowledge that are needed for image retrieval. To remedy such shortcomings, predicates should contain semantic (e.g. `INSIDE`, `FAR_AWAY`, etc.), conceptual (e.g. `large`, `small`, etc.), and *similar-to* terms in order to retrieve medical images by feature and content. The *similar-to* operators allow users to retrieve images close to the target image based on a pre-specified set of features.

An image model is proposed which consists of a Raw Data Layer (RDL), Feature and Content Layer (FCL), a Schema Layer (SL), and a Knowledge Layer (KL). Figure 1 illustrates the four-layered modeling for brain lesions.

[1]In this paper, the terms *tumor, lesion, neoplasm,* and *adenocarcinoma* are used interchangably

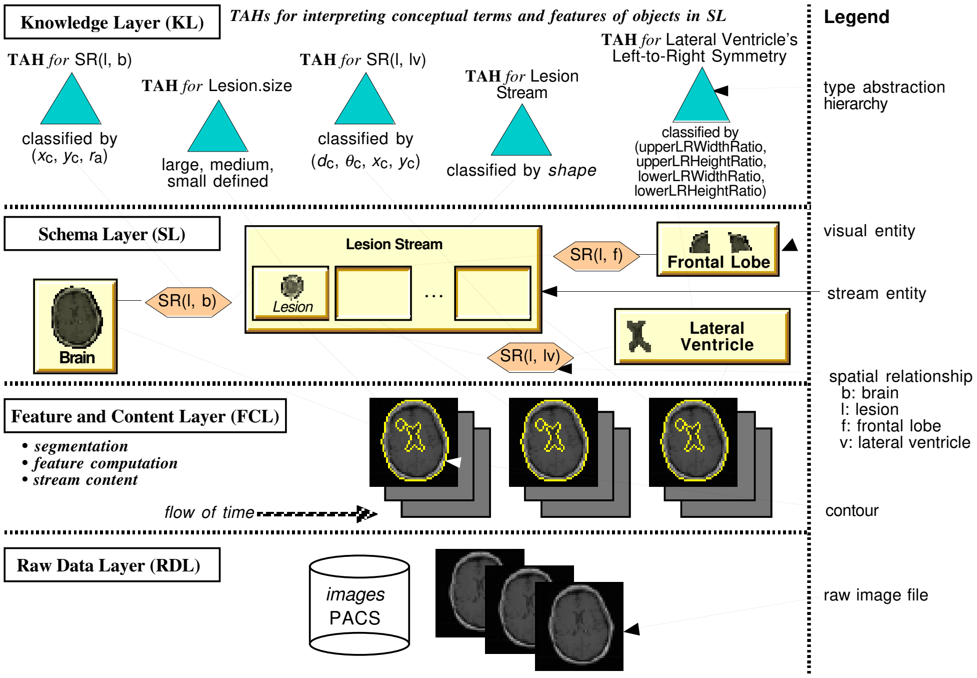

Figure 1: An example representation of brain lesions. SR(l, b), SR(l, lv), and SR(l, f) represent the spatial relationships of lesion and brain, lesion and lateral ventricle, and lesion and frontal lobe, respectively. The lesion stream entity represents a temporal sequence of lesions.

2.1 Raw Data Layer

The Raw Data Layer stores the actual pixel-level image data in the database. Its main function is to abstract from the rest of the model encoding techniques such as compression, encryption, or special formatting. When images are accessed from the Raw Data Layer, they are translated by the RDL into a canonical image format. Therefore, image data formatting is uniform at any layer above the Raw Data Layer, with low-level concerns such as compression algorithms effectively hidden and autonomous from the higher-level components of the system.

2.2 Feature and Content Layer

At the Feature and Content Layer, we store image features such as contours, spatial relationship characteristics, and temporal sequences. The contours can be segmented manually, semi-automatically (using techniques like active contours [DV+93] and liquid transforms in [NB+93]), or automatically [LTSK95, WLK92, WA+95] depending on the contrast and separability of the image objects. Raw image segmentation will be provided by the radiologists [LTSK95]. The segmented contours are used to compute spatial and temporal features, after which they are integrated into the proposed model for querying images by content and feature.

2.2.1 Spatial Feature Computation

Shape Modeling Image objects in medical images are often complex in shape and require detailed comparison on specific portions. Thus, we propose a decomposition approach to describe an object's shape. A object with a complex shape is first decomposed into context-dependent substructures. The decomposition is based on the fundamental line and curve segments identified by the generated $\psi - s$ function from the chain code of the object contours [MA79]. These decomposed fundamental lines and curves are then matched to a shape model of the specific objects. These models encapsulate the "part-of" knowledge between the lines and curves that constitute the object model. The shape models are used to identify and verify contoured objects that conform to the general expected object shapes. They are also used to locate key landmark features to aid in further feature analysis. Instance data is matched to model data using an interpretation tree that utilizes unary and binary spatial constraints [Gri90].

Spatial Relationship Modeling In modeling spatial relationships, existing semantic constructs such as *overlap* and *separate* [Fra88, CYD88] are insufficient to fully represent spatial relationships among objects [HCT96]. Additional detailed variations on spatial relationships should be captured in spatial relationship modeling. To distinguish images based on *similar* spatial relationships, relevant spatial relationship features are specified by domain experts. For example, the spatial relationship for a lesion that is near another object can be captured using the distance of the centroids of the two contours on the x-axis and y-axis, the angle of coverage (the angle for viewing a contour from the centroid of another contour), and the ratio of area to classify the spatial relationship [HCT96].

These computed features will be classified automatically with clustering algorithms into type abstraction hierarchies (TAH) (see Section 2.4) and conceptual terms are annotated at the TAH nodes to enable image querying via conceptual predicates.

2.3 Schema Layer

In the Schema Layer, we construct a database schema that represents the entities and spatial relationships among objects based on the extracted shape and spatial relationship

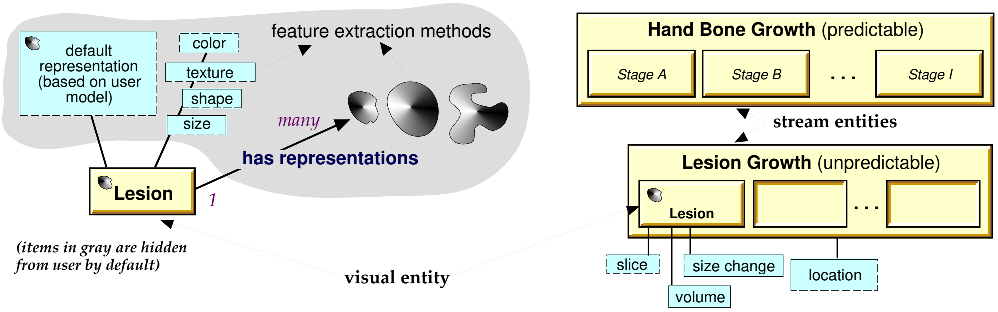

Figure 2: Visual notation of visual entities and streams, used in both schema diagrams and the visual query language.

features from object contours in the Feature and Content Layer. Other entities in this layer capture the temporal and evolutionary aspects of the database.

2.3.1 Visual Entities

Objects in an image are represented in the Schema Layer as *visual entities* (VEs). Instances of VEs consist of conventional attributes (e.g. patient ID, date, doctor name, etc.) in the raw data layer as well as visual attributes (e.g. shape, size, texture, etc.) of object contours in the Feature and Content Layer. *Spatial relationships* among entities can also be represented in the schema, as shown in Figure 2.

VEs maintain a list of canonical visual representations derived from actual images. Their features and content are extracted by the methods described in Section 2.2.

2.3.2 Stream Entities

Multiple versions of an object over a period of time (for example, the stages undergone by a tumor during the cancer process of a particular patient) can be linked to form a *stream entity* for that time period.

A stream is a sequence of ordered objects [DC97] which can be classified into *simple streams* and *composite streams*. The ordered objects in a simple stream are all regular objects, and a composite stream may have other streams as its contained objects. A composite stream is composed spatially or temporally from simple streams by *composition constructors*, such as synchronization, concatenation, and evolutionary constructors [CIT94]. Composite streams allow cross-sectional views across streams of different data to be viewed and queried at the same time (Figure 2).

2.4 Knowledge Layer

We propose to classify image shapes and spatial relationship features into a hierarchical structure known as a type abstraction hierarchy (TAH) [CC94]. The attributes used for classification are context and user sensitive, and are specified initially by the user when generating the TAHs. Spatial concepts are represented as feature value ranges. Higher nodes in the TAH represent more generalized concepts (i.e. wider range of feature values) than that of the lower nodes (i.e. narrower range of the feature values). TAH nodes generated based on shape features can be labeled with conceptual terms (e.g. `large`, `small`, etc.), and TAH nodes generated based on spatial relationship features can be labeled with semantic spatial relationship terms (e.g. near by, far away). The value ranges and the corresponding relaxation error are represented at each TAH node. Relaxation error is the expected pairwise feature distance between members in a TAH node. Hence the error is a measure of the closeness of the images under this node. Images under the same TAH node have similar characteristics with respect to a set of features and thus all of the images under the same TAH node can be considered as similar (see Figure 3). By traversing up and down the nodes in the TAH (i.e. generalization and specialization of features in the TAH, respectively), we are able to select the best node that approximately matches the target feature values. Therefore, this provides a way to process queries with *similar-to* operators.

TAH can be automatically generated via MDISC clustering algorithm [CCHY96] which classify images with shapes and/or spatial relationships. A TAH generated based on spatial relationship features is shown in Figure 3.

In addition, an ontology of spatial relationship terms (Figure 4) is used to guide the generation and labeling of TAH nodes. Similar listings of commonly used terms for shapes have been developed jointly with domain experts (e.g. radiologists). Users may edit the TAH by deleting and moving TAH nodes to satisfy the users' desired representation.

We can thus derive features from contoured objects and classify all of their images into a TAH structure based on these features. The TAHs can be used for visual query interpretation and knowledge-based query processing. A TAH directory will be developed which can be indexed on TAH characteristics (e.g. attributes used, context, user type) to retrieve TAHs for reuse.

Visual entities, spatial relationships, and stream entities in the Schema Layer are linked to TAHs in the Knowledge Layer to provide preset values for conceptual terms

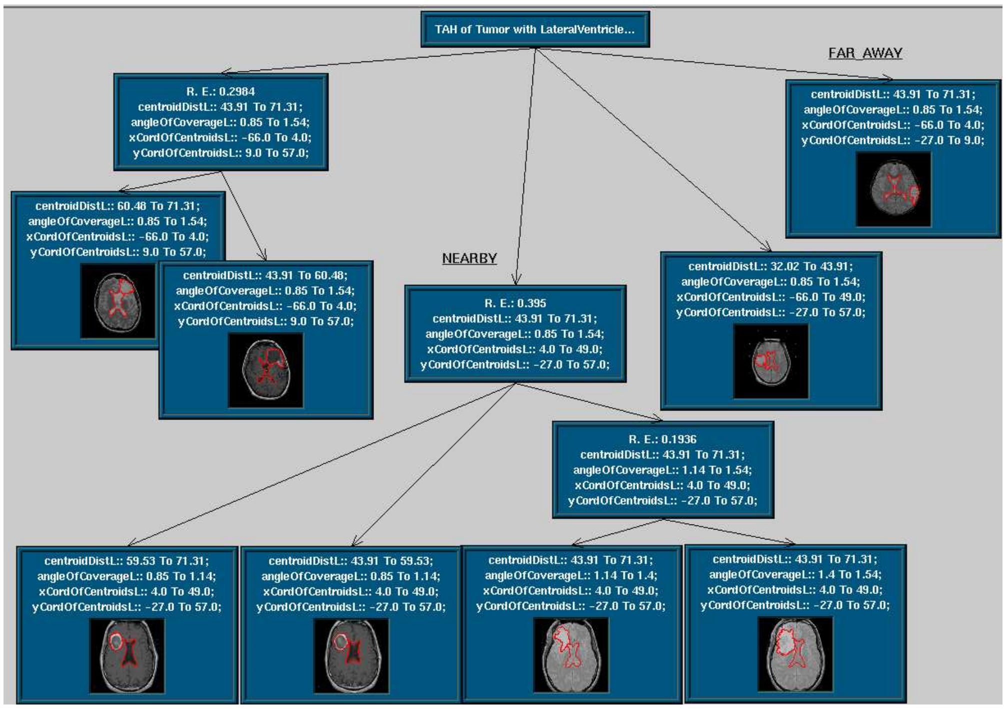

Figure 3: A type abstraction hierarchy classifies MR images based on the spatial relationship of tumors and lateral ventricle. The attributes used in the TAH are context and user sensitive, and may be pre-specified by the user.

and relaxed ranges (see Figure 1). This knowledge is used for modifying and relaxing a user's query input into the approximate or relaxed query conditions.

2.5 User Model

User models maintain profiles of object matching preferences and relaxation control policies for different user classes (Figure 5). The user model consists of lists of image objects, features representing the objects and spatial relationships, object matching policies, and policies for relaxing query conditions when no satisfactory answer is found. When object

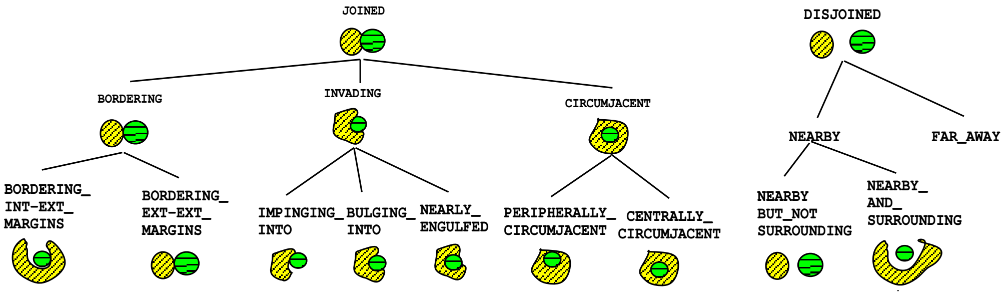

Figure 4: An ontology for semantic spatial relationship operators for topological categories between two objects with the representative icons shown.

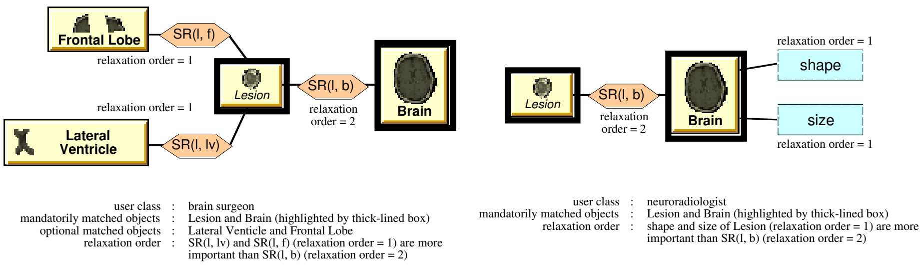

Figure 5: Example of user models for two user classes for retrieving images with a brain lesion.

matching and relaxation control policies are not explicitly specified by the user during a query, such information may be obtained from the corresponding user model to guide the relaxation of query constraints. Objects in the user model are divided into *mandatorily matched objects* which must be matched with the query context for the user model, and *optionally matched objects* which provide guidance for additional matched features to enhance the query constraints.

For example, in studying images with lesion(s), a typical concern of brain surgeons are lesion locations and their spatial relationships with other objects in the brain, while the concerns of brain radiologists include shape, size, and location of lesions in the brain (Figure 5).

When visual query is used to query similar images, the VE identifies the objects of interest as well as the user class. The knowledge-based query processor dynamically matches user models based on the identified objects and user class in the VE to provide the information required for knowledge-based query answering (e.g. features of similarity and relaxation policy). Thus, the VE and user model can be used to customize query processing for different types of users.

3 Visual Query Language

The visual query language, M Query language [DC97], is an extension from the PIC-QUERY+ language [CIT+93]. A subset of the notation for the language is shown in Figure 6, and is based on a combined entity-relationship and object-oriented data model that we have designed [DC97].

VE (visual entity) boxes contain any visual entities being queried (Figure 7). When the user wishes to ask a visual query, the VEs involved are selected and placed within a VE box to show their relative appearance, positioning, and sizing. The query processor interprets this box as a query predicate which constrains n-tuples of VEs to those n-

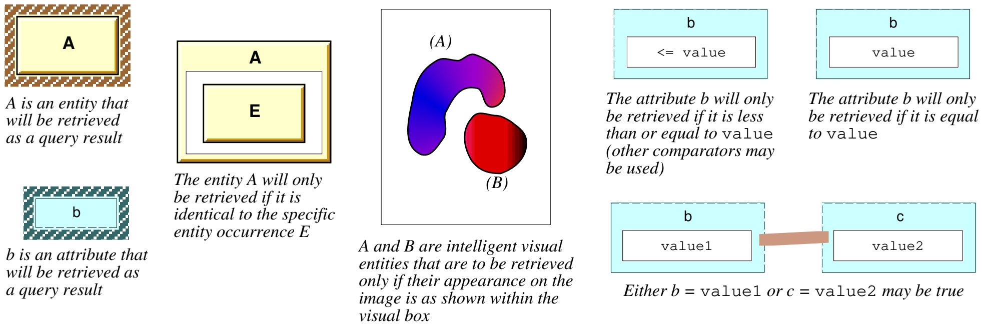

Figure 6: Visual query language notation.

tuples that are visually similar to the arrangement in the box (where n is the number of VEs placed in the box).

3.1 Visual Image Queries

In general, predicates are formed by placing values (alphanumeric or image) inside the entities or attributes for which they are intended to match. A comparison operator (such as $=$, $>$, and $<$) may be entered to specify particular types of comparisons. When a value is entered into an attribute, entity, or visual box without an explicit comparator, the system defaults the comparison to `SIMILAR_TO`.

In Figure 7, the desired similarity is based on the shape of the tumor and its spatial relationship with the lateral ventricle, as specified by shape and spatial relationship attributes. An exact match is desired for the patient ID, so an "=" is explicitly entered in the `id` attribute, while the VE box arrangement of a brain, lateral ventricle, and tumor implies a similarity comparison.

When the visual query is executed, the features of the visual entities in the query are extracted and compared based on the indicated attributes. The user can specify other similarity measures by opening a menu that describes other descriptors, such as size and roundness for the shape category, location for the distance category, angle of coverage for the nearness category, etc. Otherwise, the object matching policies in the user model are used as the default. The system can engage in a dialog with the user in cases where the visual query input may ambiguously lead to multiple query statements.

3.2 Temporal Queries

Our visual query language also supports querying of stream entities. Based on the proposed stream construct previously defined, our approach to querying this form of data

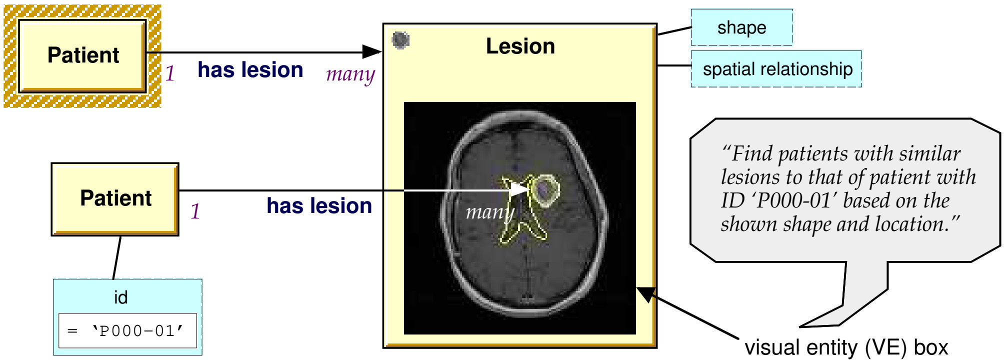

Figure 7: Visual query expression for Query 1.

corresponds directly to the user's perception of that data — as a sequence of events or data points in time, which we represent in our system as a temporal descriptive pattern.

Streams are drawn as a sequence of stream elements, enclosed by an overall stream box. Ellipses may be added in between actual stream element boxes to show that additional stream elements may exist. Time flows from left to right in this representation, so the leftmost box within a stream represents the first element of the stream, the rightmost box represents the last element, etc.

If a predicate or filter is applicable to only one element (i.e. just the first element, last element, etc.), the predicate is attached only to that element's box. If the filter is intended for a range of elements, then the predicate is attached to the set of ellipses which corresponds to that range of elements. Finally, if a predicate or filter should be applied to all elements in that stream, it is attached not to an individual stream element, but to the overall stream box. The user can edit the contents of the stream box (i.e. arrange where ellipses appear, determine how many element boxes are within the stream, add numeric time constraints, etc.) to suit the needs of the query.

Tumor size change after treatment (specified by date) can also be represented by a stream of MR images with time durations (Figure 8). Using our proposed knowledge-based query processing technique, we are able to retrieve an approximately matched sequence in the image database.

Figure 9 illustrates a sample stream query. The query combines elements from our previous PICQUERY+ work with the new visual language. The predicate (`size change` $\leq$ `-50%`) in Figure 9 is attached to the Lesion box, and so must be true for all Lesion States within the stream. The predicates in Figure 8 are VE boxes representing individual elements of the Tumor Process stream, and are applied to the stream elements whose

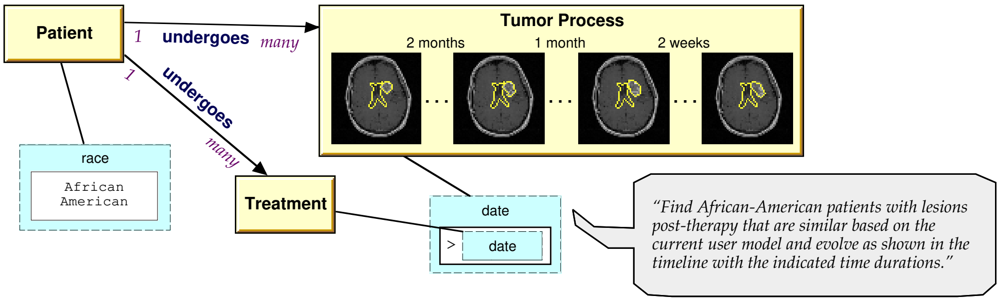

Figure 8: Visual query expression for Query 2.

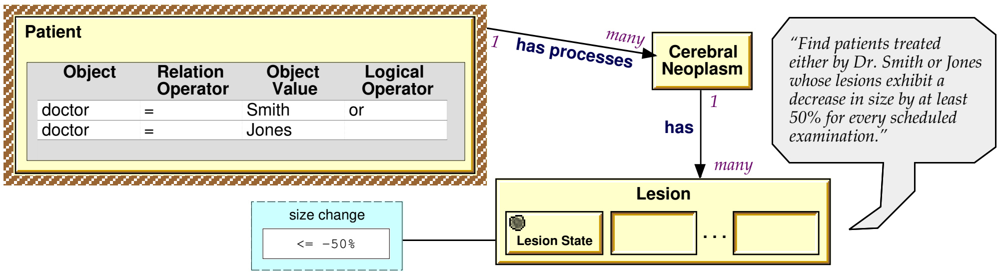

Figure 9: Visual query expression for Query 3.

time stamps fall within the specified ranges above the VE boxes. Solving this query involves translating the visual query into an algebraic form (Section 4), which can then be solved by our proposed knowledge-based query processor.

Attributes such as `size change` may also be queried on a conceptual level (i.e. `size change` is `stable` or `shows little change`) if such concepts are specified in the knowledge base in terms of actual values or value ranges.

To express a sophisticated predicate that compares multiple stream elements using different conditions over time, internal boxes of the stream notation are used to literally "draw" the way the stream is expected to change. The first and last boxes within a stream always represent its first and last elements, while any boxes placed between the first and last elements may stand for any element of the stream (including the first and last elements, if they satisfy the constraints).

Specifying the change in size is done by specifying the desired range on the `size change` attribute[2], as seen in Figure 10. To express the time constraints, we call upon the built-in `time` attribute of the `Lesion States`: we assign, to one box, the constraint that its time stamp must be within three (3) months of the date of the patient's thermal ablation

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

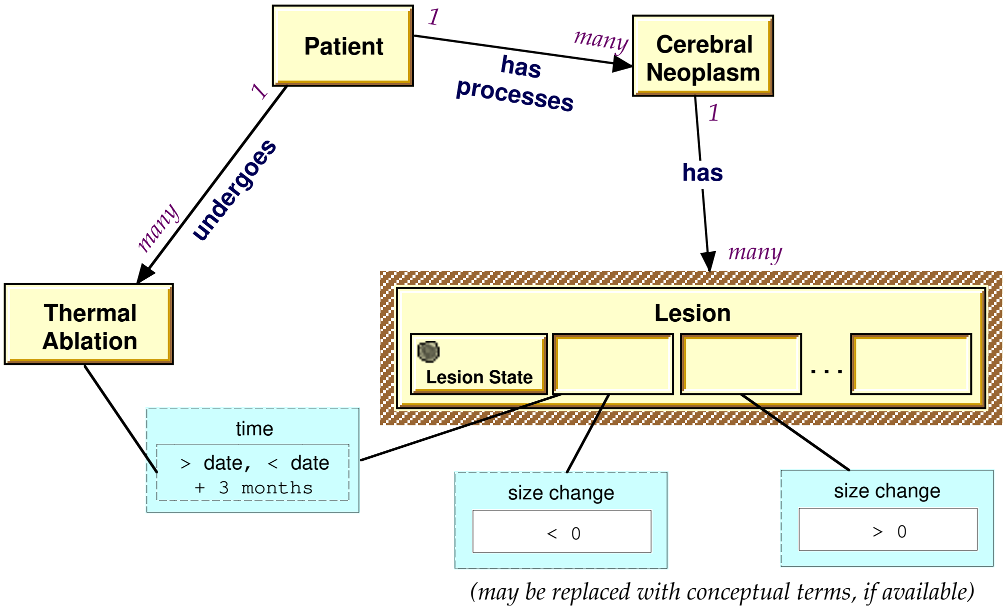

Figure 10: Visual query expression for Query 4

therapy. The inclusion of `Patient` assures that the thermal ablation treatments and lesions that are linked all belong to the same patient; without the links provided by `Patient`, any thermal ablation treatment may be matched with the lesion, regardless of their respective patients.

There is no need for an additional time predicate for the second box — by its position, it is *implicitly expected to occur after any elements that satisfy the first box*. Thus, only the size change $>$ 0 is required; the condition that this growth pattern occurs after the three-month period is already implied by the positioning of the second box to the right of the first one. The time constraint is therefore translated from an alphanumeric predicate (such as "time $<$ date + 3 `months`") to a visual one, communicated entirely by the relationship between the stream elements.

Size change may be queried on a conceptual level (i.e. size change is `stable` or `shows little change`) if a knowledge base mapping these concepts to values or value ranges is a part of the system. Similarly, Figure 11 presents the MQuery expression for Query 5.

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 language's intuitiveness or usability. Query 3 illustrates the usability of MQuery's approach in expressing multiple predicates.

Figure 9 shows how compatibility with PICQUERY+ is achieved in MQuery. A

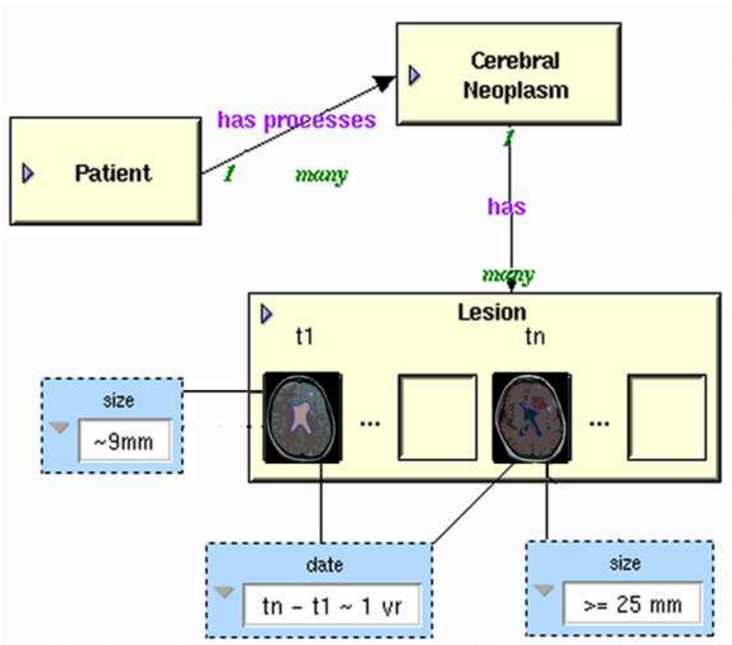

Figure 11: Visual query expression for Query 5

PICQUERY+ table is used for `Patient` instead of an entity rectangle; this is done because a table interface permits a clear, line-by-line listing of the predicates to be applied to its associated entity.

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

Query 6 *What are the volumes of the tumors that were retrieved in the previous query?*

Figure 12 presents the MQuery expression for Query 6. Query results in other windows are re-used in new queries by using copy-paste or drag-and-drop. The operation is analogous to query construction, where objects from a *schema* window are copied then pasted into a query window.

As can be seen in the figure, nested queries are achieved by replacing the contents of an entity's predicate box with one or more specific entity occurrences, thus naturally extending the more familiar functionality of placing an alphanumeric constant or comparison in an attribute's box. Figure 12 is particularly interesting because it shows how

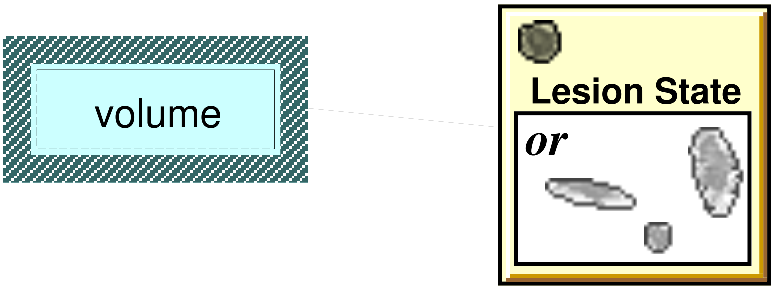

Figure 12: Incremental query for Query 6

IVEs can also be used in the same manner; note the multiple `Lesion State` objects inside the predicate box, indicating a set of `Lesion States` that have been copied from query results presumably on display elsewhere on the screen.

MQuery is implemented internally as a hierarchical data structure where each node represents a predicate or operation, along with its parameters or arguments. Then this internal representation is translated into KSTL as discussed in the following section.

4 Knowledge-based Spatial and Temporal Query Language (KSTL)

To process an MQuery expression, we need to translate it into a statement in the Knowledge-based Spatial and Temporal Query Language (KSTL). KSTL is an object-oriented query language supporting conceptual, spatial, temporal, evolutionary, and stream constructs appearing in MQuery. To leverage on available commercial technology, our KSTL can be extended from ODMG's OQL [Cat96] via methods, functions, and operators as discussed in the following.

4.1 Attribute with Conceptual Terms and Approximate Values

We use methods **IS** and **approximate** as constructs for conceptual terms and approximate values as shown in the following example.

Query 7 *'Find images with 'large'-sized tumors for patient approximately 35 years old."*

The corresponding KSTL is:

```
SELECT p.tumor.image
```

```
FROM   Patients p
WHERE  p.tumor.size.IS(``large'') AND p.age.approximate(35)
```

The set of frequently used conceptual terms for the attribute set is defined by domain experts. The approximated value for an attribute is transformed into value range(s) based on the knowledge bases which is user- and context-sensitive [CC94, HCT96].

4.2 Spatial Relationships with Semantic Terms

KSTL uses the method **srWith** to represent spatial relationships between two objects. The spatial relationships are expressed by spatial features with semantic terms. Using the **IS** method, such spatial relationships can thus be queried as follows:

Query 8 *"Retrieve all images with tumors* nearby *frontal lobe that is* invading *the lateral ventricle".*

The corresponding KSTL is:

```
SELECT t.image
FROM  Tumor t, Lateral_Ventricles l, Frontal_Lobes f
WHERE t.srWith(l).IS(``invading'') AND t.srWith(f).IS(``nearby'')
```

To process the query, the semantic terms *nearby* and *invading* are translated into specific value ranges of spatial relationship features from the TAHs [CC94, HCT96].

4.3 Object Similarity

The operator "similar-to based-on" clause can be used to find the set of objects similar to the target object. The object similarity is based on comparing the values of the specified features. Based on the knowledge base as well as the user type and target attribute values, the similar-to operator translates the specific attributes into the value ranges according to the `based_on` clause in the query [CC94, HCT96]. The answers may be ranked according to certain nearness measures (e.g., mean square error). For example,the KSTL for Query 1 is:

```
SELECT  t1.image
FROM    Tumors t1, t2
WHERE   t1.image similar_to t2.image [t2.image.ON_THE_SCREEN]
            based_on t2.location, t2.size
```

ON_THE_SCREEN is a method to select the image on the screen for processing. The similarity of images in the query is based on the location and size of tumors as specified in the query.

4.4 Evolutionary Objects

The class changes for an object in its lifespan can be: 1) changing to another class (i.e., EVOLVE_TO), 2) merging with other object(s) into a new class (i.e., FUSED_FROM), 3) splitting into objects of (different) classes (i.e., SPLIT_INTO) [CIT94]. In KSTL, the evolutionary constructs EVOLVE_TO, FUSED_FROM, and SPLIT_INTO are provided as methods to express object evolution. For example, Query 5 can be expressed by KSTL as follows:

```
SELECT  mi.image, ma.image
FROM    MicroLesions mi, MacroLesions ma, LateralVentricles lv1, lv2
WHERE   mi.evolve_to(ma) AND lv1.srWith(mi).IS(''nearby'') AND
        Interval(mi,ma).length.approximate(``1 year'') AND
        mi.diameter.approximate(9) AND
        ma.srWith(lv2).IS(''Invades'') AND
        ma.diameter >= 25mm AND lv1.patient = lv2.patient
```

In this query, the "evolve_to" construct is used to describe the tumor size as well as the spatial relationships between the tumor and lateral ventricle before and after the evolution.

4.5 Stream Behavior

A stream [DC96] is a sequence of objects. Any subset of neighboring objects in the sequence is called its *substream.* Objects in a stream are called *snapshot (objects).* A stream can be queried based on the feature values of individual snapshot objects or on the trend of feature value changes of consecutive snapshots. Most temporal database [TCG+93] and temporal relationship research [All83, LG93] has focused on the relationships among snapshots that satisfy a given set of predicates. However, constructs are needed to query streams themselves. Our extension allows streams and their substreams to be referenced in the query to specify constraints on all the snapshots.

In querying the stream characteristics, the predicates can be *snapshot predicates* and *stream predicates.* A snapshot predicate has snapshot object(s) as its operand and returns *true* if the snapshot satisfies the predicate. A stream predicate has stream(s) as its operands and returns *true* if the trend of feature value changes within snapshots satisfies the predicate.

Temporal logic has been successfully used as the expressive formalism to specify sophisticated temporal properties in concurrent programming [Pnu77] where various constructs have been proposed to capture temporal conditions. Temporal logic has also been used in temporal databases to provide the formalism for its language construction [Gab89, Eme90, BVZ95, DDI95] and has been shown to be fully expressive of queries in the historical data model [Gab89, Eme90]. Since querying data in temporal databases mostly only requires expressing temporal constraints on existing data, the requried constructs are simpler thant that of concurrent programming. Temporal database works such as [Gab89, Eme90, BVZ95, DDI95] integrate temporal logic into their languages. Their integration of temporal logic is not tightly linked with the query processing languages. On the other hand, KSTL, extended from a standard object-oriented query language, tightly integrates temporal logic semantics into its language constructs and thus can be directly processed by an object-oriented query processor. Further, in our system, the constraints on streams and snapshots are specified in the visual language (MQuery) and the (temporal) constraints are automatically translated into KSTL for query processing. The correctness of this translation via temporal logic will be shown in Section 4.5.1.

To specify the temporal property of streams, substreams, and/or the contained snapshot via temporal logic, two modal operators $\Diamond p$ and $\Box p$ are introduced to add to a snapshot predicate p. $\Diamond p$ is a new predicate which evaluates p within a stream s. $\Diamond p$ returns true if the current snapshot in s or a snapshot after the current snapshot satisfies p. $\Box p$ is another new predicate which returns true if all the snapshots after the current one (including the current snapshot) in s satisfy p. Complex temporal constraints can be expressed by combining different snapshot predicates and $\Diamond$ and/or $\Box$ operators. Predicate $\Diamond p$ and $\Box p$ not only evaluate p on the current snapshot but also on other snapshots in the stream as well. Combining predicates $p_1 p_2$ (where p_1 and p_2 may contain modal operator $\Diamond$ or $\Box$) implies that a particular snapshot exists in the evaluated stream where *all* the snapshots between this particular snapshot and the last snapshot in the evaluated stream satisfy p. We call p, $\Diamond p$, and $\Box p$ as *simple temporal logic expressions*, and expressions composed from simple temporal logic expressions as *complex temporal logic expressions*. For example,

always $p == \Box p$ (which implies that p hold all the time)

p until $q == \Box pq$ (which implies that p hold all time until the time q holds)

p since $q == (\Diamond - q)q(\Box p)$ (which implies that at the beginning q is false, and right after q turn true p holds all the time)

Current research on incorporating temporal logic into database query languages [Ric92, GM91] is mostly based on SQL which cannot provide as rich an extension on data types and variable binding as that of ODMG's OQL. In addition, these existing languages cannot express the semantics of the modal operators $\Diamond$ and $\Box$ in temporal logic. For example, to express the temporal expression $\Box pq$ in [Ric92, GM91], the beginning and ending snapshots satisfying $\Box p$ are not bound to any variable. Thus, the language cannot specify the interrelationship of snapshots occurring in different predicates which causes unclear variable binding. On the other hand, KSTL provides clear variable binding and can represent most temporal logic expressions. As a result, KSTL can be used as the processing language for MQuery.

Let $p(x)$ be a snapshot predicate expressed in KSTL. Thus, $p(x)$ corresponds to the predicate p in previous discussion and $\Diamond p$ and $\Box p$ are its two variations in temporal logic. S is a set of streams to be evaluated against temporal logic expressions for retrieving streams that satisfy the temporal logic expression. Note that the following OQL commands are useful for stream querying:

- For evaluating $\Diamond p$ on a set of stream S, we can use the *in* operator in OQL as an existing construct:

  FROM S as s, x in s WHERE $p(x)$

  Note that S is a set of streams. Thus s is bound to a stream in each iteration. Since the stream referred by s is also a collection of objects, it can appear in the FROM clause with x bound to a member object in s.

- For expressing the expression $\Box p$, we can use the `forall` expression (an existing construct in OQL) as:

  FROM S as s WHERE `forall` x in s: $p(x)$

- For expressing any ordered collection s, the OQL-supported functions $first(s)$ and $last(s)$ can be used to return the first and last object in the ordered collection.

In addition to these existing constructs in OQL, we need two new functions: *next* and *Allsubstreams* for stream querying as follows:

- *next*: for any snapshot object x, $x.next$ returns the next object of x in the same stream.
- $AllSubStream(s)$: for a stream s, $AllSubStream(s)$ returns a collection containing all the substreams in the stream s.

These two extensions support complex temporal logic expressions. $x.next$ allows us to express the concatenation of two neighboring simple temporal logic expressions. The function $AllSubStream(s)$ generates all the substreams in S and this set allows us to bind variable(s) to any candidate substream. The constraints on the substreams are then expressed via the bound variable(s) in the WHERE clause.

The following translation between the temporal logic expressions and KSTL statements illustrate how the constructs in KSTL provide querying semantics on streams based on temporal logic semantics[3].

$\Diamond p ==$ FROM S as s, x in s WHERE $p(x)$

$\Box p ==$ FROM S as s
WHERE forall x in s: $p(x)$

$\Diamond p \Diamond q ==$ FROM S as s, x in s, y in S
WHERE $p(x)$, $q(y)$, $x.before(y)$

$\Diamond p \Box q ==$ FROM S as s, x in s, $AllSubStream(s)$ as z
WHERE $p(x)$, forall y in z: $q(y)$, x.next == first(z)

$\Box p \Diamond q ==$ FROM S as s, x in s, $AllSubStream(s)$ as z
WHERE forall y in z: $p(y)$, first(z) == first(s), p(x), z.before(x)

$\Box p \Box q ==$ FROM S as s, $AllSubStream(s)$ as $z1, z2$
WHERE forall $y1$ in z: $p(y1)$, forall $y2$ in z: $q(y2)$, $first(z1) == first(s)$,
$last(z2) == last(s)$, $last(z1).next == first(z2)$

[3]The translation algorithm is provided in the appendix.

By extending OQL with the two additional functions based on temporal logic [Pnu77, Eme90], KSTL can express complex temporal constraints. KSTL provides temporal constraint expression with clearer variable binding as compared with other extended temporal query languages [Ric92, GM91]. The extended temporal querying semantics allows KSTL to specify complex stream queries and served as the processing language for MQuery.

4.5.1 Translation of MQuery into KSTL

In this section, we shall show that the visually expressed MQuery predicates can be textually represented by KSTL. There exists a direct correspondence between a MQuery expression and a temporal logic expression. Since a temporal logic expression can be translated into a KSTL statement, an MQuery expression can thus be translated into a KSTL statement.

Let's now explain the one-to-one correspondence between a MQuery statement and a temporal logic expression. In MQuery, a snapshot is represented as a box (called a *snapshot box*), the constraints attached to this snapshot box correspond to a snapshot predicate p in the temporal logic expression. A box contains a stream or substream is called a *stream box*. In MQuery, two snapshot boxes (whose predicates are represented as p and q) appearing in the same stream box can be expressed as a temporal logic expression either as $\Diamond pq$, $\Diamond p \Diamond q$, $p \Diamond q$, or pq depending on the position of the "..." symbol(s) between the snapshots in the stream boxes (for example, see Figure 8).

In MQuery, the temporal logic predicate for a stream/substream box can either be in the form of p, $\Diamond p$, or $\Box p$ depending on the constraints specified on this substream. For example, a substream whose duration lasts longer than 2 years can be expressed in the form of p or $\Diamond p$. However, requiring the size of every lesion contained in a lesion substream to continuously decrease requires a temporal logic expression of the form like $\Box p$, since we require the size of every preceeding lesion to be larger than its successor. In this case, the predicate p need to specify the condition that "the size of a preceeding lesion larger than that of its succeeding one" and the $\Box$ operator ensures that this condition is satisfied by *every* snapshot in the stream. Thus, the snapshot and stream box in MQuery both have their own temporal logic predicates and can be translated into the appropriate KSTL statement.

For the constraint boxes which attach to more than one box in MQuery, they can be translated into the KSTL predicates using the different cases distinguished in the appendix.

Using temporal logic as the bridge, the visual MQuery statement can be translated into KSTL statements for processing. In the followings, we shall provide two examples to illustrate this translation.

Query 2: "Retrieve streams that have four specific snapshots and the corresponding time periods between the snapshots as shown in Figure 8" The following is the translated KSTL:

```
SELECT  p, x1, x2, x3, x4
FROM    Patients p, p.tumorSequence as s,
           x1 in s, x2 in s, x3 in s, x4 in s,
           TumorImages t1, t2, t3, t4
WHERE   p.treatment.before(x1) AND p.race=='African American' AND
           x1 similar_to t1 [t1.imageID='001']
              based_on t1.tumor.size, t1.tumor.location AND
           x2 similar_to t2 [t2.imageID='002']
              based_on T1.tumor.size, t1.tumor.location AND
           x3 similar_to t3 [t3.imageID='003']
              based_on t1.tumor.size, t1.tumor.location AND
           x3 similar_to t4 [t4.imageID='004']
              based_on t1.tumor.size, t1.tumor.location AND
           INTERVAL[x1,x2].approximate(``2 months'') AND
           INTERVAL[x2,x3].approximate(``1 month'') AND
           INTERVAL[x3,x4].approximate(``2 weeks'')
```

Here, the method `tumorSequence` returns only one tumor sequence for each patient. If the method `tumorSequence` can return more than one sequence, additional variables should be used to ensure that the variables `x1, x2, x3, and x4` refer to the same single stream in the query. Further query optimization is left to the OQL query optimizer.

So far, we have limited the predicate p as a snapshot predicate. However, p can be generalized to a stream predicate. For example, let s be a tumor size stream. $p(s)$ denotes a stream predicate which describes whether the "size changing" pattern of s is *similar to* the pattern of a particular given target stream t. Then $\Diamond p(s)$ is a new predicate which is evaluated as true if there exists a substream in s whose "size changing" pattern similar to that of the target stream t. Due to the unclear semantics of $\Box p(s)$, this generalization is limited to $\Diamond p(s)$. The function *AllSubStreams* in KSTL returns a set containing all the candidate substreams. This returned set is used in the FROM clause of a KSTL statement to bind the variable stream s. The bound variable s is used in the WHERE clause to express the constraints $p(s)$. In MQuery, this corresponds to the constraint boxes attached to a substream box or a VE (sub)stream box (see Query 4 in Figure 10). The following is the corresponding KSTL for Query 4:

```
SELECT p, s
FROM   Patient p, p.tumorStream s, AllSubStream(s) z1, z2
WHERE  forall x1 in z1: x1.sizechange < 0  AND
       forall x2 in z2: x2.sizechange > 0  AND
       z1.length < ''3 month'' AND
       last(z1).next = first(z2) AND
       p.thermalAblation.before(z1)
```

`last` and `first` are functions in OQL which retrieve the first and last element in an ordered collection. `next` is a method in KSTL which retrieves the immediate downstream snapshot of the current snapshot. Thermal ablation is a type of therapy for treating tumors and the method `thermalAblation` returns the patient's treatment records.

Predicates can be used to express constraints on sub-streams as long as the corresponding methods are available to evaluate against streams. Since not all the sub-streams in the *AllSubStream(s)* need to be examined to evaluate the query, a query processor should optimize the evaluation of the function *AllSubStream*(s). This optimization can be accomplished by using a filtering process such as the optimization technique used in the string and list retrieval [SLVZ95]. For example, if the beginning or ending snapshot is required to satisfy certain constraints, only those sub-streams that begin or end with snapshots satisfying the constraints are examined. Due to its high execution cost, examining all candidate substreams in the *AllSubStream(s* set should always be a last resort for processing the stream predicate.

5 Query Processing

Knowledge-based query answering consists of the following three phases (Figure 13).

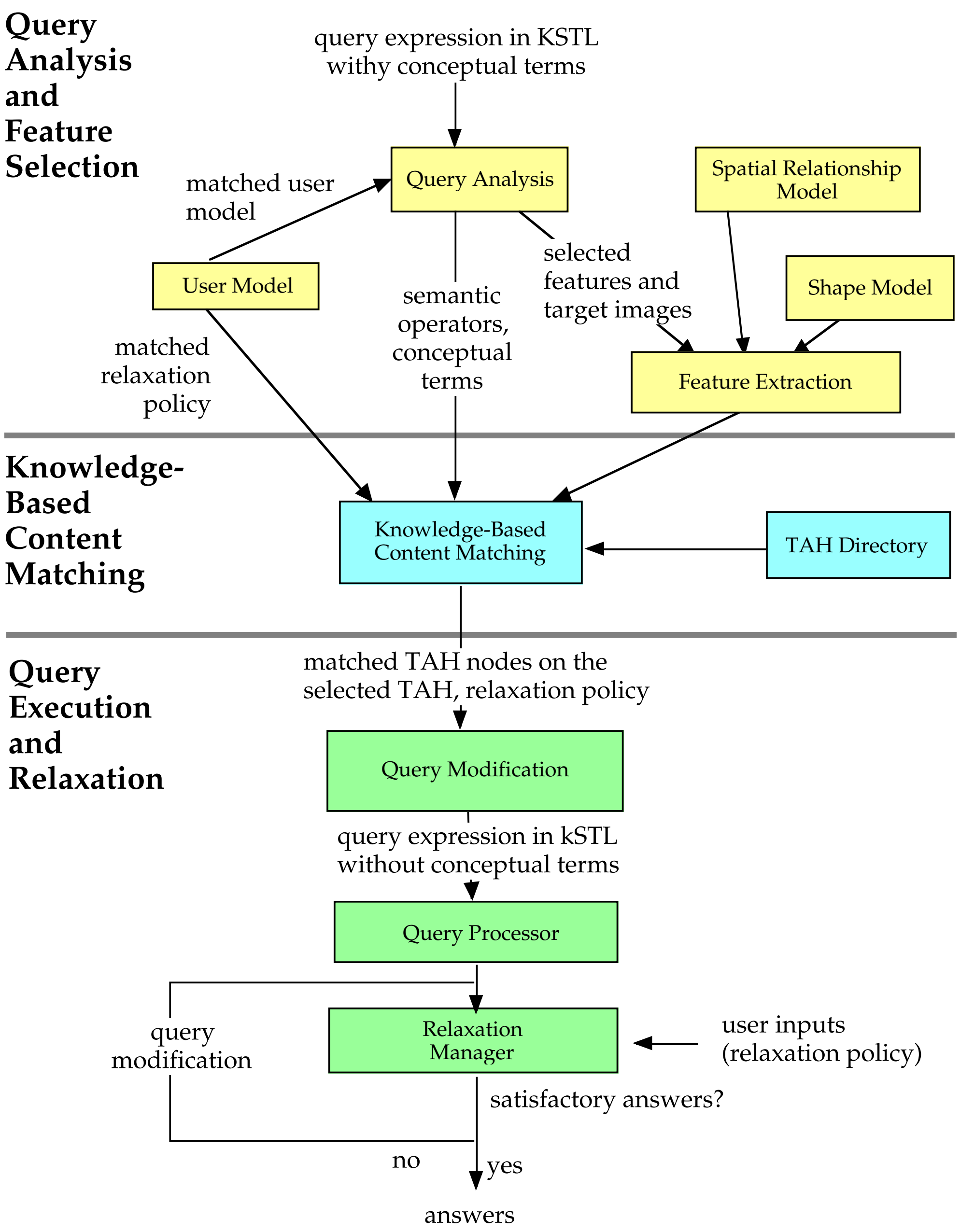

Figure 13: The flow diagram of knowledge-based query processing.

Query Analysis and Feature Selection Based on the target image, query context, and user class, the system analyzes and selects the relevant features and spatial relationships for knowledge-based content matching. A user model will be consulted to derive the matching and relaxation policy.

Knowledge-based Content Matching Spatial relationship and conceptual terms in the query are used to select the appropriate TAH from the TAH directory for query modification. For queries with *similar-to* operators, features and spatial relationships of the target images and user class are used for selecting the TAH(s). For queries with semantic spatial relationship operators and/or conceptual terms, the labels on the TAH node are used for matching the conceptual terms. The images under the TAH nodes are the images satisfied by the conceptual term (see Figure 3). After this phase, the knowledge required for resolving the conceptual terms (e.g. TAHs and matched user model) are attached with the query.

Query Execution and Relaxation Query conditions are modified based on the value ranges of the matched TAH nodes for semantic spatial relationship operators and conceptual terms. For the *similar-to* operator, the selected TAHs are traversed until a TAH node is reached whose value range is closest to the target image (see Figure 3). The value ranges in the parent TAH nodes of the matched TAH node can be used to replace the similar-to conditions. Thus, the query is transformed into a regular query (i.e. without containing conceptual terms) and can be sent to the commercially available OQL query processor to retrieve answers where the OQL query optimizer will optimize the queries since, after the query modification, the proposed OQL extention does not violate the OQL syntax and its declarative nature. This process of relaxation by query modification may be repeated until the set of returned answers match the user requirements. (e.g. number of similar images, relaxation error, etc. [CC94]). The returned images can be ranked based on a specific measure (e.g. their *relaxation error (RE)* with the target image) [CCHY96].

6 Data Flow and System Architecture

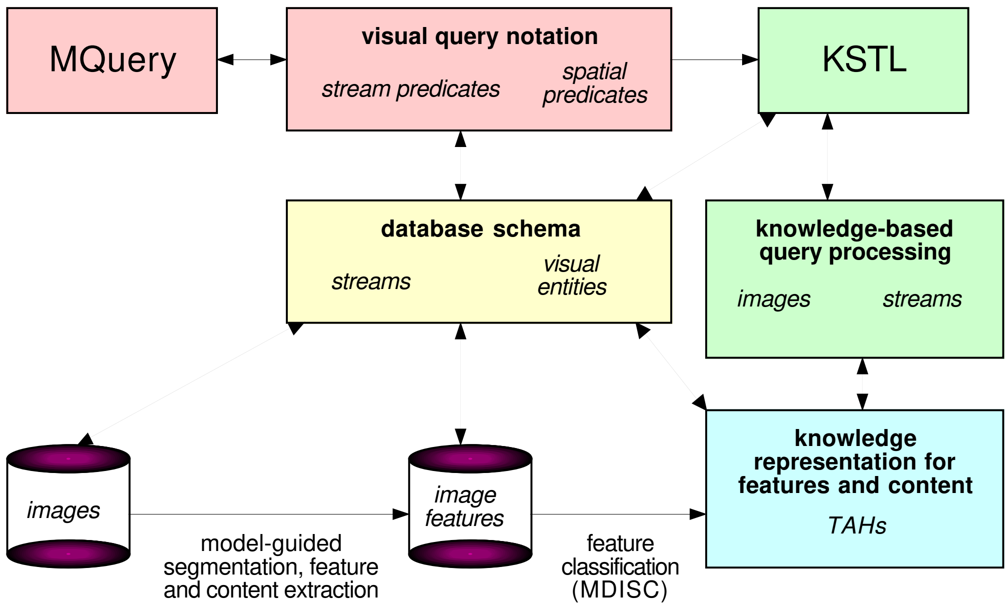

Figure 14: Data flow and system architecture.

Figure 14 illustrates the overall flow and key elements of our proposed system. Our raw data set will be images stored in the UCLA Picture Archiving and Communication System [HT92]. These images will be sent through segmentation routines [LTSK95] to generate contours of objects of interest in the images. Then, methods in the Feature and Content Layer extract image features from these contours and spatial relationships. These features are then mapped into visual entities and spatial relationships at the Schema Layer, for use by KSTL and our visual query language. In addition, the features are classified by the MDISC clustering algorithm to automatically produce the type abstraction hierarchies required for knowledge-based query answering.

At query time, the visual query language interprets a user's query input on the screen (physical positions and other pictorial information). Based on query notation and the database schema, the interpreter maps the user's input into appropriate query conditions. The translated query conditions and terms are used to formulate KSTL statements. The knowledge-based query processing parses the KSTL statement then uses type abstraction hierarchies for approximate matching of features and content to derive answers. The query result is returned back to the graphical user interface for presentation and visualization.

7 Comparison with Other On-Going Research

Pixel matching methods employed for content-based retrieval are time-consuming and have limited practical use since little of the image object semantics is explicitly modeled. QBIC [NB+93] uses global shape features such as area and circularity to retrieve similarly shaped objects. However, due to the limited precision of global shape features [Lev85], such an approach has limited expressiveness for answering queries with conceptual terms and predicates. Thus, we use a decomposition approach to describe the specific shape features of interested objects.

Statistical approaches can be used to retrieve similar images by relaxing a certain percentage of the standard deviation of the feature values (e.g., VIMS [BPJ93]), However, the same amount of relaxation is applied throughout the distribution, and thus is insensitive to the position of the operating point. Moreover, many image features are based on multiple attributes. Using standard deviation to retrieve similar feature values lacks the consideration of correlation among different attributes.

Retrieving image by features has been studied recently [CIT94, ACF+93, CCT95].

However, matching exact features is difficult. Therefore, it is essential to use a cooperative answering technique [CCHY96] to provide approximate matching of features. Previously proposed cooperative database systems used the rule based approach [CD89], and are therefore not scalable. In our approach, knowledge is structurally represented at different levels of abstraction by the type abstraction hierarchy and can be automatically generated from relational databases for both numerical and non-numerical data. Our knowledge acquisition algorithm uses the technique of unsupervised *conceptual clustering* [MS83, CCHY96]. In order to provide spatial relaxations of image features, we have extended this knowledge representation into spatial domains.

In previous cooperative systems, no relaxation control facilities were provided for the relaxation process nor the quality of the answer. Our system provides a cost model to optimize the relaxation and includes the handling of feature relaxation control for image data.

Current research in multimedia databases tackles individual components of the overall multimedia database problem. Models exist for representing image, sound, or video data [GBT93], and query languages for retrieving images by content have also been developed [NB$^+$93]. Modeling and query languages over conventional alphanumeric databases has also been progressing [MK93]. However, to support the multimedia medical database requirements of our system, we need to support combined alphanumeric and multimedia requirements. The visual query language, MQuery, supports knowledge-based temporal and spatial query answering. Other visual query languages [Cru92, WS93] can only support a subset of the requirements for the visual, multimedia query language provided by the MQuery language. Many existing works also model and query multimedia information [BVZ95, DDI95]. Similarly to [BVZ95], temporal logic is used as the formalism in our work to combine the visual language and its processing. Due to the fact that KSTL is extended from ODMG's OQL, our processing language is more directly compatible with commercially available database engine than those in [BVZ95, DDI95]. Also, predicates in MQuery (as well as KSTL) allow us to specify useful stream behaviors in querying medical image sequences (e.g., tumor size growing pattern).

Medical doctors continue to prefer highly-customized user interfaces that are very closely linked to specific application domains. However, limited usability testing was performed and when given enough training, doctors were able to use MQuery for simple database tasks. The only current operating alternative continues to be text-, mainframe-, and dumb terminal-based, thus making any direct comparisons somewhat awkward to make. Subjectively, however, the visual, highly-graphical approach was preferred over the above alternatives.

8 Conclusions

In this paper, we presented techniques for image retrieval by feature and content with temporal, spatial, and conceptual constructs. Innovative techniques include: 1) a four-layered integrated spatial and temporal data model that characterizes low-level image features (such as raw image data, and contours), abstract semantic image representations (including image objects and streams), and generic domain knowledge; 2) automatic feature analysis and classification for knowledge-based query answering; 3) development of a visual query language, *MQuery*, which is used to formulate queries with temporal, spatial, and conceptual image predicates; and 4) techniques for translating the visual query language into a textual query language, and Knowledge-based Spatial Temporal query Language (KSTL) which extends ODMG's OQL query language.

We have implemented a Knowledge-Based Medical Database System (KMeD) at UCLA, and the staff in the medical school is currently evaluating its functionality and effectiveness. The evaluation feedback will be useful in improving the system and providing further research directions.

9 Acknowledge

The development of KMeD is a team effort. We would like to thank the development and implementation effort of J. Dionisio for the M query language and the domain knowledge consultation from Dr. D. R. Aberle of the UCLA Radiological Science Department.

References

[ACF+93] M. Arya, W. Cody, C. Faloutsos, J. Richardson, and A. Toga. QBISM: A prototype 3-D medical image database system. *Bulletin of the Technical Committee on Data Engineering*, 16(1):38–42, March 1993.

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

[BPJ93] J. R. Bach, S. Paul, and R. Jain. A visual information management system for the interactive retrieval of faces. *IEEE Transaction on Knowledge and Data Engineering*, Oct 1993.

[BVZ95] A. D. Bimbo, E. Vicario, and D. Zingoni. Symbolic description and visual querying of image sequences using spatial-temporal logic. *IEEE Transactions on Knowledge and Data Engineering*, August 1995.

[Cat96] R. G. G. Cattell, editor. *The Object Database Standard: ODMG - 93 (Release 1.2).* Morgan Kaufmann, 1996.

[CC94] W. W. Chu and Q. Chen. A structured approach for cooperative query answering. *IEEE Transactions on Knowledge and Data Engineering*, 6(5), Oct 1994.

[CCHY96] W. W. Chu, K. Chiang, C. C. Hsu, and H. Yau. An error-based conceptual clustering method for providing approximate query answers. *Communications of ACM, Virtual Extension Edition (http://www.acm.org/cacm/extension)*, December 1996.

[CCT95] W. W. Chu, A. F. Cardenas, and R. K. Taira. KMeD: A knowledge-based multimedia medical distributed database system. *Information Systems*, 20(2):75–96, 1995.

[CD89] F. Cuppens and R. Demolombe. How to recognize interesting topics to provide cooperative answering. *Information Systems*, 14(2):163–173, 1989.

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

[CIT94] W. W. Chu, T. Ieong, and R. Taira. A semantic modeling approach for image retrieval by content. *The VLDB Journal*, 3(4):445–478, Oct. 1994.

[Cru92] Isabel F. Cruz. Doodle. In *Proceedings of the 1992 SIGMOD*, 1992.

[CYD88] S. K. Chang, C. W. Yan, and D. C. Dimitroff. An intelligent image database system. *IEEE Transaction on Software Engineering*, May 1988.

[DC96] J. D. N. Dionisio and A. F. Cardenas. Mquery: A visual query language for multimedia, timeline, and si mulation data. *Journal of Visual Languages and Computing*, 1996. Special issue on *Image and Video Databases: Visual Browsing, Querying, and Retrieval*. To appear.

[DC97] J. D. N. Dionisio and A. F. Cardenas. A unified data model for representing multimedia, timeline, and simulation data, 1997. To appear in *IEEE Transactions on Knowledge and Data Engineering.*

[DDI95] Y. F. Day, S. Dagtas, and M. Iino. Object-oriented conceptual modeling of videa data. In *IEEE International Conference on Data Engineering*, 1995.

[DV+93] D. Daneels, D. Van Campenhout, et al. Interactive outlineing: An improved approach using active contours. In *Image and Video Storage and Retrieval, SPIE*, 1993.

[Eme90] E. A. Emerson. *Handbook of Theoretical Computer Science*, volume Volume B, Formal Models and Semantics, chapter Temporal and Modal Logic. MIT Press/Elsevier, 1990.

[Fra88] Frank. PSQL: Pictorial spatial query language. *IEEE Transaction on Software Engineering*, May 1988.

[Gab89] D. Gabbay. The declarative past and imperative future temporal logic in specification: Altrincham workshop. In *LNCS 398*. Springer-Verlag, 1989.

[GBT93] S. Gibbs, C. Breiteneder, and D. Tsichritzis. Data modeling of time-based media. In D. Tsichritzis, editor, *Visual Objects*, pages 1–22. Centre Universitaire d'Informatique, Université de Genève, 1993.

[GM91] D. Gabbay and P. McBrien. Temporal logic and historical databases. In *Conference on Very Large Data Bases*, 1991.

[Gri90] W. Grimson. *Object Recognition by Computer: the role of Geometric Constraints.* MIT Press, MIT, MA, 1990.

[HCT96] C. C. Hsu, W. W. Chu, and R. K. Taira. A knowledge-based approach for retrieving images by content. IEEE Transaction on Knowledge and Data Engineering, August 1996.

[HT92] H. K. Huang and R. K. Taira. Infrastructure design of a picture archiving and communication system. *American Journal of Roentgenology*, 158:743–749, 1992.

[Lev85] M. D. Levine. *Vision in Man and Machine.* McGraw Hill Publishing Co., Inc., 1985.

[LG93] T. Little and A. Ghafoor. Interval-based conceptual models for time-dependent multimedia data. *IEEE Transactions on Knowledge and Data Engineering*, Aug. 1993.

[LTSK95] B. J. Liu, R. K. Taira, J. Shim, and P. Keaton. Automatic segmentation of bones from digital hand radiographs. In *Proceedings of the SPIE: Medical Imaging Image Processing*, volume 2434, pages 659–669, February 1995.

[MA79] W. N. Martin and J. K. Aggarwal. Computer analysis of dynamic scenes containing currilinear figures. *Pattern Recognition*, 11:169–178, 1979.

[MK93] L. Mohan and R. L. Kashyap. A visual query language for graphical interaction with schema-intensive databases. *IEEE Transactions on Knowledge and Data Engineering*, 5(5):843–858, October 1993.

[MS83] R. S. Michalski and R. E. Stepp. Learning from observation: Conceptual clustering. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning*, volume 1. Margan Kaufmann Publishers, Inc., 1983.

[NB+93] Wayne Niblack, R. Barber, et al. The qbic project: Querying images by content using color, texture, and shape. In *Storage and Retrieval for Images and Video Databases, SPIE*, 1993.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th Ann. IEEE Symp. on Foundations of Computer Science*, 1977.

[Ric92] J. Richardson. Supporting lists in a data model (a timely approach). In *ACM SIGMOD 1992*, 1992.

[SLVZ95] B. Subramanian, T. W. Leung, S. L. Vandenberg, and S. B. Zdonik. The aqua approach to querying lists and trees in object-oriented databases. In *IEEE International Conference on Data Engineering*, 1995.

[TCG+93] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors. *Temporal Databases.* The Benjamin/Cummings Publishing Company, Inc., 1993.

[WA+95] R. Wasserman, R. Acharya, et al. Multimodality tumor delineation via fuzzy fusion and deformable modelling. In *Proceedings of the SPIE, Medical Image: Image Processing*, 1995.

[WLK92] A. J. Worth, S. Lehar, and D. N. Kennedy. A recurrent cooperative/competitive field for segmentation of magnetic resonance brain images. *IEEE Transactions on Knowledge and Data Engineering*, 4(2):156–161, April 1992.

[WS93] Weiland and Shneiderman. A graphical query interface based on aggregation/generalization hierarchies. *Information Systems*, 1993.

A The Conversion of Temporal Logic Expression to KSTL Statements

In this appendix, we shall show how a temporal logic expression can be translated into a KSTL statement with a FROM and Where clause. Let us first define a few definitions.

A *pure snapshot predicate* is a snapshot predicate that does not contain any modal operators. A *simple temporal logic predicate* is a pure snapshot predicate with no more than one modal operator $\Diamond$ or $\Box$. Let the temporal logic expression be $P = p_1 \ldots p_n$ where $p_i, 1 \leq i \leq n$ are simple temporal logic expressions. Let S be all the streams available for selection. During the conversion, the temporal logic expression P is translated into a KSTL statement with a FROM and WHERE clause. Since the temporal logic expression selects streams from S, the initial FROM clause is "FROM S as s".

All individual p_i, which expresses a temporal constraint on a snapshot or a substream, can be translated into KSTL predicates in the WHERE clause as follows:

case 1: If p_i is a pure snapshot predicate or a snapshot predicate with a modal operator $\Diamond$, then the statement "x_i in s" is added to the FROM clause to bind a snapshot in s to x_i. The statement "$p_i(x)$" is added to the WHERE clause to express the constraint on the candidate snapshot. "$p_i(x)$" is the corresponding snapshot predicate of p_i expressed in KSTL.

case 2: If p_i is a snapshot predicate with a modal operator $\Box$, then the statement "$AllSubStream(s)$ as z_i" is added to the FROM clause to bind a candidate substream to z_i. The statement "`forall` y_i in z_i:$p_i(y)$" is added to the WHERE clause to express the constraint on the candidate substream. "$p_i(y)$" is the corresponding snapshot predicate of p_i expressed in KSTL.

The temporal relationship between the neighboring simple temporal logic expressions p_i and p_{i+1} $(1 \leq i \leq n-1)$ should now be added to the WHERE clause. For any two neighboring simple temporal logic expressions $p_i p_{i+1} (1 \leq i \leq n-1)$, constraints on their implied temporal relationship are added to the WHERE clause as follows:

case 1: When p_i is a pure snapshot predicate or a snapshot predicate with a $\Diamond$ modal operator and p_{i+1} is a pure snapshot predicate, the snapshots satisfying p_i and p_{i+1} are immediate neighbors in the stream. Thus, the statement "$x_i.next == x_{i+1}$" is added to the WHERE clause to specify their temporal relationship.

case 2: When p_i is a pure snapshot predicate or a snapshot predicate with a $\Diamond$ modal operator and p_{i+1} is a snapshot predicate with a $\Diamond$ modal operator, the snapshot satisfying p_i appears before the snapshot satisfying p_{i+1} in the stream. The statement "$x_i.before(x_{i+1})$" is added to the WHERE clause to specify their temporal relationship.

case 3: When p_i is a pure snapshot predicate or a snapshot predicate with a $\Diamond$ modal operator and p_{i+1} is snapshot predicate with a $\Box$ modal operator, the snapshot satisfying p_i and the first snapshot of the substream satisfying p_{i+1} are immediate neighbors in the stream. The statement "$x_i.next == first(z_{i+1})$" is added to the WHERE clause to specify their temporal relationship.

case 4: When p_i is a snapshot predicate with a $\Box$ modal operator and p_{i+1} is a pure snapshot predicate, the last snapshot of the substream satisfying p_i and the snapshot satisfying p_{i+1} are immediate neighbors in the stream. The statement "$last(z_i).next == x_{i+1}$" is added to the WHERE clause to specify their temporal relationship.

case 5: When p_i is a snapshot predicate with a $\Box$ modal operator and p_{i+1} is a snapshot predicate with a $\Diamond$ modal operator, the last snapshot of the substream satisfying p_i occurs before the snapshot satisfying p_{i+1} in the stream. The statement "$last(z_i).before(x_{i+1})$" is added to the WHERE clause to specify their temporal relationship.

case 6: When p_i is a snapshot predicate with a $\Box$ modal operator and p_{i+1} is a snapshot predicate with a $\Box$ modal operator, the last snapshot of the substream satisfying p_i and the first snapshot of the substream satisfying p_{i+1} are immediate neighbors in the stream. The statement "$last(z_i).next == first((z_{i+1})$" is added to the WHERE clause to specify their temporal relationship.

Note that p_1 and p_n express the boundary conditions that a selected stream should satisfy. These boundary conditions should also be translated into KSTL as follows:

case 1: When p_1 is a snapshot predicate with a $\Box$ modal operator, the statement "$first(z_1) == first(s)$" is added to the WHERE clause to specify the beginning condition of the stream s.

case 2: When p_1 is a pure snapshot predicate, the statement "$x_1 == first(s)$" is added to the WHERE clause to specify the beginning condition of the stream s.

case 3: When p_n is a pure snapshot predicate or snapshot predicate with a $\diamond$ modal operator, the statement "$x_n == last(s)$" is added to the WHERE clause to specify the ending condition of the stream s.

case 4: When p_n is a snapshot predicate with a $\diamond$ modal operator, the statement "$last(y_n) == last(s)$" is added to the WHERE clause to specify the ending condition of the stream s.

Finally, relationships between the neighboring pairs of x_i's (e.g., INTERVAL[x_2,x_1].approximate(T) where T is an interval length specified by users) are added to the WHERE clause. Note that these relationships cannot be expressed inside a temporal logic expression due to unclear variable binding in a temporal logic expression.