1

A VISUAL MULTIMEDIA QUERY LANGUAGE FOR TEMPORAL ANALYSIS OF VIDEO DATA

ABSTRACT

The storage of various media in multimedia databases poses new challenges to query techniques — challenges that exceed the expressive power of traditional text-based query languages. New query interfaces should take advantage of characteristics *inherent* in multimedia data, such as the dynamic temporal nature of video, the visual and spatial characteristics of images, the pitch of audio, etc. The focus of our research is to exploit the temporal *continuity* and combined *spatio-temporal* characteristics of video data for the purpose of video analysis. We do so by integrating a visual query paradigm with a dynamic visual presentation of results into a user-friendly interactive visualization environment. In this chapter, we present our overall approach for identifying trends in video data via querying for *relationships* between video annotations. Our approach allows users to analyze the video in terms of *temporal* relationships between events (e.g., events of type A frequently follow events of type B). We present a temporal visual query language (TVQL) for specifying relative temporal queries between sets of annotations. This query language builds on the notion of dynamic query filters and significantly extends them. It is tailored for temporal analysis — allowing users to pose queries, as well as to *browse* the data in a *temporally continuous* manner, thereby aiding them in the discovery of temporal trends. The TVQL is augmented with complementary temporal diagrams, which provide intuitive visual feedback for quickly and qualitatively verifying the temporal query specified. This chapter includes the complete specification of our TVQL, a transformation function for generating corresponding temporal diagrams, and the process for mapping TVQL queries to system queries.

[by Stacie Hibino and Elke A. Rundensteiner, University of Michigan, (hibino, rundenst)@eecs.umich.edu.

To appear in an upcoming Multimedia DBMS book, edited by Kingsley Nwosu.]

1 INTRODUCTION

1.1 Motivation

Multimedia databases are becoming more and more common, allowing users to store and retrieve various media for different purposes. Medical researchers are storing x-rays in medical image databases, art historians are archiving and documenting artwork, geographers are creating geographical information systems, and digital libraries are becoming more popular. Video data is also becoming more common, being collected by educational researchers for classroom studies, by software evaluators during usability testing, by scientists for evaluating the behavior of microscopic entities, etc. An advantage of video data is that it preserves detailed information which would be difficult to capture by other means of data collection. The challenge of analyzing video data is to abstract and conclude quantitative results from such a rich, qualitative medium.

The ability to store various media thus poses new challenges to query techniques, moving beyond traditional text-based query languages. This is due to the fact that media such as audio, images and video have basic characteristics which are different from text. For example, video has temporal and visual characteristics, audio has basic characteristics such as pitch and timbre, images have features of hue and saturation, etc. Query interfaces to multimedia databases should thus take advantage of such inherent characteristics in order to properly allow the exploitation of the richness of information captured in the media.

Recent work in image and video databases illustrate the use of query techniques which exploit inherent visual and spatial characteristics [20, 26]. For example, Ueda, et al [26] have developed techniques to find images based on the user's selection of an area in an existing image. Their technique works for simple queries equivalent to "find all images with blue sky" to more sophisticated queries such as object extraction, including the location of video segments containing a specific person. The same techniques for locating segments containing complex objects such as people can also be used to determine situations such as whether a person is standing in a doorway. While these techniques do take advantage of the inherent characteristics of the media, they typically focus more on "low-level" bit analysis and typically have high computation costs. In order to support a higher level of analysis — and video analysis in particular, we need to move up to an *object* level of abstraction. In this way, once we can characterize media according to the objects they contain, we can then more efficiently query the media database based on inherent *relationships* between these objects. In the case of video analysis, we can then examine temporal and/or spatial relationships between objects and events.

While some work in video databases and analysis[1] [19, 21, 27, 13] has used the temporal characteristics of video, little work in temporal query techniques of dynamic media has been done to take advantage of the temporal *continuity* and/or the combined *spatio-temporal* characteristics of the medium. Even when users may only be interested in temporal analysis, the use of spatial context may provide visual clues aiding in the analysis process. As video data becomes more and more a common form of data collection, such techniques will become a more critical need for analysis. Our work is designed to address issues related to these needs.

1.2 Introduction to Our Approach

In order to support temporal analysis of video data, using spatio-temporal characteristics and exploiting the temporal continuity of the medium, we have identified three primary needs to address: 1) the need to abstract and store spatio-temporal (and semantic) information from the video data directly, 2) the need to query for temporal relationships within the data in a continuous manner, and 3) the need to present the results in a spatio-temporal visualization.

Similar to previous work in video analysis [13, 19, 22], we are also using annotations for abstracting information and coding the video data. The advantages of using annotations are that 1) they allow users to abstract both *temporal and spatial* information from the data, 2) they simplify analysis by reducing the amount of information to be processed, and 3) when layered on top of the original data, they allow users to preserve context without corrupting the original data. In contrast to previous work, however, we are not using annotations to *pre-code* relationships, but we restrict them to only abstracting information about *atomic* objects and events. Coding only the atomic content requires less storage space and less time than coding every relationship between each atom, especially when users are interested in several different types of relationships. In addition, it provides more flexibility in the analysis process, aiding users in *discovering relationships* rather than requiring them to predefine or pre-code all possible relationships. Our annotation model is described in Section 2.

[1] As suggested in the previous paragraph, we use the term *video analysis* to refer to the process of identifying trends and relationships between events in the video data. This is in contrast to bit-level video analysis such as that used for object extraction.

Once we have stored video annotations in a database, we can then query the annotation collection in search of data trends. In order to support the trend searching process and exploit the temporal continuity and combined spatio-temporal characteristics of the underlying medium, we are designing a user-friendly interactive visualization environment. This environment integrates a visual query paradigm with a dynamic visual presentation of results. The visualization is *dynamically* updated as users incrementally adjust query parameters via direct manipulation of buttons and sliders. This chapter describes our overall environment (Section 3) and then presents the details of our *temporal* visual query language (TVQL — see Section 5). Our TVQL can be used to specify *relative* temporal queries between sets of annotations. This query language builds on the notion of dynamic query filters [2] and significantly extends them. It is tailored for temporal analysis — allowing users to pose queries, as well as to *browse* the data in a *temporally continuous* manner, thereby aiding them in the discovery of temporal trends. The TVQL is augmented with complementary temporal diagrams, which provide intuitive visual feedback for quickly and qualitatively verifying the temporal query specified (Section 6).

1.3 Sample Scenario

Consider the case where educational researchers collect and analyze classroom video data to study social interactions in the classroom. The researchers could use annotations to code (i.e., describe and interpret) the video data, abstracting information such as when the teacher is speaking, when the student is speaking, etc. That is, they can create annotations such as text labels to identify objects and people, circles to highlight interesting situations, etc. Once the annotations have been stored in a database, the researchers can begin exploring the relationships between different types of classroom events. For example, they can create one subset of "student speaking" events and one subset of "teacher speaking" events. They could then search for temporal trends in the video data by specifying queries to locate occurrences of, and relationships between subsets of annotations. In this way, the researchers can use our TVQL to determine temporal relationships such as which students frequently speak after the teacher speaks. They might first search for temporal relationships where a student starts talking at least ten seconds *after* the teacher stops speaking. By incrementally adjusting this query to locate those students who begin speaking *before* the teacher finishes, researchers can identify situations where a student might be interrupting the teacher or talking out of turn. This ability to incrementally adjust queries allows users to temporally browse the data in a new and powerful way.

1.4 Relationship to Previous Work

Preliminary descriptions of this work have been presented elsewhere [14, 16]. This chapter expands this previous work, providing a more refined and complete description and derivation of our temporal visual query language (TVQL). More specifically, in this chapter, we present the full specification of the TVQL, including the derivation of the temporal duration component and more technical details regarding the underlying relationships between the four relative temporal position filters. In addition, we also include a complete description of the annotation model, as well as an outline of the process for mapping TVQL queries to system queries.

1.5 Overview

This chapter is organized into nine additional sections. Section 2 describes our annotation model. Section 3 presents an overview of our interactive visualization approach and Section 4 describes how we specify *relative* queries within our framework. Section 5 describes our temporal visual query language for specifying relative temporal queries and Section 6 presents our visual enhancements to our TVQL, including the derivation of our complementary temporal diagrams. Section 7 describes the process of mapping TVQL queries to system queries and Section 8 presents some preliminary evaluation of the query language. Section 9 discusses related work and finally, Section 10 presents our current status along with a summary of contributions.

2 ANNOTATION MODEL

Section 1.2 introduced the use of annotations for abstracting spatio-temporal and semantic information from video data. The use of annotations simplifies the analysis process by allowing us to evaluate and manipulate the data at an object and event level rather than at the bit-level. Recall that we restrict annotations to coding *atomic* objects and events rather than for *pre-coding* relationships within the video data. In this way, users can *explore* the data *in search of* relationships rather than predefining a subset of relationships to evaluate. This section provides an overview of the annotation model we assume by defining the terms video, video segment, event, and video annotation as they are used in the context of this chapter.

Video. In this chapter, a *video* object v has the following temporal characteristics: a start time, $v.t.start\_time$ (defined as time 0); an end time, $v.t.end\_time$, defining the finite (maximum) length of the video in seconds (up to two decimal points); a start frame, $v.t.start\_frame$, indicating the physical video frame number corresponding to the start time of the video; a corresponding end frame $v.t.end\_frame$. We assume for simplicity that the frames are numbered consecutively, starting from 0. We define a function $VF(Vtime)$ which maps a given video time (in seconds) to the corresponding frame: $VF(Vtime) = round((Vtime * fps) + f\_offset)$, where fps = the number of frames per second of the video and f_offset = the frame number corresponding to time 0 of the video. Similarly, we define a function $VT(frameNum) \longrightarrow time$: $VT(frameNum) = (frameNum - f\_offset)/fps$.

A video object v also has some spatial characteristics including position ($v.sp.x$, $v.sp.y$), width ($v.sp.width$), and height ($v.sp.height$) of the video in screen pixel units. We assume that a video object is one *continuous* medium, including the start and end frames as well as *every* frame between them. In other words, v is a complex object decomposed of an ordered sequence F of frame objects, $F = [f_0, f_1, \ldots, f_z]$.[2]

Video segment. A *video segment vs* is a continuous subset of frames of the video object (i.e., vs is defined over frames $[f_i, \ldots, f_j]$, where $0 \leq i \leq j \leq z$). Similar to the video object v, a video segment vs of v has a start frame ($vs.start\_frame \geq v.start\_frame$) and an end frame ($vs.end\_frame \leq v.end\_frame$) as well as starting and ending times. In addition, the length of a video segment must be less than or equal to that of the video (i.e., $vs.end\_frame - vs.start\_frame \leq v.end\_frame - v.start\_frame$).

Video annotations. A *video annotation va* is used to indicate real-world situations within the video, also referred to as *events*. It is visually represented by a media object (e.g., text, audio, graphic, etc.) that is temporally linked to a video segment and spatially linked to a position on top of the video. That is, the video annotation specifies spatial information for displaying the media object relative to the video window, so that $v.sp.x \leq va.sp.x \leq v.sp.x + v.sp.width$ and $v.sp.y \leq va.sp.y \leq v.sp.y + v.sp.height$. The video annotation references the media object through a reference to a content-based descriptive object. In addition to spatial, temporal and content-based characteristics, each annotation also has space for user-specified comments.

[2] We assume that we are dealing with uncompressed video, so that (unlike MPEG files) every frame is complete and independent of its predecessors. As a consequence each frame is directly accessed without the need for additional calculations.

More formally, for a given video v_p, we denote the set of annotations $Ann(v_p) = \{va_1, va_2, \ldots, va_n\}$. Each annotation va_i (where $1 \leq i \leq n$) has four components: temporal information $va_i.t$, spatial information $va_i.sp$, a reference to a descriptive object $va_i.d$ and space for users' comments $va_i.comments$ (see Figure 1).

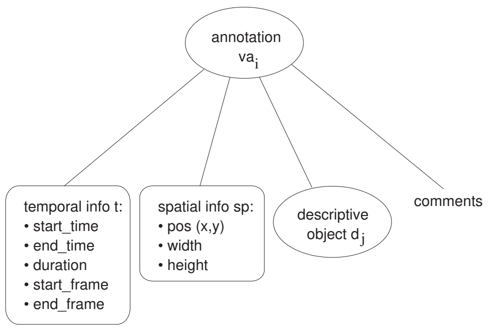


Figure 1 Characteristics of a video annotation object.

The reference from an annotation va_i to a descriptive object d_j is denoted by $description(va_i) = d_j$. Given $D = \{d_1, d_2, \ldots, d_m\}$ as the set of all descriptive objects, then each descriptive object d_j (where $1 \leq j \leq m$) has four components: content information $d_j.content$, history information $d_j.history$, default spatial information $d_j.sp$, and a reference to a media object $d_j.md$. The characteristics of the descriptive object are presented in Figure 2.

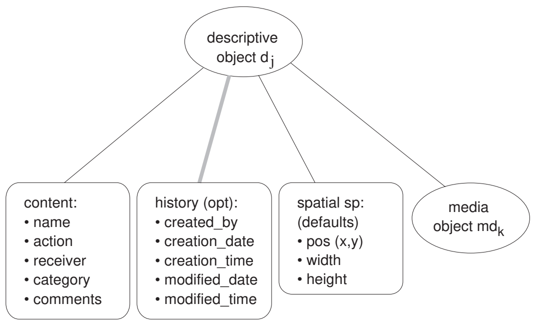


Figure 2 Characteristics of a descriptive object.

The descriptive object *content* is used to store semantic information about the event being annotated. *Content.name* is the name of the object or person that d_j represents. *Content.action* is the action being taken (e.g., talking) and *content.receiver* is the person or object being acted on. The content action and receiver are optional while the name is required. *Content.category* allows users to group content into their own categories. For example, the researcher can assign all individual "student-working" and "student-talking" (e.g., Jane talking, John working, etc) descriptions into a more general category of "student activity." This provides users with a natural mechanism for grouping coded data and should also simplify the selection of annotation subsets. The *content.comments* provides a free form space for optional user comments. The d_j*.content.comments* associated with the descriptive objects are different from the more specific va_i*.comments*.

The descriptive object history information about when an annotation was created or last modified is optional (as denoted by the shaded arc). The spatial information provides default spatial values to relieve users from having to re-specify them every time the same descriptive object is used. Each descriptive object contains a reference to a media object. A media object is the visual or auditory representation of the video annotation (e.g., a circle, arrow, talk bubble icon, etc.). Several descriptive objects can share the same media object. In this way, the users can consistently assign, for example, all "student-talking" descriptive objects to a black talk bubble icon and all "teacher-talking" descriptive objects to a red talk bubble icon. If we let $M = \{md_1, md_2, \ldots, md_p\}$ be a set of media objects, then each media object md_k (where $1 \leq k \leq p$) has the characteristics depicted in Figure 3.

Users can choose one of three kinds of media: audio, 1-D, or 2-D. (In our diagrammatic representation, connected arcs represent alternatives.) The 1-D and 2-D media types provide additional media choices. Once a media object has been created, it is stored on a palette so as to be easily accessible for re-use. This will aid users in coding the video data in a consistent manner.

Although we are currently requiring researchers to input annotations manually, we expect that previous work by others in the area of object extraction (e.g., [20]) will eventually be used to generate annotations automatically. We also expect, however, that it will be too inefficient to operate on the raw data directly. Hence, even when image analysis and object recognition algorithms have matured, it is likely that they will still be applied *upfront*, to interpret video and generate annotations much like those presented in this section, rather than during query processing.

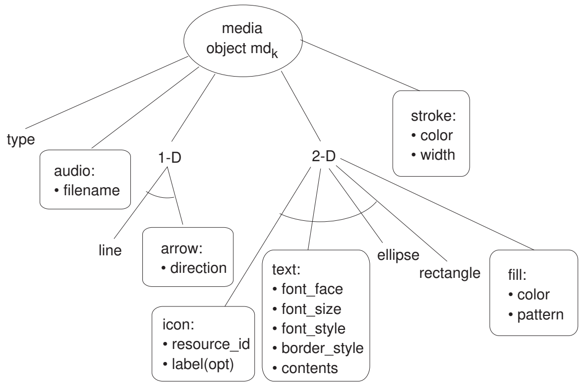


Figure 3 Characteristics of a media object.

3 OVERVIEW OF OUR APPROACH

Our overall goal is to support users in analyzing video data by providing tools for querying video annotations in search of relationships and data trends. We postulate that visually presenting the results retrieved from relative queries will facilitate this searching for trends. The query and review part of the video analysis process is similar to the process used in visual information seeking (VIS) [3]. The VIS process is a process for *browsing* database information; a process characterized by rapid filtering, progressive refinement, continuous reformulation of goals, and visual scanning to identify results. These characteristics make VIS particularly well suited for searching for trends in the video annotations. We are thus adopting this methodology as our underlying framework.

Figure 4 presents our overall approach to applying and extending VIS to the problem of video data analysis [15]. In our MultiMedia VIS (MMVIS), users first code the video data with annotations. These annotations are stored in an underlying database (also known as the Annotation Server). Users can then explore and analyze the video through iteratively specifying queries using a visual query language (VQL) and reviewing the visualization of results presented. Similar to VIS, our interface will also use *dynamic query filters* [2]. Our filters, however, will be customized to handle relative *temporal* queries. The advantage of query filters is that they allow rapid incremental adjustment of query parameters via the use of buttons and sliders. This is in contrast to text-based query languages, where query specification and modification are typically much

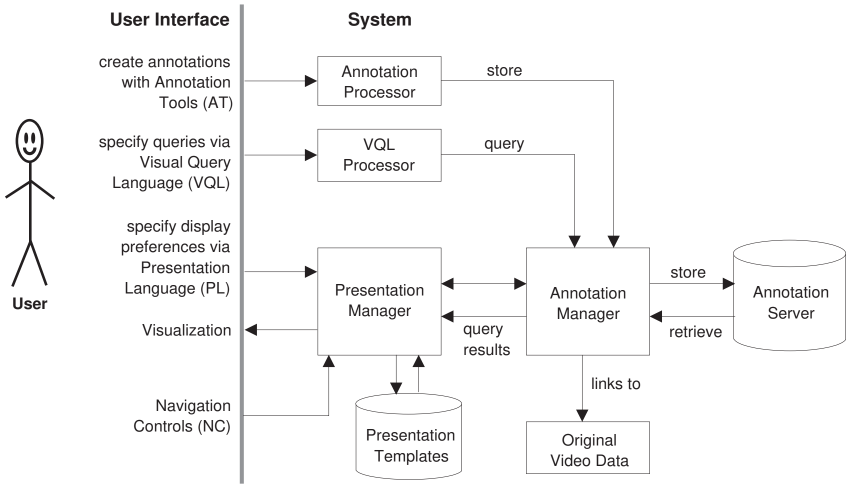


Figure 4 Framework for MultiMedia Visual Information Seeking (MMVIS).

more complicated and less intuitive. Our temporal extensions represent a novel application of the dynamic query filter technology.

The queries are interpreted by a VQL processor and then forwarded to the Annotation Manager. The retrieved results are passed to a Presentation Manager. The Presentation Manager takes the query results, along with any user-defined display preferences and updates a visualization. In this way, the visualization is updated every time users make changes to any query filter. Users can visually scan the results to look for data trends. If no trends are found, they can use the presentation language (PL) to clarify the visualization, the navigation controls to further explore query results, or the VQL to incrementally adjust the query. In addition, if users wish to test a new hypothesis or explore different characteristics of the data, they can use the VQL to do so. Thus, queries are expressed *incrementally* as users specify desired values for each query parameter. In such an environment, users can gain a sense of causality between adjusting a query filter and the corresponding changes presented in both the other query filters and the visualization.

Based on the long-term objectives outlined above, we now have tackled a subset of the issues required to build the MMVIS environment. The goals of this chapter are (1) to define a general temporal visual query language (TVQL) that can be easily integrated into a variety of applications and (2) to design a TVQL interface that is easy and intuitive to use by application (i.e., video annotation and analysis) users who typically are query language novices. Thus, the specific problem addressed in this chapter is that of the users' need for a simple interface for specifying relative temporal queries to video data. *Relative* queries are necessary for examining *relationships* between events and thus for identifying trends. A *visual* language is desired to correspond with the graphical nature of the objects in the database and is much more suitable for the type of novice users we are targeting. Finally, a *temporal* language is required to facilitate searching for temporal data trends. While we plan to explore both spatial and temporal aspects of video analysis in the future, we have focused our initial work on only the temporal aspect.

4 SPECIFYING RELATIVE QUERIES

Before we can describe our *temporal* extensions to VIS, we must first describe how to extend VIS to handle *relative* queries. While we can use the VIS approach to identify and specify interesting *subsets* of data, we cannot use the approach to specify *relationships* between these subsets [3]. In order to better understand this problem, consider a query such as "show me all annotations where the student is working *while* the teacher is speaking." While we can use dynamic queries (DQs) to select subsets such as the "student working" and "teacher speaking" annotations, we cannot use DQs to specify a desired relationship between members of these subsets. In the case of the example, we need to be able to specify a relative condition that holds true for each (teacher-speaking, student-working) pair returned. Rather than specifying a range of *absolute* conditions, we need to specify *relative* ones.

Extending the use of DQs to handle *relative* temporal queries of video data requires binding subsets of the data to variables and specifying temporal relationships between these subsets. Two subsets can be specified through the use of two sets of standard query filters. We thus provide a query palette for specifying each subset. Parameters specified in the Subset A query palette automatically bind the subset formed to the variable A. The corresponding functionality is provided for binding the second subset to variable B.

The simple example in Figure 5 illustrates how we can use query filters in the Subset A palette to bind the subset "teacher talking" to variable A, and those in the Subset B query palette to bind the "student working" subset to variable B. The actual subset query palettes allow the user to set more parameters and parameter values (e.g., to specify annotation media types, secondary actions, etc.). The sliders at the bottom of each subset palette allow users to specify *absolute* temporal information for each subset. This is in contrast to the temporal relationship R *between* sets A and B that will be specified with *specialized temporal query filters* (see Section 5). Progressive refinement of queries will be preserved by allowing the adjustment of query filters for A, B, or R at any time and in any order.

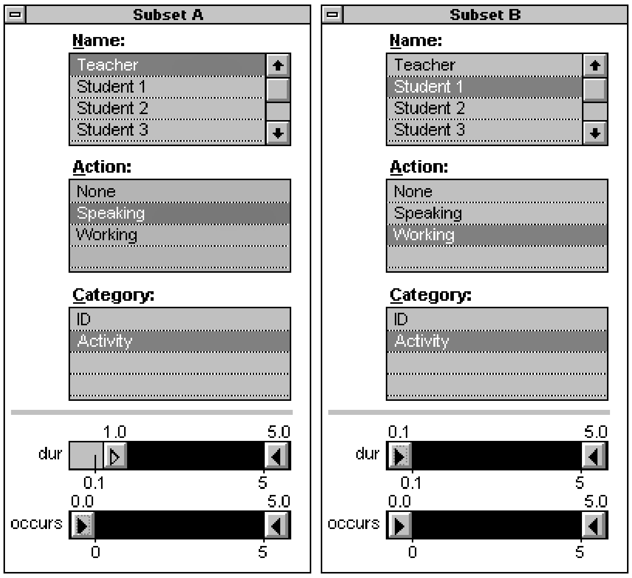


Figure 5 Simple example of potential subset query filters. The Subset A palette binds the subset "teacher talking" to variable A, and the Subset B query palette binds the "student working" subset to variable B.

Note that while value ranges for discrete predicate domains such as "Name" and "Category" are selected from lists, numerically valued ranges such as "Dur" (for duration) are selected using double-thumbed slider filters. This type of filter was introduced in [3] for specifying ranges of values. Each thumb has an arrow, pointing towards the range being specified. The thumb arrow is filled to indicate that the endpoint of a range is included, or empty to indicate that the endpoint of a range is excluded. Ranges are filled in the sliders for further clarification. The text above the slider indicates exact values. Figure 6 identifies each of these components of the double-thumbed slider.

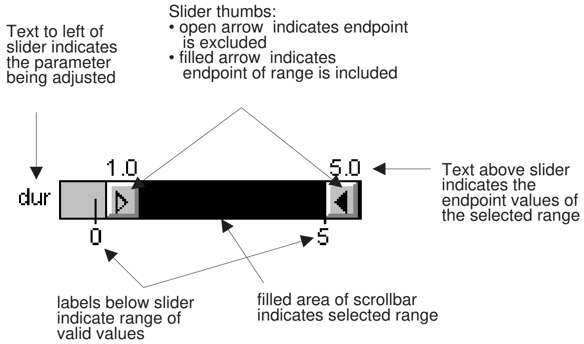


Figure 6 Description of a *double-thumbed slider* query filter.

In addition to double-thumbed sliders, Ahlberg and Shneiderman [3] introduce a number of other interfaces for different types of query filters. For example, they use a single-thumbed slider (with no arrow) for specifying a single value, or a single-thumbed slider with an arrow to indicate a range towards one of the extremes of the slider. Rather than requiring the user to change the type of slider filter for different types of ranges, we are adopting a single type of slider filter (i.e., the one illustrated in Figure 6) to handle all range query specifications. It is as powerful as the specialized type of sliders, and the uniformity should simplify the user interface.

Based on the description of the double-thumbed sliders, we now see that in the example in Figure 5, subset A includes all "teacher talking" annotations that have duration greater than 1.0 minute and less than or equal than 5 minutes. Similarly, subset B includes all "student working" annotations with duration that is greater than or equal 0.1 and less than or equal to 5.0 minutes. In both subsets, there is no restriction on when the annotations temporally occur within the 20 minute video. Again, these *absolute* temporal constraints specified in the subset palettes are different from the *relative* temporal constraints which will be specified using the temporal query palette. That is, while we can use the absolute sliders to restrict analysis to the first five minutes of the video (by setting the right thumb of the occurs filter to 5.0), we cannot use these absolute sliders to specify *relative* temporal constraints such as limiting the results to situations where A events end up to one minute before B events start. In the next section, we present our specialized relative temporal query filters which are designed to address these needs.

5 THE TVQL

This section presents a temporal visual query language (TVQL) for specifying relative temporal queries. Our TVQL is composed of temporal query filters, a disjunctive operator, and a macro operator. Our goal here is to gracefully integrate these temporal filters and operators with the standard DQ and VIS properties introduced in Section 3.

We assume that *absolute* temporal duration and position are specified with the corresponding subsets A and B (see Figure 5). *Relative* temporal duration and position can then be specified using our TVQL. Query filters for *relative* temporal duration and position are presented in Sections 5.1 and 5.2. The disjunctive operator is introduced in Section 5.2. User-created macros (i.e., the macro operator) included for improving efficiency of query specification over time are presented in Section 5.3.

5.1 The Relative Duration Query Filter

Two events A1 and B1 can be temporally related in terms of their durations. Although relative duration may sometimes be dictated by relative temporal position (e.g., if event A1 "contains" event B1, then the duration of event A1 is longer than the duration of event B1), this is not the case for all temporal position relationships (see Section 5.2). In addition, users may wish to specify relative duration queries *independent* of relative temporal position. For example, users may wish to locate all events where student S1 works on a task for a longer period of time than student S2. In this particular case, we do not care whether these two events are concurrent, partially overlapping, or positionally unrelated. To address this need, we propose a query filter for relative duration.

Given the durations of events A1 and B1 ($A1.t.duration$ and $B1.t.duration$, respectively), the basic relationship for relative duration between the events can be described as follows:

$$A1.t.duration \theta B1.t.duration$$

where $\theta \in \{ <, >, = \}$. While this relationship can be used to *qualitatively* describe the relative duration between two events, it cannot be used to specify *quantitative* information about that relative duration. For example, if we are interested in cases where event A1 has a longer duration than event B1, then we can specify the constraint $A1.t.duration > B1.t.duration$. This does not, however, specify the quantity by which A1's duration exceeds that of B1. We can address this problem by redefining the relationship in terms of differences:

$$(A1.t.duration - B1.t.duration)\theta 0$$

where $\theta \in \{ <, >, = \}$ and 0 denotes zero. If we let $\Delta dur = A1.t.duration - B1.t.duration$, then we can reduce the above specification to the following simpler expression:

$$\Delta dur \theta 0,$$

where $\theta \in \{ <, >, = \}$. Now if we are interested in cases where the duration of A1 is one or more minutes longer than the duration of B1 (e.g., student S1 spends at least one minute more on a task than student S2), then we can set Δdur > 0 and more specifically, we can set $\Delta dur \geq 1$ minute. This is in contrast to the first representation, where we would only be able to say $A1.t.duration > B1.t.duration$.

In addition to allowing us to quantify the relative duration, the Δ representation facilitates specification of values from a continuous range. This allows us to use a slider query filter for specifying relative duration (see Figure 7). Our relative duration specification is now compatible with the DQ interface we are designing for video analysis. Rather than requiring users to translate their queries into notations (e.g., Δdur) and algebraic equations, we provide full labels to the left of the filters and qualitative English text above the filters. In this way, users can "read" the query they have specified. In Figure 7, the query filter can be read as "the dur of A is more than one minute greater than the dur of B." These descriptive enhancements are described in more detail in Section 6.2.

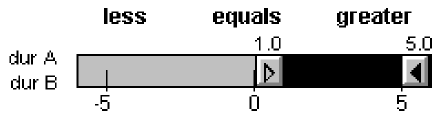
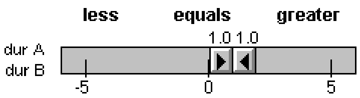
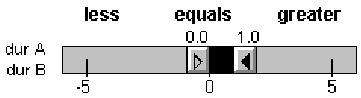
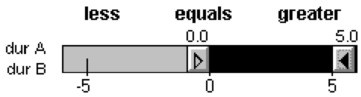


Figure 7 Query filter for specifying relative duration. Example: Student S1 works one or more minutes longer on tasks than student S2. Given a subset A bound to "student S1 working" and a subset B bound to "student S2 working," we can specify the query "Show all situations where student S1 works more than one minute longer than student S2" by setting the left thumb of the query filter to > 1 (i.e., by sliding the left thumb to position 1.0 and making sure the left thumb arrow is unfilled to exclude 1.0 in the selected range).

The type of range selection supported by DQ filters allows us to specify values at different levels of granularity (e.g., to specify a single value, a small range of values, or a larger range of values). Table 1 provides examples to illustrate this advantage for the relative duration query filter. The first example in Table 1 illustrates how to query for all (A_i,B_j) pairs such that A_i is exactly one minute longer than B_j; the second example shows how to query for all pairs such that A_i is at most one minute longer than B_j, and the final example shows how to find all pairs such that A_i is longer than B_j. While these types of queries can also be submitted using a text-based query language, the DQ-based interface allows users to easily and incrementally adjust their queries and thus easily move from a coarse-grained level of analysis to a fine-grained one and vice versa.

Table 1 Examples for using the relative temporal duration query filter to specify queries at various levels of granularity.

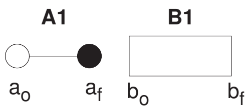
Example: (Find all (Ai,Bj) pairs such that:)	**Query Filter**
Ai is one minute longer than Bj: Ai.t.duration - Bj.t.duration = 1.0	
Ai is at most one minute longer than Bj: Ai.t.duration - Bj.t.duration > 0 ^ Ai.t.duration - Bj.t.duration <= 1.0	
Ai is longer than Bj: Ai.t.duration - Bj.t.duration > 0	

5.2 Specifying Relative Temporal Position

Given two events A1 and B1, the relative temporal position between these events refers to the relationship between the temporal starting and ending points of the events. It describes information such as whether A1 starts before B1 or whether A1 and B1 finish at the same time. In order to be complete, a temporal visual query language needs to be able to specify any primitive temporal relationship [1] as well as any combination of these primitives. This section describes how our TVQL can be used to accomplish this.

Primitive Temporal Relationships

Allen [1] describes thirteen primitive temporal relationships between two events: *before, meets, during, starts, finishes, overlaps,* the symmetric counterparts to these six relationships, and the *equals* relationship (see first column of Table 2). These temporal relationships can be described in terms of the relationships between the temporal starting and ending points of each of the events. Consider two events A1 and B1, with starting and ending points a_o,a_f and b_o,b_f, respectively (i.e., where $a_o = A1.t.start\_time$, $a_f = A1.t.end\_time$, $b_o = B1.t.start\_time$, and $b_f = B1.t.end\_time$):



There are four pairwise endpoint relationships between these events:

$$a_o\theta b_o, a_o\theta b_f, a_f\theta b_o, \text{ and } a_f\theta b_f,$$

where $\theta \in \{<, >, =\}$. We assume that each event has duration greater than 0 so that the conditions $a_o < a_f$ and $b_o < b_f$ always hold true.

Similar to relative duration, these relationships can be redefined in terms of differences:

$$\begin{array}{lll}
a_o - b_o\theta 0, & a_o - b_o & = & A1.t.start_time - B1.t.start_time; \\
a_o - b_f\theta 0, & a_o - b_f & = & \text{A1.t.start_time - B1.t.end_time}; \\
a_f - b_o\theta 0, & a_f - b_o & = & A1.t.end_time - B1.t.start_time; \\
a_f - b_f\theta 0, & a_f - b_f & = & A1.t.end_time - B1.t.end_time;
\end{array}$$

where $\theta \in \{<, >, =\}$. Recall that the benefit of describing these relationships in terms of differences is that it allows us to specify quantitative and continuous ranges of values, which is a natural specification to be handled by DQ sliders. All of Allen's thirteen temporal relationships can be uniquely defined by specifying one to three of these endpoint relationships. The last four columns in Table 2 summarize the minimum endpoint relationships required to uniquely specify a corresponding primitive temporal relationship (r_t). Double-lined boxes

Table 2 Primitive Temporal Relationships. Double-line borders and bold values indicate minimum endpoint relationships required.

r_t [1]	graphical description	Freksas icons [8]	$a_o - b_f$	$a_o - b_o$	$a_f - b_f$	$a_f - b_o$
< before	a_o a_f b_o b_f		< 0	< 0	< 0	**< 0**
m meets			< 0	< 0	< 0	**= 0**
o overlaps			< 0	**< 0**	**< 0**	**> 0**
fi finished by			< 0	**< 0**	**= 0**	> 0
di contains			< 0	**< 0**	**> 0**	> 0
si started by			< 0	**= 0**	**> 0**	> 0
= equals			< 0	**= 0**	**= 0**	> 0
s starts			< 0	**= 0**	**< 0**	> 0
d during			< 0	**> 0**	**< 0**	> 0
f finishes			< 0	**> 0**	**= 0**	> 0
oi overlapped by			**< 0**	**> 0**	**> 0**	> 0
mi met by			**= 0**	> 0	> 0	> 0
> after			**> 0**	> 0	> 0	> 0

indicate which relationships are required, whereas the other relationships can automatically be inferred.

Given four dynamic query filters for the four endpoint relationships, we can thus specify any *primitive* temporal relationship. Therefore, we incorporate these filters into our temporal visual query language. We do so by integrating them into a temporal query palette. Because a_o - b_o and a_f - b_f query filters can be used to uniquely define over half of the primitive temporal relationships, they are placed as the top two filters (see Figure 8).

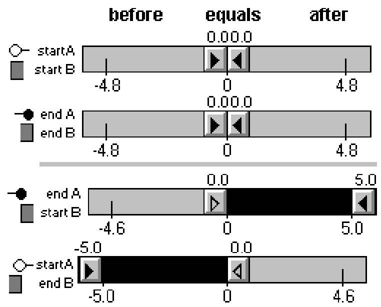


Figure 8 Query filters for specifying relative temporal position queries. Example: Given a subset A bound to "student S1 working" and a subset B bound to "student S2 working," we can specify the query "Show all situations where students S1 and S2 start and finish working at the same time" by setting the top filter startA-startB=0 and the second filter endA-endB=0.

Similar to the relative duration query filter, we provide descriptive labels to the left of the filters and qualitative labels above the filters, thereby allowing users to "read" the query filters more easily (see Section 6.2). Note that the arrows on the thumbs of the query filters in Figure 8 are always pointing inwards, thereby allowing the user to specify a *continuous* range. The significance of a continuous range is described in more detail below in the section on temporal neighborhoods.

Constraints and Interdependencies Between Filters

As in the case of other query filters (i.e., those used to specify a subset), these four temporal query filters are bound to one another so that users cannot specify invalid queries. That is, when one query filter is changed, the other three are automatically updated to exclude any invalid ranges, if necessary.

In the example in Figure 8, the user needs to set both of the top two filters. However, only one filter can be set at a time. Suppose the user first specifies the top filter (i.e., setting startA-startB to 0). As soon as this new value is set, the bottom two filters are *automatically* updated so that endA-startB $>$ 0 and startA-endB $<$ 0. These latter two filters are updated because these are the only valid ranges for them when startA-startB = 0 (see the last four columns of Table 2). The endA-endB filter remains unchanged, because it can have any valid value when startA-startB = 0. After the user sets endA-endB = 0, neither of the other three filters are updated because they are already set to include valid values. In this subsection, we present the strategies we have developed to automatically update the temporal query filters to maintain consistency among the temporal endpoint relationships they represent.

Table 3 Relationships between query filter ranges and temporal primitives.

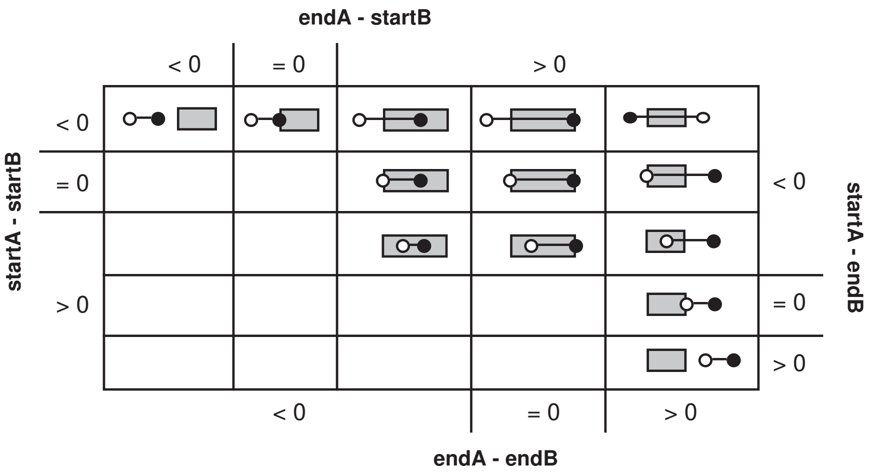


Table 3 formally presents the relationships between the temporal position query filter ranges and the corresponding temporal primitives. Based on the assumption that all A and B events have durations greater than zero, the table identifies invalid combinations and required constraints for the filters. In this way, the table provides indications of how to bind these filters to one another. For example, we can use the table to determine constraints for the previous example, where we were setting startA-startB = 0. In this situation where we are looking for all events that start at the same time, we see from the table that we are also constraining our search to find all events in which endA-startB $>$ 0, and startA-endB $<$ 0. There are no constraints on endA-endB because the table indicates that when startA-startB = 0, endA-endB can be <, =, or > 0. The constraints make sense, since when events start at the same time, the starting point of either event should be less than the ending point of the other.

While Table 3 is sufficient for coarse-grained constraints, it does not provide enough information for more detailed constraints. For example, if we are looking for events of type A that start one minute after events of type B start (i.e., startA-startB = 1) then we must make sure we also have a range specification for endA-startB that includes some values greater than 1. This is due to the inherent condition that all events have duration greater than zero. Table 3 indicates that when we have startA-startB = 1 (i.e., startA-startB > 0), then we must also have endA-startB > 0. However, it is does not provide detailed enough information to indicate how much endA-startB should be greater than zero.

Because we know that duration is defined as durA = endA-startA and durB = endB-startB, we can derive exact relationships between each of the query filters. For example:

$$\begin{aligned} \text{startA-startB} &= (\text{startA-endB}) + (\text{endB-startB}) \\ &= (\text{startA-endB}) + \text{durB} \end{aligned}$$

Table 4 summarizes the filter relationships that we can derive in a similar manner and indicates the implications of these relationships for constraining the interdepencies between the filters. That is, the implications can now be used to dictate the fine-grained constraints between the filters. The table indicates that when either of the top two temporal position query filters are adjusted, both of the bottom filters must be checked to see if they should be constrained. Similarly, if the range of either of the bottom two filters are changed, all three of the other filters must be checked for possible constraints. Thus, when we add larger values to the range of the startA-startB query filter (i.e., move the right query filter thumb to the right), we must also check the range value of the endA-startB query filter to make sure it contains large enough values. Similarly, when we add smaller values to the range of the startA-startB query filter (i.e., move the left query filter thumb to the left), we must also check the range value of the startA-endB query filter to make sure it contains small enough (i.e., lesser) values.

Table 4 Algebraic Relationships Between Temporal Position Query Filters.

Query Filter (QF)	Relationships to Other QFs	Implications (given durA > 0, durB > 0)
(startA-startB)	= (endA-startB) - durA = (startA-endB) + durB	< (endA-startB) > (startA-endB)
(endA-endB)	= (endA-startB) - durB = (startA-endB) + durA	< (endA-startB) > (startA-endB)
(endA-startB)	= (startA-startB) + durA = (endA-endB) + durB = (startA-endB) + durA + durB	> (startA-startB) > (endA-endB) >> (startA-endB)
(startA-endB)	= (startA-startB) - durB = (endA-endB) - durA = (endA-startB) - durA - durB	< (startA-startB) < (endA-endB) << (endA-startB)

Combining Temporal Primitives by Neighborhood

In addition to the thirteen primitive temporal relationships, we may also need to specify a combination of these primitives. For example, we may wish to find all events where student S1 starts working at the same time that student S2 starts working (i.e., they start at the same time, but may or may not end at the same time). Because the filters allow us to specify ranges of values for the endpoint relationships, we can use them to specify such a combination of temporal relationships. That is, we can set startA-startB to 0 and endA-endB to "any" in the case of the example (see Figure 9). Note that this one query palette now specifies a disjunctive combination of related primitive temporal relationships, corresponding to the disjunction of several primitive predicates in typical textual query languages (e.g., [24]). That is, in the example, setting startA-startB = 0 and endA-endB = any is equivalent to requesting events of type A which "start", "equal", or are "started by" events of type B.

Moreover, because the filters can specify ranges and because they are bound to one another to prevent invalid combinations, we can use the filters to specify single "neighborhoods" of related temporal primitives. Freksa [8] defines two primitive temporal relationships between two events to be (*conceptual*) *neighbors* if a continuous change (e.g., shortening, lengthening, or moving of the duration of the events) to the events can be used to transform either relation to the other [without passing through an additional primitive temporal relationship]. Thus, the "overlaps" and "finished by" relations in Table 2 *are*

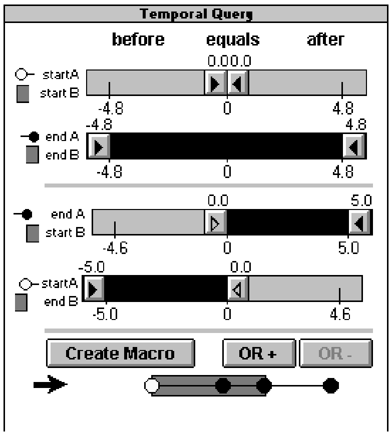


Figure 9 Using dynamic query filters to specify a temporal neighborhood. Example: Student S1 starts working at the same time that student S2 starts working. Beginning with the query specified in the previous figure (i.e., "student S1 and S2 start and finish working at the same time") and increasing the range of endA-endB to include all valid values, we can use the query filters to specify a conceptual neighborhood of (possible) temporal relationships. (Note: This figure presents the full temporal query palette. The use of the "OR+" and "OR-" buttons are described at the end of this sub-section while the "Create Macro" button is described later in this section. The temporal diagram at the bottom of the palette visually describes the query specified (see Section on Temporal Diagrams).)

neighbors, because we can move the ending point of A from the middle of B to the end of B without specifying any additional primitive relationship. On the other hand, the "overlaps" and "contains" relations are *not* neighbors. This is because we cannot move the ending point of A past the ending point of B without first passing through the "finished by" relation. A set or combination of temporal relationships between two events then forms a (*conceptual*) *neighborhood* if it consists of relations that are path-connected through conceptual neighbors. Thus, the use of continuous dynamic query filter *sliders* to specify temporal relationships allows us to capture meaningful disjunctions of the temporal primitives.

Disjunctive Operator: Combining Temporal Primitives and Neighborhoods

The disjunctive operator allows users to create a temporal query consisting of any combination of primitive and neighborhood queries. The operator consists of two buttons — an "OR+" and an "OR-" button (see Figure 9). When users click on OR+, the current state of the temporal query filters is saved in the current temporal diagram (see Section 6.1), and a copy of that diagram is added to the bottom of the palette. An arrow to the left of the diagrams indicates the current part of the OR query that is being specified or edited. If users wish to edit a previously specified part, they can simply click on its corresponding diagram, which moves the arrow to indicate that that part of the OR query is being edited. In addition, the dynamic query filters are also automatically updated to match the part of the disjunctive temporal query being edited. The OR- button can be used to remove the currently selected part of the disjunction.

Consider the example where a user wishes to find all events where student S1 and student S2 start working at the same time, but finish working at different times. This example requires a discontinuous range for the same endpoint relation (i.e., for the endA-endB filter, since we need to set endA-endB < 0 or endA-endB > 0), thus making it impossible to specify this with one set of query filters. Figure 10 illustrates the use of dynamic query filters and the disjunctive operator for this example. A user can specify the first half of the disjunction by setting startA-startB $= 0$ and endA-endB < 0 (see Figure 10a), and then clicking on the OR+ button. Once the OR+ button has been clicked, a new temporal diagram appears at the bottom of the query palette. An arrow points to this newly created temporal diagram to indicate the part of the disjunction being specified. A user can then specify the remainder of this query by setting endA-endB > 0. The new diagram is automatically updated to reflect the temporal primitives specified by the query filters (see Section 6.1 on temporal diagrams). The final state of the temporal query palette for this example disjunctive query is presented in Figure 10b. This example illustrates how the sliders and the disjunctive operator form complementary mechanisms which are sufficient to specify any disjunctive combination of temporal (position) relationships between two sets of events.

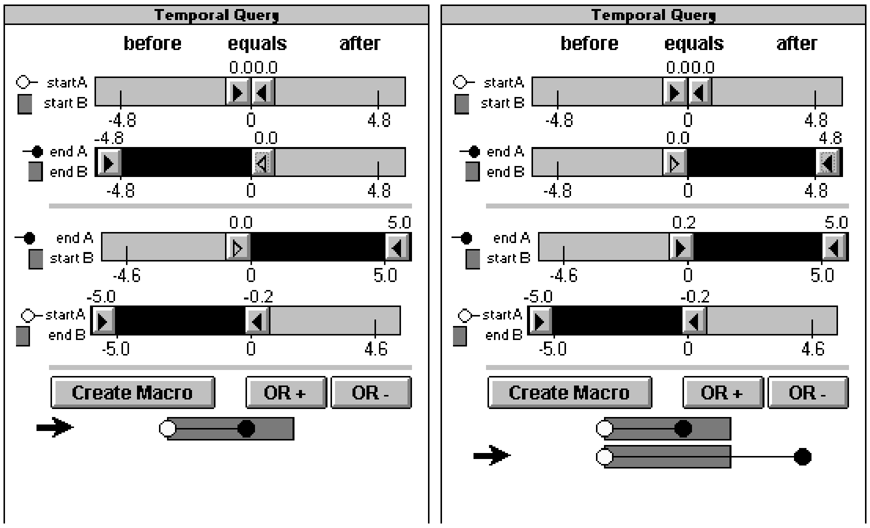


Figure 10 Using the disjunctive operator to specify a non-neighborhood. Example: Students S1 and S2 start working at the same time but finish working at different times. The user specifies this query by forming the following disjunction: "students S1 and S2 start working at the same time but student S1 finishes before student S2" OR "students S1 and S2 start working at the same time but student S1 finishes after student S2." Part (a) of the figure shows the state of the temporal query palette after the user has specified the first part of the disjunctive query. Part (b) shows the final state of the palette after the user has clicked the OR+ operator and finished specifying the second half of the disjunctive query.

5.3 User-Created Macros

Although users can easily specify temporal queries through the temporal query filters, they may have a tendency to use some settings over and over again. For this reason, we provide mechanisms for users to 1) define and save "meaningful" groups of, or macros for, temporal relationships and 2) load and re-use these macros. While it may be difficult to define and enumerate all meaningful groups of relationships, we can identify some groups that we expect to be of use. For example, we can specify an "all starts" temporal relationship to describe the case where events start at the same time. In addition, we can provide a button (i.e., operator) to allow users to create and name their own macros for specifying temporal relationships. Users can then load any macro into the temporal query palette and manipulate the query filters starting from the settings specified by the macro.

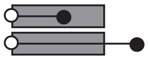
6 VISUAL ENHANCEMENTS TO TVQL

Although users can quickly adjust the query filters to specify any of the individual or any combination of the relative temporal position primitives, they may find difficulty in determining whether or not the query specified corresponds to the desired query semantics. Part of this ambiguity may be inherent in the finely-grained definitions of the primitives (e.g., three different primitives meet the specification that events "start" at the same time). In order to reduce ambiguity, we have thus incorporated two techniques for providing quick visual confirmation of the specified query — the use of temporal diagrams and descriptive labels. We consider these aids to be unique augmentations to our temporal query language, increasing the utility of our approach to novice users. Each of these is described in more detail below.

6.1 Temporal Diagrams

Figures 9 and 10 include temporal diagrams which we have added to the bottom of the temporal query palette to enhance usability. These diagrams are qualitative visual representations of the specified temporal position query. In the diagrams, the rectangle refers to B events, while the connected circles are used to designate potential relative positions of A events. The open circles represent possible starting points for set A, while filled circles represent possible ending points. Freksa [8] uses similar diagrams to describe individual relationships but then uses different icons to describe combinations of temporal relationships (see columns 2 and 3 of Table 2). While the use of Freksa's icons provides a compact visual description, it is more difficult to decipher than our temporal diagrams. This is partly due to the fact that individual endpoint relationships (e.g., differences between starting vs. ending points) are obscured. That is, temporal queries model relationships over time, and this continuous temporal dimension is not explicitly captured in Freksa's icons. In addition, Freksa did not intend to develop a query language for users, but rather to demonstrate some theoretical principles of temporal neighborhoods. Our goals thus are met by a more intuitive capture of individual temporal relations and their relationships. Table 5 presents sample comparisons between the use of Freksa's icons and the temporal diagrams we propose. These examples illustrate how our temporal diagrams not only indicate the number of primitives combined, but also that these primitives share the same starting point.

Table 5 Comparison of Freksa's Icons to Temporal Diagrams.

Freksa's icons	Temporal Diagrams
	
	

We have developed a transformation function to automatically generate the temporal diagrams from the relative temporal position query filters. The primitive operators of this transformation function are defined using Table 6. We introduce a half-filled circle (e.g., visual results of (11) and (12) from Table 6) to denote the overlap of starting and ending points of A.

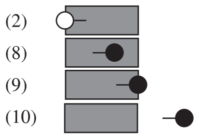
The transformation function has twelve (vs. thirteen, the number of temporal primitives) components because it evaluates potential *endpoints* specified rather than specific primitives selected. That is, it is used to determine which of the five starting points of A are specified, which of the five ending points of A are specified, and whether there is a case where a starting and ending point of A overlap at either the starting or ending points of B. The function is complete because it tests for each possible endpoint position.

In general, the relative temporal position query filters pass values to the transformation function. Each part (i.e., line) of the function evaluates as true or false. The visual results of each valid part of the transformation function are then collapsed into one temporal diagram and a line is used to connect the endpoints. Consider the example of Figure 9: "student S1 starts working at the same time that student S2 starts working." In this example, we have startA-startB = 0, endA-endB = any, startA-endB < 0 and endA-startB > 0. Using

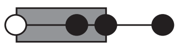
Table 6 Transformation function used to define temporal diagrams.

	Filter Value(s)	Visualized by
(1)	startA-startB < 0	
(2)	startA-startB = 0, endA-startB <> 0	
(3)	startA-startB > 0, startA-endB < 0	
(4)	startA-endB= 0, endA-endB <> 0	
(5)	startA-endB> 0	
(6)	endA-startB < 0	
(7)	endA-startB = 0, startA-startB <> 0,	
(8)	endA-startB > 0, endA-endB < 0	
(9)	endA-endB = 0, startA-endB<> 0	
(10)	endA-endB > 0	
(11)	startA-startB = 0, endA-startB = 0	
(12)	endA-endB = 0, startA-endB = 0	

the transformation operators depicted in Table 6, we see that the following parts evaluate to true:



This leads to the following composite diagram:



As described in Section 5.2, query filters are linked together so as to permit the specification of only logically feasible situations. Hence, only valid relationships will be passed to the transformation function from the query filters. As a consequence, it is not possible to derive an invalid diagram, such as one in which the only starting point of A comes after one or more of its ending points.

In addition, because only continuous ranges are specified by the query filters, all valid starting and ending point combinations of the diagrams are included. A new temporal diagram is derived for each part of a query formed using the disjunctive OR+ operator (see Figure 10).

The temporal diagrams presented in this section represent a valuable aid for confirming the specified query to the users in a visually intuitive manner (e.g., see Figure 10). This should increase the speed and certainty with which users modify or construct complex queries.

6.2 Descriptive Labels: Additional Textual and Visual Aids

In order to provide further clarification of the specified temporal query, we also include textual and iconic descriptive labels (see Figures 7 to 10). The text labels to the left of the query filter identify the relationship being specified. The corresponding icons to the left of the text labels of the relative temporal position query filters provide visual cues to aid users in quickly identifying which query filter they wish to adjust. These icons are the corresponding part of the A and B visual representations which are used in the temporal diagrams. For example, *start A* is represented by an *open* circle. A short line segment is added to the right of the open circle to emphasize that it represents the *starting* point (see Figures 8 to 10).

The labels above the query filters provide *qualitative* descriptions for the corresponding underlying quantitative ranges. The relative duration query filter uses *less*, *equals*, and *greater* qualitative descriptors (see Figure 7) while the relative temporal position query filters use the *before*, *equals*, and *after* descriptors (e.g., see Figure 10). Together, the descriptive labels above and to the left of the temporal query filters allow users to formulate relative temporal queries without having to translate them into notations (e.g., a_o) and algebraic equations. In addition, if users are temporally browsing the data through (randomly) manipulating the query filters (vs. intentionally looking for a specific temporal relationship), they can use the descriptive labels to "read" the specified query once they have identified an interesting result. In Figure 10a, for example, the query palette can be read as "the start of A equals the start of B, and the end of A is before the end of B." The qualitative relationship specified can also be visually verified with the corresponding temporal diagram. In this way, the descriptive labels and temporal diagrams complement each other.

7 MAPPING TVQL TO SYSTEM QUERIES

This section describes how the VQL processor will map the temporal query specification expressed by the DQ filters into an internal system query for the Database Manager. There are three steps to this mapping:

1. Each of the temporal DQ filters corresponding to a part of a disjunctive temporal query (or simply a single set of the filters, if there is no disjunction) is mapped to a temporal predicate, using tables 7a and 7b below;
2. these predicates are AND-ed together;
3. each of the disjunctive components are OR-ed together.

Table 7a Mapping temporal DQ filter name to corresponding system notations.

Temporal DQ Filter Name	**Corresponding System Notation**
startA-startB	Ai.t.start_time - Bj.t.start_time
endA-endB	Ai.t.end_time - Bj.t.end_time
startA-endB	Ai.t.start_time - Bj.t.end_time
endA-startB	Ai.t.end_time - Bj.t.start_time
durA-durB	Ai.t.duration - Bj.t.duration

In the case of the example in Figure 8 ("Students S1 and S2 start and finish working at the same time"), where there is no disjunction, the VQL Processor would first use the above table to map the startA-startB query filter to the temporal predicate Ai.t.start_time - Bj.t.start_time = 0, the endA-endB query filter to the temporal predicate Ai.t.end_time - Bj.t.end_time = 0, etc. These temporal predicates would then be ANDed together to build a temporal clause for the final query (ORing is not necessary, since the example does not include a disjunction). The resulting textual query generated for this example corresponds to:

```
range of description(Ai).content.name = "student S1"
range of description(Ai).content.action = "working"
```

Table 7b Mapping temporal DQ filter thumb settings to algebraic range specifications

DQ Thumb Settings for DQ Filter <DQ Var>	Corresponding Algebraic Range Specification
▶ ◀ , leftVal = rightVal	<DQ Var> = leftVal
▶ , leftVal	<DQ Var> >= leftVal
▷ , leftVal	<DQ Var> > leftVal
◀ , rightVal	<DQ Var> <= rightVal
◁ , rightVal	<DQ Var> < rightVal

```
range of description(Bj).content.name = "student S2"
range of description(Bj).content.action = "working"
retrieve into A_R_B (a=Ai.id, b=Bj.id)
where Ai.id ≠ Bj.id
when  Ai.t.start_time - Bj.t.start_time = 0
   AND Ai.t.end_time - Bj.t.end_time = 0
   AND (Ai.t.start_time - Bj.t.end_time ≥ - 300
        AND Ai.t.start_time - Bj.t.end_time < 0)
   AND (Ai.t.end_time - Bj.t.start_time > 0
        AND Ai.t.end_time -Bj.t.start_time ≤ 300)
   AND Ai.t.duration - Bj.t.duration = 0;
```

In this example, the last three lines of the temporal condition provide redundant information (e.g., if two events start and end at the same time, then they must also have the same duration). This indicates how the TVQL Processor could optimize the generated query to reduce the number of temporal constraints specified. We are currently investigating the use of special-purpose indexing structures for optimizing the TVQL Processor even further.

8 PRELIMINARY EVALUATION

The example mapping of a TVQL query to a text-based query in the previous section illustrates one of the advantages of our visual approach. By using the TVQL, users can specify temporal constraints by simply adjusting the ranges of the temporal query filters. This can be done through *direct manipulation*, without having to type predicates as for a textual query. In this way, the use of filters allows users to *browse* the database without having to remember the syntax of a text-based language. In other words, users can also use direct manipulation to *incrementally* adjust their queries. In the text-based language, they would have to retype the query over or regain and then edit the text.

On the other hand, a forms-based query interface might address some of the drawbacks of a text-based query language. We hypothesize, however, that our visual query approach will be more effective than a forms-based language for the purpose of searching for *temporal trends* in the video data. This hypothesis is supported by a study comparing the use of dynamic queries to a forms-based query language [2]. In this study, the researchers found that users performed significantly better using dynamic queries over a forms-based approach for three out of five tasks, one of which involved looking for trends in the data.

We are currently in the process of evaluating users' conceptual understanding of the TVQL interface through user testing. Preliminary results indicate that the dynamically updated temporal diagrams are strong visual aids explaining the semantics of the query to the users.

9 RELATED WORK

Research in temporal queries has focused more on the context of historical databases rather than on databases of temporal media (e.g., [25]). Historical databases, such as those used to manage medical images [5] are different from databases of temporal media in that they focus on *discrete* changes entered into the databases as an effect of changes made in the real world (e.g., John broke his leg on April 10, 1994). That is, these databases focus on discrete changes to static objects rather than changes in continuous, dynamic media (such as video).

Although other researchers are working on extensions to dynamic query filters, this work has focused on aggregation extensions to the interface [9, 11]. While this recent work can be added to our system for the purpose of forming subsets, they are not sufficient to meet the needs of video analysis described in this chapter. Our extensions to VIS and dynamic queries address these needs by 1) providing a framework for specifying *relative* queries between subsets and 2) providing specialized temporal query filters for specifying relative *tem-*

poral queries between subsets of data. The relative framework will allow us to continue to work on future relative extensions to dynamic queries, such as specialized query filters for spatial and motion analysis. Our temporal query filters are consistent with the original DQ filters in terms of interlinking the filters together and maintaining the notion of progressive refinement through continuous value range selections.

Besides the work in dynamic query filters by Ahlberg et al [2] and others (e.g., [11]), previous work in graphical or visual query languages has focused on diagrammatic, forms-based, or iconic approaches. Diagrammatic query languages allow users to specify queries by using direct manipulation of schema-type constructs to represent a set of objects and their relationships (e.g., [28, 23]). Essentially, you could build a query graphically by representing each object by a circle, attaching predicates to the circles, and using lines to denote relationships between the circles. While such an interface could be used to build temporal queries, it would lack the continuous browsing support provided by our TVQL. Similarly, while novel techniques have been used to combine forms-based queries with automated object extraction from videos (e.g., [21]), such an interface would also require users to specify temporal primitives as well as ranges of temporal values by hand, thus disrupting the ability to review the video data in a temporally continuous manner.

While some work has begun in temporal queries of continuous, dynamic media, such work has tended to focus on issues other than analysis. For example, work by [18] focuses on synchronization and multimedia playback. Work by Davis [7] focuses on repurposing video, using iconic annotations to represent objects and events in a video and collecting these icons in a parallel timeline format. While his iconic language supports temporal encodings including *relative* ones, it requires users to select icons for each type of temporal relationship individually. In addition, it is not clear whether the language is restricted to only retrieving pre-coded relationships. In contrast, our system allows users to *discover* temporal relationships through using the TVQL (which provides *continuous* ranges of temporal values) and to explore the video data in a *temporally continuous* manner.

Research in multimedia databases varies somewhat due to the interpretation of the term *multimedia*. Some researchers consider image databases (i.e., images + text databases) to be multimedia, but such databases do not deal with temporally-based media such as video or audio. On the other hand, databases which handle temporal media tend to focus on semantic or text-based queries as well as on *locating* information rather than *analyzing* it (e.g., [17, 4]). In these types of databases, media are typically images or short clips with textual captions or descriptions. Information is then located by semantic inferencing on these textual descriptions (e.g., [17]). The drawback of this type of approach is that it does not take advantage of the temporal and/or spatial characteristics inherent in the media. While some image databases allow users to search using spatial information, such approaches have either not dealt with temporal media, or have focused on object extraction (e.g., [10]). We distinguish our work from previous work in this area by using both temporal and spatial characteristics of the media, as well as by addressing needs for both retrieval and analysis.

10 CONCLUSION, CURRENT STATUS AND FUTURE WORK

In this chapter, we have presented a framework for video analysis based on applying and adapting a visual information seeking (VIS) approach [3] to spatio-temporal video data. In our extensions to VIS, we have defined a temporal visual query language (TVQL) for specifying *relative temporal* queries, including relative temporal position and duration between video annotations. The four query filters used for relative temporal position enable users to specify any *primitive* temporal relationship as well as *conceptual neighborhood* combinations of these primitives. Together, the temporal query filters provide users with a visual paradigm for *browsing* the video data through direct manipulation and in a *temporally continuous* manner. For completeness, we have provided a disjunctive operator for specifying any combination of temporal primitives and temporal neighborhoods. We also introduced the use of temporal diagrams and descriptive labels to enhance the clarity of the specified query. Lastly, we have supported the reuse of redundant query specifications through user-created macros.

The primary contributions of this work include 1) a VIS approach to video analysis, 2) the temporal visual query language (TVQL) for specifying *relative* temporal queries and for facilitating *temporal analysis* (i.e., searching for temporal trends in video data), 3) a transformation function for deriving temporal diagrams, 4) a description of the automated maintenance of interdependencies between the temporal position query filters, and 5) a formal annotation model for abstracting temporal, spatial, and content-based characteristics from video data. By analyzing an annotation layer on top of the video, we have designed an approach which can be applied to other dynamic media (e.g., animation).

Our approach is not limited to the analysis of video *data* but is one which can also be used to analyze other genres of video such as movies, sports, etc.

The work presented in this chapter is just one component of the design of an integrated environment for video analysis (see Section 3). We have implemented our TVQL in a Windows-based multimedia pc (MPC) environment and are currently conducting a usability study, comparing the effectiveness of TVQL with more traditional forms-based interfaces. We are continuing work on the TVQL processor, the temporal visualizations, and the spatial and motion analysis aspects of our MMVIS environment.

Acknowledgements

This work was supported in part by University of Michigan Rackham Fellowship, NSF NYI #R1-94-57609, equipment support from AT&T, and NSF/ARPA/ NASA digital lab grant to the University of Michigan.

REFERENCES

[1] Allen, J.F. (1983). Maintaining knowledge about temporal intervals. Communications of the ACM, 26(11), 832-843.

[2] Ahlberg, C., Williamson, C., & Shneiderman, B. (1992). Dynamic Queries for Information Exploration: An Implementation and Evaluation. *CHI'92 Conference Proceedings*, 619-626: ACM Press.

[3] Ahlberg, C., & Shneiderman, B. (1994). Visual Information Seeking: Tight Coupling of Dynamic Query Filters with Starfield Displays. *CHI'94 Conference Proceedings*, 313-317: ACM Press.

[4] Chakravarthy, A.S. (1994). Toward Semantic Retrieval of Pictures and Video. *AAAI'94 Workshop on Indexing and Reuse in Multimedia Systems*, 12-18.

[5] Chu, W.W., Ieong, I.T., Taira, R.K., & Breant, C.M. (1992). A Temporal Evolutionary Object-Oriented Data Model and Its Query Language for Medical Image Management. *Proceedings of the 18th VLDB Conference*, 53-64: Very Large Data Base Endowment.

[6] Chua, T.-S., Lim, S.-K., & Pung, H.-K. (1994). Content-Based Retrieval of Segmented Images. *ACM Multimedia'94 Proceedings*: ACM Press.

[7] Davis, M. (1994). Knowledge Representation for Video. *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 120-127: AAAI Press.

[8] Freksa, C. (1992). Temporal reasoning based on semi- intervals. *Artificial Intelligence*, 54(1992), 199-227.

[9] Fishkin, K. and Stone, M.C. (1995). Enhanced Dynamic Queries via Movable Filters. *CHI'95 Conference Proceedings*, 415-420. ACM Press.

[10] Gevers, T. and Smeulders, A.W.M. (1992). Indexing of Images by Pictorial Information. *Visual Database Systems, II* (E. Knuth and L.M. Wegner, Eds.), North Holland: Amsterdam, 93-100.

[11] Goldstein, J. & Roth, S. (1994). Using Aggregation and Dynamic Queries for Exploring Large Data Sets. *CHI'94 Conference Proceedings*, 23-29. ACM Press.

[12] Hampapur, A., Weymouth, T., & Jain, R. (1994). Digital Video Segmentation. *ACM Multimedia'94 Proceedings*, 357-364: ACM Press.

[13] Harrison, B.L., Owen, R., & Baecker, R.M. (1994). Timelines: An Interactive System for the Collection of Visualization of Temporal Data. *Proceedings of Graphics Interface '94*. Canadian Information Processing Society.

[14] Hibino, S. & Rundensteiner, E. (1995a). A Visual Query Language for Identifying Temporal Trends in Video Data. To appear in *Proceedings of the First International Workship on Multimedia Database Management Systems*.

[15] Hibino, S. & Rundensteiner, E. (1995b). Interactive Visualizations for Exploration and Spatio-Temporal Analysis of Video Data. To appear in *IJ-CAI'95 Workshop on Intelligent Multimedia Information Retrieval*.

[16] Hibino, S. & Rundensteiner, E. (Dec 1994). A Graphical Query Language for Identifying Temporal Trends in Video Data. University of Michigan, EECS Technical Report CSE-TR-225-94.

[17] Lenat, D. & Guha, R.V. (1994). Strongly Semantic Information Retrieval. *AAAI'94 Workshop on Indexing and Reuse in Multimedia Systems*, 58-68.

[18] Little, T.D.C. & Ghafoor, A. (1993). Interval-Based Conceptual Models for Time-Dependent Multimedia Data. *IEEE Transactions on Knowledge and Data Engineering*, 5(4), 551-563.

[19] Mackay, W. E. (1989). EVA: An experimental video annotator for symbolic analysis of video data. *SIGCHI Bulletin*, 21(2), 68-71.

[20] Nagasaka, A. and Tanaka, A. (1992). Automatic Video Indexing and Full-Video Search for Object Appearances. *Visual Database Systems, II* (E. Knuth and L.M. Wegner, Eds.), 113-127. Elsevier Science Publishers.

[21] Oomoto, E. & Tanaka, K. (1993). OVID: Design and Implementation of a Video-Object Database System. *IEEE Transactions on Knowledge and Data Engineering*, 5(4), 629-643.

[22] Roschelle, J., Pea, R., & Trigg, R. (1990). VIDEONOTER: A tool for exploratory analysis (Research Rep. No. IRL90- 0021). Palo Alto, CA: Institute for Research on Learning.

[23] Santucci, G. & Sottile, P.A. (1993). Query by Diagram: a Visual Environment for Querying Databases. *Software — Practice and Experience*, 23(3), 317-340.

[24] Snodgrass, R. (1987). The Temporal Query Language TQuel. *ACM Transactions on Database Systems*, 12(2), 247-298.

[25] Snodgrass, R. (1992). Temporal Databases. *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space* (A.U. Frank, I. Campari, and U. Formentini, Eds.), Springer-Verlag: New York, 22-64.

[26] Ueda, H., Miyatake, T., Sumino, S., & Nagasaka, A. (1993). Automatic Structure Visualization for Video Editing. *InterCHI'93 Proceedings*, (pp. 137-141): ACM Press.

[27] Weber, K. & Poon, A. (1994). Marquee: A Tool for Real-Time Video Logging. *CHI'94 Conference Proceedings*, 58-64: ACM Press.

[28] Whang, K., Malhotra, A., Sockut, G., Burns, L., & Coi, K-S. (1992). Two-Dimensional Specification of Universal Quantification in a Graphical Database Query Language, *IEEE Transactions on Software Engineering*, 18(3), 216-224.