University of Alberta

Video Modeling and Its Integration in a Temporal Object Model

by

John Z. Li, Iqbal A. Goralwalla, M. Tamer Özsu, Duane Szafron
Laboratory for Database Systems Research
Department of Computing Science
University of Alberta
Edmonton, Alberta
Canada T6G 2H1
{zhong,iqbal,ozsu,duane}@cs.ualberta.ca

Technical Report TR 96–02
January 1996

DEPARTMENT OF COMPUTING SCIENCE
The University of Alberta
Edmonton, Alberta, Canada

Abstract

Video modeling has become a topic of increasing interest in the area of multimedia research. One of the key aspects in the video medium is the temporal relationship between video frames. In this report[1] we propose a tree-based model for specifying spatial and temporal semantics of video data. Our focus here is on the temporal issues. We present a unique way of integrating our video model into an objectbase management system which has rich multimedia temporal operations. We further show how temporal histories are used to model video data. Using histories to model video data is both simple and natural. It also can lead to a uniform behavioral model. A user can then explore the video objectbase using object-oriented techniques. Such a seamless integration gives a uniform interface to end users. The integrated video objectbase management system supports a broad range of temporal queries and is extensible, thus allowing the easy incorporation of new features into the system.

Keywords: multimedia, temporal, object-oriented, database, video model, query, clips

[1]This research is supported by a grant from the Canadian Institute for Telecommunications Research (CITR) under the Network of Centre of Excellence program of the Government of Canada.

Contents

1 Introduction 4

2 The Common Video Object Tree Model 5

2.1 Video Clip Sets 6

2.2 Salient Objects 7

2.3 The Common Video Object Tree 9

3 The OBMS Support 12

3.1 TIGUKAT Model Overview 13

3.2 The TIGUKAT Temporal Object Model 14

3.2.1 Temporal Ontology 14

3.2.2 Temporal Histories 17

4 System Integration 17

4.1 Integrating the CVOT Model 18

4.2 Modeling Video Features 21

4.3 Applications of the CVOT model 22

4.4 Query Examples 22

5 Related Work 26

6 Conclusions 27

List of Figures

1 CVOT System Architecture 5
2 Stream-based Video Clips and Frames 6
3 Salient objects and Clips 9
4 Common Video Object Tree Built by GMCO Algorithm 10
5 Greedy Maximum Common Objects Algorithm 11
6 Expanding Subroutine 12
7 The Basic Time Type Hierarchy 14
8 Different Types of Ordering Relations Between Intervals 15
9 The Video Type System 18
10 Video Hierarchy From Viewers' Point of View 23

List of Tables

1 Behaviors on time intervals and time instants 16
2 Behaviors on histories and time-stamped objects 17
3 Behavior Signatures of Videos 19
4 Behavior Signatures of Clips 20
5 Behavior Signatures of Frames 20
6 Some Behavior Signatures of Events and Salient Objects 21
7 Simple Behavior Signatures of Some Video Objects 23

1 Introduction

Management of multimedia data poses special requirements on the database management systems. In a broad sense *multimedia data* includes any data types, e.g., numeric data, character strings, graphics, images, audio, video, and animation, and data from arbitrary sources [Kim95]. In this paper, we concentrate on the video data type, especially *video modeling*, which is the process of translating raw video data into an efficient internal representation for capturing video semantics. A video model is an essential part of an abstract multimedia information system model which can be used as the basis of declarative querying. The abstract model has to be mapped to a concrete one. Object-oriented technology is generally accepted as a promising tool for modeling multimedia data [WK87, CK95, DDI+95].

The procedural process of extracting video semantics from a video is called *video segmentation*. There are two approaches to video segmentation in an object-oriented context: *stream-based* and *structured*. In a stream-based approach, a clip is considered as a sequence of *frames* that are displayed at a specified rate while in a structured approach a clip is considered as a sequence of scenes. Each approach has its own advantages and disadvantages as described in [Gha96]. However, very little work [Gha96] has been done on the structured approach because of its technical difficulties. On the other hand, the stream-based approach has received most of the research attention because of its technical feasibility. We concentrate on stream-based approaches.

Most of the video models either employ image processing techniques for indexing video data or use traditional database approaches based on keywords or annotated textual descriptions [SZ94, OT93, LAF+93] to represent video semantics. In most cases, the annotated description of the video contents is created manually. This is a time consuming process. This paper proposes a video model called the Common Video Object Model (CVOT). The model has the capability of automatic video segmentation and incorporates temporal relationships among video objects. This allows native support for a rich set of temporal multimedia operations.

We seamlessly integrate the abstract CVOT model to a powerful temporal object model providing concrete objectbase management system (OBMS)[2] support for video data. The system that we use in this work is TIGUKAT[3] [ÖPS+95] which is an experimental system under development at the University of

[2]We prefer the terms "objectbase" and "objectbase management system" over the more popular terms "object-oriented database" and "object-oriented database management system", since the objects that are managed include code as well as data. Furthermore, we are using the term *video objectbase*, instead of *video database*.

[3]TIGUKAT (tee-goo-kat) is a term in the language of Canadian Inuit people meaning "objects." The Canadian Inuits (Eskimos) are native to Canada with an ancestry originating in the Arctic regions.

Alberta. We exploit the behaviorality and uniformity of the TIGUKAT object model in incorporating the CVOT model uniformly.

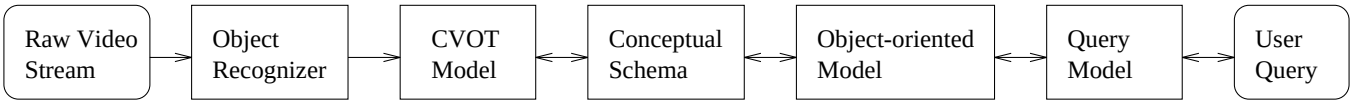

Figure 1: CVOT System Architecture

Figure 1 shows the architecture of our proposed system. Rww video data is processed by an *Object Recognizer* which uses image processing technique to recognize salient (physical) objects. The salient objects are encoded in the CVOT model by their properties, such as size, location, moving direction, etc. Then a conceptual schema can be built based on the analysis of the encoded information. A unified model is necessary for users to query the system and for the system to process the queries. An object-oriented model is used because of its powerful representation of the users' view and its suitability to multimedia data [WK87]. The major contributions of this paper are the following:

- A new model for organizing video clips based on a common object tree is proposed. Compared to the existing models, this new model is simple and efficient. The flexibility of video segmentation is another feature of this model.
- A unique way of integrating the CVOT model into an objectbase management system with rich temporal operations is presented.
- A new uniform approach of modeling video using temporal histories is introduced.
- The integrated video objectbase supports qualitative temporal operations on video data.

The rest of the paper is organized as follows. We introduce the Common Video Object Tree model and an algorithm for building trees in Section 2. A brief discussion of the TIGUKAT system and its temporal extension is presented in Section 3. Section 4 describes how the CVOT model can be seamlessly integrated into the TIGUKAT objectbase management system. We also present many query examples to show the expressiveness of such an integrated system. Our conclusions and future work are given in the last section.

2 The Common Video Object Tree Model

Each *video* consists of a number of *clips*. A *clip* is a consecutive sequence of *frames*, which are the smallest units of video data. The information about semantics of a video must be structured so that indexes can be

built for efficient data retrieval from a video objectbase. The functionality of a video objectbase depends on its model of time.

There are several different ways to segment a video into clips, e.g., by *fixed time intervals* or by *shots*. A *fixed time interval* segmentation approach divides a video into equal length clips using a predefined time interval (e.g. 2 seconds) while a *shot* is a set of continuous frames captured by a single camera action [HJW95]. In our model there is no restriction on how videos are segmented. Without loss of generality, we assume that any given video stream has a finite number of clips and any clip has a finite number of frames as shown in Figure 2. The main idea of the Common Video Object Tree model is to find all the common objects among clips and to group clips according to these objects. We use a tree structure to represent such a clip group. In this section we give a formal definition of the model and then give an algorithm for constructing the tree.

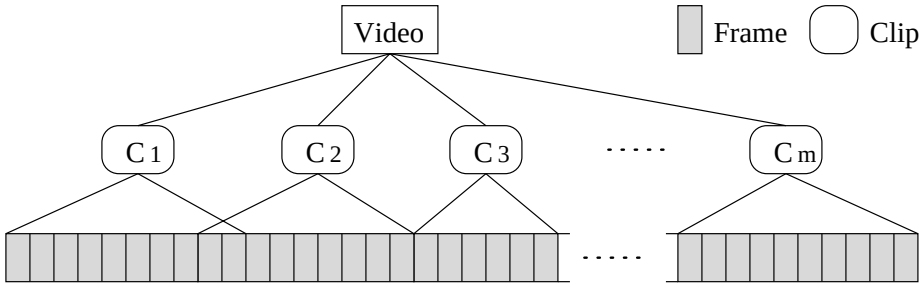

Figure 2: Stream-based Video Clips and Frames

2.1 Video Clip Sets

A clip is associated with a time interval $[t_s, t_f]$. More specifically a clip is a set of consecutive frames between a start time t_s and a finish time t_f: $[t_s, t_f] = \{t | t_s \leq t \leq t_f\}$ where t_s and t_f are the relative (discrete) time instants in a given video and $t_s \leq t_f$. Since all clips have a start and finish time, a partial order could be defined over clips. To simplify description, we use $C_i = [t_{s_i}, t_{f_i}]$ to mean that clip C_i is associated with a time interval $[t_{s_i}, t_{f_i}]$ although, semantically, C_i should be all the frames within this interval.

Definition 1 Let $C_i = [t_{s_i}, t_{f_i}]$ and $C_j = [t_{s_j}, t_{f_j}]$ be two clips. Then $\preceq$ is defined as the partial order over clips with $C_i \preceq C_j$ iff $t_{s_i} < t_{s_j}$ and $t_{f_i} < t_{f_j}$. Also, $C_i \prec C_j$ iff $t_{s_i} \leq t_{f_i} < t_{s_j} \leq t_{f_j}$.

Definition 2 A set of clips C is said to be *ordered* iff C is finite, i.e., $C = \{C_1, \ldots, C_m\}$ and there exists a partial order such that $C_1 \preceq C_2 \preceq \ldots \preceq C_m$. A set of clips $C = \{C_1, C_2, \ldots, C_m\}$ is said to be *strongly*

ordered iff C is ordered and $C_1 \prec C_2 \prec \ldots \prec C_m$. A set of clips $C = \{C_1 \ldots, C_m\}$ is said to be *perfectly ordered* iff C is ordered and for any two neighboring clips $C_i = [t_{s_i}, t_{f_i}]$ and $C_{i+1} = [t_{s_{i+1}}, t_{f_{i+1}}]$, we have $t_{s_{i+1}} = t_{f_i} + 1$ $(\forall i = 1, 2, \ldots, m-1)$. A set of clips $C = \{C_1 \ldots, C_m\}$ is said to be *softly ordered* iff C is ordered and for any two neighboring clips $C_i = [t_{s_i}, t_{f_i}]$ and $C_{i+1} = [t_{s_{i+1}}, t_{f_{i+1}}]$, we have $t_{f_i} \geq t_{s_{i+1}}$ $(\forall i = 1, 2, \ldots, m-1)$.

Example 1 (a) $C = \{[1, 10], [13, 17], [28, 100]\}$ is an ordered clip set because $[1, 10] \preceq [13, 17] \preceq [28, 100]$ and a strongly ordered set because $[1, 10] \prec [13, 17] \prec [28, 100]$. However, it is not softly ordered because $t_{f_1} = 10 \not\geq t_{s_2} = 13$.
(b) $C = \{[1, 10], [2, 8], [13, 17]\}$ is not ordered since $[1, 10] \not\preceq [2, 8]$ and $[2, 8] \not\preceq [1, 10]$. For the same reason $C' = \{[1, 10], [1, 10], [13, 17]\}$ is not ordered because $[1, 10] \not\preceq [1, 10]$.
(c) $C = \{[1, 10], [11, 17], [18, 100]\}$ is a perfectly ordered clip set because C is ordered and $t_{s_2} = t_{f_1} + 1, t_{s_3} = t_{f_2} + 1$. It is easy to verify that C is also strongly ordered.
(d) $C = \{[1, 10], [8, 17], [15, 30]\}$ is a softly ordered clip set because C is ordered and $t_{f_1} = 10 \geq t_{s_2} = 8$ and $t_{f_2} = 17 \geq t_{s_3} = 15$.

The above examples indicate that ordered clip sets disallow both same interval segmentation (set C' in Example 1(b)) and subinterval segmentation (set C in Example 1(b)). In our model we only consider softly ordered and perfectly ordered clip sets because the associated intervals of their clips can be merged into larger intervals. This is important in the case of a stream-based representation. The following theorem states the relationship between strongly ordered clip sets and perfectly ordered clip sets.

Theorem 1 All perfectly ordered clip sets are also strongly ordered.

Proof: Let $C = \{[t_{s_1}, t_{f_1}], \ldots, [t_{s_m}, t_{f_m}]\}$ be a perfectly ordered clip set. This means $[t_{s_1}, t_{f_1}] \preceq \ldots \preceq [t_{s_m}, t_{f_m}]$ with $t_{s_{i+1}} = t_{f_i} + 1$ $(i = 1, \ldots, m-1)$ and $t_{s_i} \leq t_{s_{i+1}}$ and $t_{f_i} \leq t_{f_{i+1}}$ $(i = 1, \ldots, m-1)$. Hence, from $t_{s_i} \leq t_{f_i} < t_{s_{i+1}} \leq t_{s_{i+1}}$ $(i = 1, \ldots, m-1)$ we have $[t_{s_i}, t_{f_i}] \prec [t_{s_{i+1}}, t_{f_{i+1}}]$ $(i = 1, \ldots, m-1)$. Therefore, $[t_{s_1}, t_{f_1}] \prec \ldots \prec [t_{s_m}, t_{f_m}]$. From the definition of strongly ordered clip sets it follows that C is strongly ordered. ■

Note that the reverse of this theorem is not necessarily true, i.e. strongly ordered clip sets may not be perfectly ordered clip sets. This is shown in Example 1(a).

2.2 Salient Objects

A *salient object* is an interesting physical object in video frames. Each video frame has many salient objects, e.g. persons, houses, cars, etc. We assume there is always a finite set (possibly empty) of salient

objects $SO = \{SO_1, SO_2, \ldots, SO_n\}$ for a given video. The spatial property of an SO_i is defined by a minimum bounding rectangle (X_i, Y_i) and a depth d, where $X_i = [x_{s_i}, x_{f_i}], Y_i = [y_{s_i}, y_{f_i}]$. x_{s_i} and x_{f_i} are salient object SO_i's projection on X axis and similarly for $y_{s_i} and y_{f_i}$. The depth d indicates whether the object is in front or behind other objects. Hence, a salient object's spatial property can be represented by a 3-ary tuple (X_i, Y_i, d) and we call such a tuple a *bounding box*.

Let $\mathcal{SO}$ be the collection of all salient object sets and $\mathcal{C}$ be the collection of all clip sets. We introduce two functions. One is the function $\mathcal{F}$: $SO \rightarrow \mathcal{C}$ which maps a salient object from $SO \in \mathcal{SO}$ into an ordered clip set $C \in \mathcal{C}$. The other is the function $\mathcal{F}'$: $C \rightarrow \mathcal{SO}$ which maps a clip from $C \in \mathcal{C}$ into a salient object set $SO \in \mathcal{SO}$. Intuitively, function $\mathcal{F}$ returns a set of clips which contains a particular salient object while the reverse function $\mathcal{F}'$ returns a set of salient objects which belong to a particular clip. We define the *common salient objects* for a given clip set as those salient objects which appear in every clip within the set. Some salient objects may appear in many different clips, but others may not. Hence, the number of common salient objects between clips are different. In order to quantify such a difference we introduce clip *affinity*.

Definition 3 The *affinity* of m clips $\{C_1, \ldots, C_m\}$ is defined as

$$aff(C_1, \ldots, C_m) = |\mathcal{F}'(C_1) \cap \mathcal{F}'(C_2) \cap \ldots \cap \mathcal{F}'(C_m)|$$

where $\{C_1, \ldots, C_m\}$ is an ordered clip set, $m \geq 2$, $|X|$ is the cardinality of set X, and $\cap$ is the standard set intersection.

Example 2 Figure 3 shows a video in which John using bat bat1 and Ken using bat bat2 are playing table tennis while Mary is watching. After playing, John drives his car home. Let us assume that the salient objects are $SO = \{$john, ken, mary, ball, bat1, bat2, car$\}$ If the video is segmented as in Figure 3, then $C = \{C_1, C_2, C_3, C_4, C_5\}$ is a perfectly ordered clip set. Furthermore, john, ball, and bat1 are in C_1; john, mary, ball, bat1 are in C_2; ken, ball, bat2 are in C_3; ken, ball, bat2 are in C_4; and john, car are in C_5;
Then,

$\mathcal{F}(\text{john}) = \{C_1, C_2, C_5\}$ $\quad$ $\mathcal{F}'(C_1) = \{\text{john}, \text{ball}, \text{bat1}\}$

$\mathcal{F}(\text{ken}) = \{C_3, C_4\}$ $\quad$ $\mathcal{F}'(C_2) = \{\text{john}, \text{ball}, \text{bat1}, \text{mary}\}$

$\mathcal{F}(\text{mary}) = \{C_2\}$ $\quad$ $\mathcal{F}'(C_3) = \{\text{ken}, \text{ball}, \text{bat2}\}$

$\mathcal{F}(\text{ball}) = \{C_1, C_2, C_3, C_4\}$ $\quad$ $\mathcal{F}'(C_4) = \{\text{ken}, \text{ball}, \text{bat2}\}$

$\mathcal{F}(\text{bat1}) = \{C_1, C_2\}$ $\quad$ $\mathcal{F}'(C_5) = \{\text{john}, \text{car}\}$

$\mathcal{F}(\text{bat2}) = \{C_3, C_4\}$

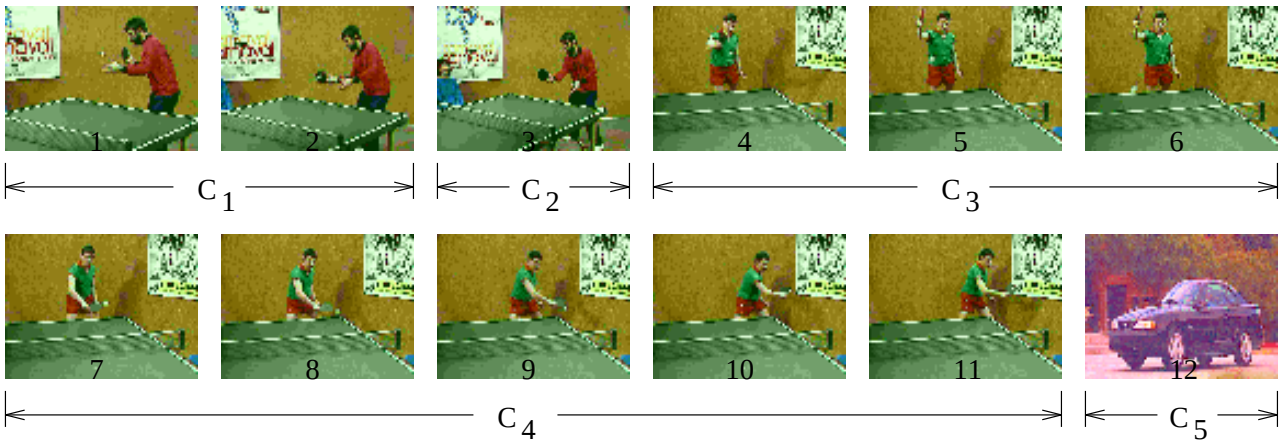

Figure 3: Salient objects and Clips

$\mathcal{F}(\mathsf{car}) = \{C_5\}$.

Now, the affinity of C_1 and C_2 is

$$aff(C_1, C_2) = |\mathcal{F}'(C_1) \cap \mathcal{F}'(C_2)| = |\{\mathsf{john}, \mathsf{ball}, \mathsf{bat1}\} \cap \{\mathsf{john}, \mathsf{ball}, \mathsf{bat1}, \mathsf{mary}\}|$$
$$= |\{\mathsf{john}, \mathsf{ball}, \mathsf{bat1}\}| = 3.$$

Similarly, $aff(C_2, C_3) = 1$, $aff(C_1, C_2, C_3) = 1$, etc.

Theorem 2 The affinity function is *monotonically non-increasing*. That is, if $\{C_1, \ldots, C_m\}$ is an ordered clip set, then $aff(C_1, \ldots, C_k) \geq aff(C_1, \ldots, C_k, C_{k+1})$ where $k = 2, 3, \ldots, m-1$.

Proof: The proof is trivial if we use the set intersection property, $A \cap B \subseteq A$ where A and B are any two sets. Let set A be $\mathcal{F}'(C_1) \cap \ldots \cap \mathcal{F}'(C_k)$ and set B be $\mathcal{F}'(C_{k+1})$. Then, $\mathcal{F}'(C_1) \cap \ldots \cap \mathcal{F}'(C_k) \cap \mathcal{F}'(C_{k+1}) \subseteq \mathcal{F}'(C_1) \cap \ldots \cap \mathcal{F}'(C_k)$. It follows that $|\mathcal{F}'(C_1) \cap \ldots \cap \mathcal{F}'(C_k) \cap \mathcal{F}'(C_{k+1})| \leq |\mathcal{F}'(C_1) \cap \ldots \cap \mathcal{F}'(C_k)|$. Therefore, $aff(C1, \ldots, C_k, C_{k+1}) \leq aff(C_1, \ldots, C_k)$ ■

2.3 The Common Video Object Tree

Clustering clips is an important issue as it affects both the effectiveness and efficiency of query retrievals. A clustering scheme should also maintain any existing temporal relationships among frames. We propose a tree-based model, called the *Common Video Object Tree* (CVOT), which builds a tree based on the common salient objects in a set of clips. Trees provide an easy and efficient way of clustering clips with less complexity than graphs. For any given softly or perfectly ordered clip set C, each leaf node in a CVOT tree is an element of C. All the leaf nodes are ordered from left to right by their time intervals. An internal node represents a set of common salient objects, which appear in all its child nodes. The only

node that can have empty common salient object set is the root node. Every node (including internal, leaf, and root node) has a time interval and a set of salient objects which appear during this time interval. The time interval of an internal node has a start time which is equal to the start time of its leftmost child node and a finish time which is equal to the finish time of the rightmost child node. Figure 4 shows an example of a CVOT tree which is built from Example 2. As seen in Figure 4, the cardinality of the common object set shrinks as we traverse the tree from the leaf nodes to the root. This is in conformance with the monotonically non-increasing nature of clip affinity stated in Theorem 2. The figure also shows how the time intervals are propagated up from the leaf nodes. For example, the internal node N_2 has the interval $[1, 3]$ which is composed from its two child leaf nodes C_1 and C_2. The root always spans all of the time intervals in the whole clip set.

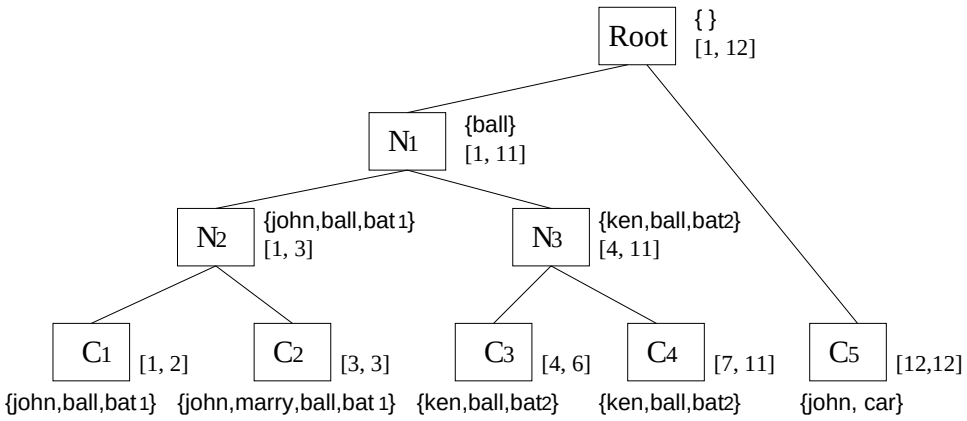

Figure 4: Common Video Object Tree Built by GMCO Algorithm

We have developed a greedy algorithm, called *Greedy Maximum Common Objects* (GMCO), to build the CVOT tree (Figure 5). The GMCO algorithm uses a bottom-up method in building a CVOT tree for a given set of clips. The idea is to find the largest set of neighboring clips without reducing the set's affinity. An internal node is then created for this new set of clips. This process continues until either the common object set is empty or all nodes are merged into one node (root). If the common object set is empty, we directly attach this node to the root of the CVOT.

The algorithm first checks if the clip set is a singleton. If so, the element is attached directly to the root. If the clip set has more elements, each of these are made a leaf node. The next step is to compute a largest clip set without reducing the affinity of this set. This is done by checking whether the affinity of two neighboring clips is zero. If so, the clip set is not expanded because a larger clip set will only decrease its affinity. If the affinity of two neighboring clips is not zero, this value is set to be the initial affinity of the clip set. Then, a subroutine **Expand** is called to compute the largest clip set. **Expand**(C, T, *AFF*)

```
GMCO(C, SO, R)
C = {C_1, ..., C_m}: a softly or perfectly ordered clip set;
SO = {SO_1, ..., SO_n}: a salient object set;
R: root node of a CVOT tree;
{
    NewC: a new ordered clip set initialized to ∅;
    Temp: a temporary set initialized to ∅;
    while (C ≠ ∅) {
        if (C == {C_1}) {
            Attach C_1 to R;
            C = C − {C_1};
        } else {
            if (aff(C_1, C_2) == 0) {
                Attach C_1 to R;
                C = C − {C_1};
            } else {
                Temp = {C_1, C_2};
                Expand(C − Temp, Temp, aff(C_1, C_2));
                Create a new node N;
                NewC = NewC ∪ {N};
                w = |Temp|; /* (Temp = {C_1, C_2, ..., C_w}) */
                Assign F'(C_1) ∩ F'(C_2) ∩ ... ∩ F'(C_w) into N;
                Assign [t_{s_{C_1}}, t_{f_{C_w}}] into N;
            } /* end of if */
        } /* end of if */
    } /* end of while */
    if (NewC ≠ ∅)
        GMCO(NewC, SO, R);
}
```

Figure 5: Greedy Maximum Common Objects Algorithm

```
Expand(C, T, AFF)
C = {C_{r+1}, ..., C_m}: an ordered clip set;
T = {C_1, ..., C_r}: a new ordered clip set and r ≥ 2;
AFF: affinity of T (integer constant);
{
    if (C == ∅) return;
    if (aff(C_1, ..., C_r, C_{r+1}) ≥ AFF)
        Expand(C − {C_{r+1}}, T ∪ {C_{r+1}}, AFF);
    else
        return;
}
```

Figure 6: Expanding Subroutine

shown in Figure 6 expands a common object set T by selecting more elements from the common object set C as long as the affinity is not smaller than an integer value *AFF*. Here, sets C and T are disjoint while the initial value of *AFF* is the affinity of the first two clips. The correctness of the subroutine **Expand** is guaranteed by Theorem 2, i.e., the monotonically non-increasing nature of the clip affinity. Finally, algorithm **GMCO** recursively builds the tree by adding another level. The CVOT tree in Figure 4 is built from Example 5 by algorithm **GMCO**. In Figure 4, node C_1 has time interval $[1, 2]$ and a set of salient objects $\{\mathsf{john}, \mathsf{ball}, \mathsf{bat1}\}$; node C_2 has time interval $[3, 3]$ and a set of salient objects $\{\mathsf{john}, \mathsf{mary}, \mathsf{ball}, \mathsf{bat1}\}$; node C_3 has time interval $[4, 6]$ and a set of salient objects $\{\mathsf{ken}, \mathsf{ball}, \mathsf{bat2}\}$; node C_5 has time interval $[12, 12]$ and a set of salient objects $\{\mathsf{john}, \mathsf{cal}\}$. The affinity of C_1 and C_2 is three while the affinity will be one if C_3 is added. Therefore, C_1 and C_2 should have a parent node N_2 with a time interval $[1, 3]$ and a salient object set $\{\mathsf{john}, \mathsf{ball}, \mathsf{bat1}\}$. Similarly, node N_3 has time interval $[4, 11]$ and a set of salient objects $\{\mathsf{ken}, \mathsf{ball}, \mathsf{bat2}\}$. Node C_5 has to be attached directly to the root node because it is the only one left in the clip set. Since the affinity of N_2 and N_3 is one, a new internal node N_1 is created as shown in Figure 4. Then N_1 is directly attached to the root node. Hence, the *Root* node has time interval $[1, 12]$ and an empty salient object set.

3 The OBMS Support

CVOT is an abstract model. To have proper database management support for continuous media, this model needs to be integrated into a data model. We choose an object model for this purpose for an obvious reason. In particular we work within the context of the TIGUKAT system [ÖPS+95]. In this section we

introduce the TIGUKAT object model and its temporal extension.

3.1 TIGUKAT Model Overview

The TIGUKAT object model [ÖPS+95] is purely *behavioral* with a *uniform* object semantics. The model is *behavioral* in the sense that all access and manipulation of objects is based on the application of behaviors to objects. The model is *uniform* in that every component of information, including its semantics, is modeled as a *first-class object* with well-defined behavior. Other typical object modeling features supported by TIGUKAT include strong object identity, abstract types, strong typing, complex objects, full encapsulation, multiple inheritance, and parametric types.

The primitive objects of the model include: *atomic entities* (reals, integers, strings, etc.); *types* for defining common features of objects; *behaviors* for specifying the semantics of operations that may be performed on objects; *functions* for specifying implementations of behaviors over types; *classes* for automatic classification of objects based on type[4]; and *collections* for supporting general heterogeneous groupings of objects. In this paper, a reference prefixed by "`T_`" refers to a type, "**C_**" to a class, "*B_*" to a behavior, and "`T_X< T_Y >`" to the type `T_X` parameterized by the type `T_Y`. For example, `T_person` refers to a type, **C_person** to its class, *B_age* to one of its behaviors and `T_collection< T_person >` to the type of collections of persons. A reference such as David, without a prefix, denotes some other application specific reference.

The primitive type system is a complete lattice with the `T_object` type as the root of the lattice and the `T_null` type as the base. `T_null` binds the lattice from the bottom. It is a subtype of every other type in the system. The access and manipulation of an object's state occurs exclusively through the application of behaviors. We clearly separate the definition of a behavior from its possible implementations (functions). The benefit of this approach is that common behaviors over different types can have a different implementation in each of the types. This provides direct support for behavior *overloading* and *late binding* of functions (implementations) to behaviors.

The model separates the definition of object characteristics (a *type*) from the mechanism for maintaining instances of a particular type (a *class*). A *type* defines behaviors and encapsulates behavior implementations and state representation for objects created using that type as a template. The behaviors defined by a type describe the *interface* to the objects of that type.

[4]Types and their extents are separate constructs in TIGUKAT.

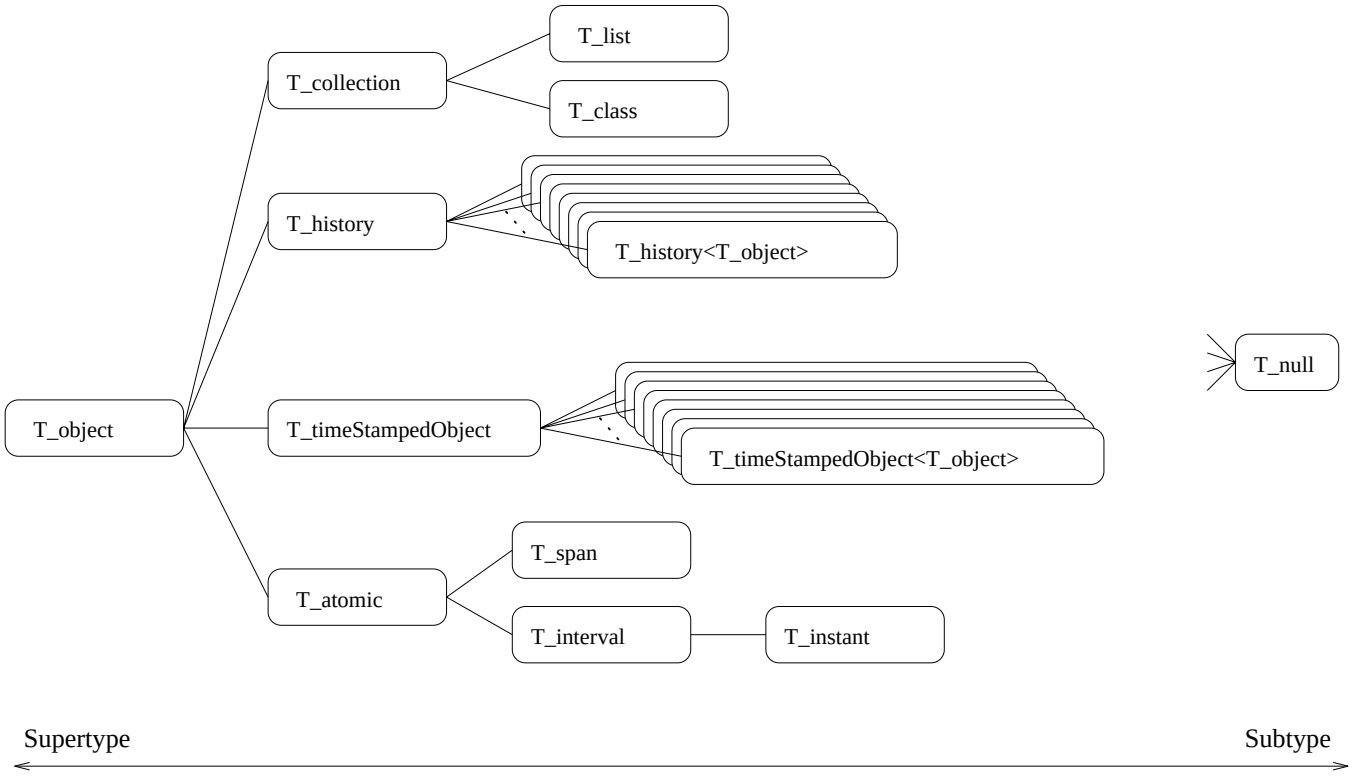

Figure 7: The Basic Time Type Hierarchy

3.2 The TIGUKAT Temporal Object Model

The TIGUKAT temporal model includes a rich and extensible set of types and behaviors to support various notions of time. This section contains a brief overview of the temporal ontology and temporal history features of this model. These features are relevant to the integration of the CVOT model described in Section 2 into TIGUKAT; we refer the reader to [GLÖS95] for a more detailed description of the temporal model. Figure 7 gives part of the time type hierarchy that includes the temporal ontology and temporal history features of the temporal model.

3.2.1 Temporal Ontology

A *time interval* is identified as the basic anchored specification of time and a wide range of operations on time intervals is provided. Unary operators which return the lower bound, upper bound and length of the time interval are defined. The model supports a rich set of ordering operations among intervals [All83] (these are depicted in Figure 8), e.g., *precedes*, *overlaps*, *during*, etc. as well as set-theoretic operations

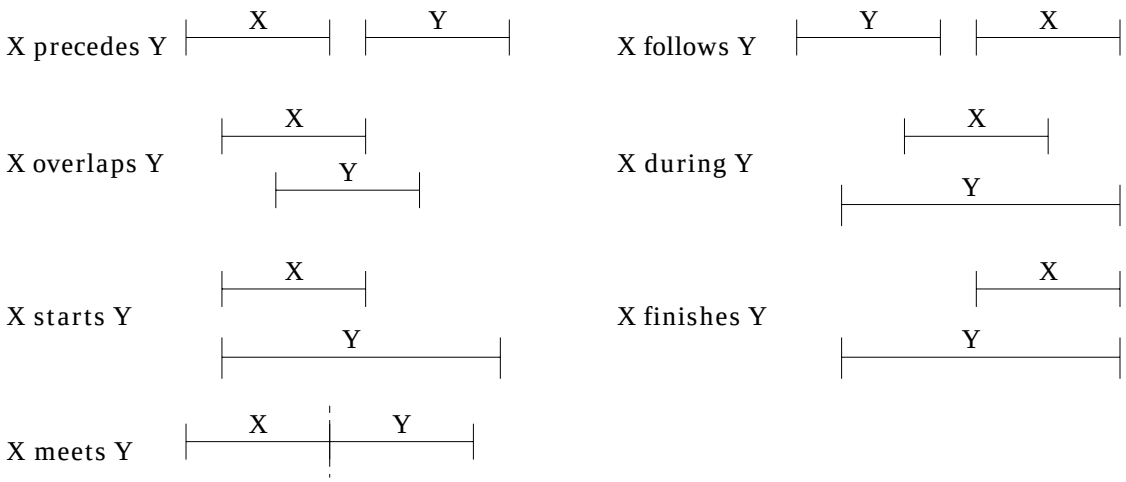

Figure 8: Different Types of Ordering Relations Between Intervals

viz *union*, *intersection* and *difference*[5]. A time duration can be added or subtracted from a time interval to return another time interval. A time interval can be expanded or shrunk by a specified time duration. Different kinds of open, closed, half open and half closed intervals are modeled.

A *time instant* (*moment*, *chronon*, etc.) is a specific anchored moment in time. A time instant is modeled as a special case of a (closed) time interval which has the same lower and upper bound, e.g., $Jan\ 24, 1996 = [Jan\ 24, 1996,\ Jan\ 24, 1996]$. A wide range of operations can be performed on time instants. A time instant can be compared with another time instant with the transitive comparison operators $<$ and $>$. A time duration can be added to or subtracted from a time instant to return another time instant. A time instant can be compared with a time interval to check if it falls before, within or after the time interval.

A *time span* is an unanchored relative duration of time. A time span is basically an atomic cardinal quantity, independent of any time instant or time interval. Time spans have a number of operations defined on them. A time span can be compared with another time span using the transitive comparison operators $<$ and $>$. A time span can be subtracted from or added to another time span to return a third time span. The detailed behavior signatures corresponding to the operations on time intervals, time instants, and time spans are given in the Table 1.

[5]Note that the union of two disjoint intervals is not an interval. Similarly, for the difference operation, if the second interval is contained in the first, the result is not an interval. In the temporal model, these cases are handled by returning an object of the *null* type (`T_null`). The `T_null` type is a subtype of all other types in the TIGUKAT type lattice, including the *interval* type (`T_interval`). Hence, every instance of `T_null` is also an instance of `T_interval`.

T_interval	*B_lb*:	T_instant
	B_ub:	T_instant
	B_length:	T_span
	B_precedes:	T_interval → T_boolean
	B_follows:	T_interval → T_boolean
	B_during:	T_interval → T_boolean
	B_meets:	T_interval → T_boolean
	B_overlaps:	T_interval → T_boolean
	B_starts:	T_interval → T_boolean
	B_finishes:	T_interval → T_boolean
	B_union:	T_interval → T_interval
	B_intersection:	T_interval → T_interval
	B_difference:	T_interval → T_interval
	B_subtract:	T_span → T_interval
	B_add:	T_span → T_interval
	B_expand:	T_span → T_interval
	B_shrink:	T_span → T_interval
T_instant	*B_lessthaneqto*:	T_instant → T_boolean
	B_greaterthaneqto:	T_instant → T_boolean
	B_elapsed:	T_instant → T_span
	B_subtract:	T_span → T_instant
	B_add:	T_span → T_instant
	B_intersection:	T_interval → T_instant
	B_difference:	T_interval → T_instant
	B_shrink:	T_span → T_instant
	B_succ:	T_instant
	B_pred:	T_instant
T_span	*B_lessthan*:	T_span → T_boolean
	B_greaterthan:	T_span → T_boolean
	B_lessthaneqto:	T_span → T_boolean
	B_greaterthaneqto:	T_span → T_boolean
	B_add:	T_span → T_span
	B_subtract:	T_span → T_span
	B_succ:	T_span
	B_pred:	T_span

Table 1: Behaviors on time intervals and time instants

3.2.2 Temporal Histories

One requirement of a temporal model is an ability to adequately represent and manage histories of objects and real-world events. Our model represents the temporal histories of objects whose type is `T_X` as objects of the `T_history<T_X>` type as shown in Figure 7. A temporal history consists of objects and their associated timestamps (time intervals or time instants). A *timestamped object* knows its timestamp and its associated object (value) at (during) the timestamp. A temporal history is made up of such objects. Table 2 gives the behaviors defined on histories and timestamped objects. Behavior *B_history* defined on `T_history<T_X>` returns the set (collection) of all timestamped objects that comprise the history. Another behavior defined on history objects, *B_insert*, timestamps and inserts an object in the history. The *B_validObjects* behavior allows the user to get the objects in the history that were valid at (during) the given time.

`T_history<T_X>`	*B_history*:	`T_collection<T_timeStampedObject<T_X>>`
	B_insert:	`T_X,T_interval` → `T_boolean`
	B_validObjects:	`T_interval` → `T_collection<T_timeStampedObject<T_X>>`
`T_timeStampedObject<T_X>`	*B_value*:	`T_X`
	B_timeStamp:	`T_interval`

Table 2: Behaviors on histories and time-stamped objects

Each timestamped object is an instance of the `T_timeStampedObject<T_X>` type. This type represents objects and their corresponding timestamps. Behaviors *B_value* and *B_timeStamp* defined on `T_timeStampedObject` return the value and the timestamp of a timestamped object, respectively.

4 System Integration

Integrated multimedia systems can result in a uniform object model, simplified system support and possibly better performance. In such a system, the multimedia component can directly use many functions provided by the OBMS, such as concurrency control, data recovery, access control etc. In this section we discuss the integration of the CVOT model into an OBMS as well as the type hierarchy and behavior definitions of video data. We explain why temporal histories are used to model the various features of the CVOT model and the contents of a video. Further, we show how to construct powerful multimedia queries using the behaviors defined on time instants, intervals and spans. Figure 9 shows our proposed video type system. The types that are in a grey shade are directly related to our video model and they will be discussed in detail throughout this section.

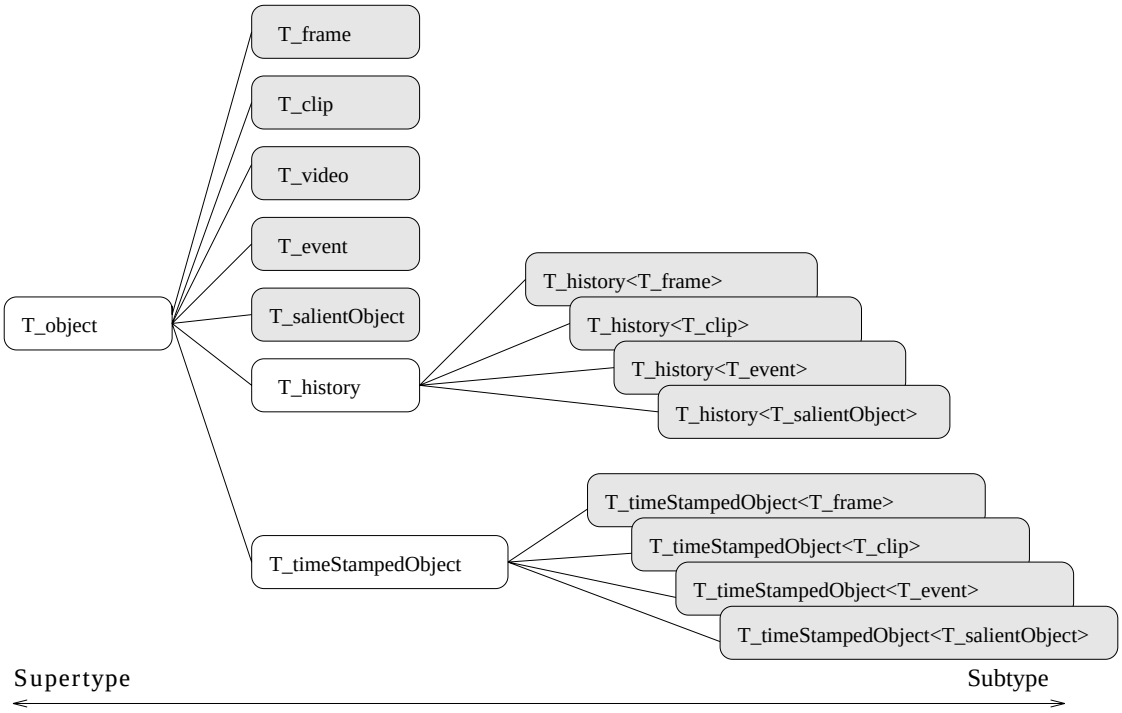

Figure 9: The Video Type System

4.1 Integrating the CVOT Model

We start by defining the `T_video` type to model videos. An instance of `T_video` has all the semantics of a video. As we saw in Section 2, a video is segmented into a set of clips. Since a clip set is ordered and each clip has an associated time interval, it is natural to model this set as a history. We model a clip set by defining the behavior *B_clips* in `T_video`. *B_clips* returns a history object of type `T_history< T_clip >`, the elements of which are timestamped objects of type `T_clip`.

Example 3 Suppose myVideo is an instance (object) of `T_video`. Then,

- myVideo.*B_clips* returns an instance (object) of type `T_history< T_clip >`. Let this object be myVideoClipHistory.
- myVideoClipHistory.*B_history* returns a collection (clip set) which contains all the timestamped clip objects of type `T_timeStampedObject< T_clip >` in myVideo. Let one of these clip history object be myVideoCHOneClip.
- myVideoCHOneClip.*B_value* returns the content of myVideoCHOneClip, while myVideoCHOneClip.*B_timeStamp* returns the time interval of myVideoCHOneClip.

T_video		
	B_clips:	T_history<T_clip>
	B_cvotTree:	T_tree
	B_search:	T_salientObject, T_tree → T_tree
	B_softlyOrdered:	T_boolean
	B_perfectlyOrdered:	T_boolean
	B_stronglyOrdered:	T_boolean
	B_length:	T_span
	B_publisher:	T_collection<T_company>
	B_producer:	T_collection<T_person>
	B_date:	T_instant
	B_play:	T_boolean

Table 3: Behavior Signatures of Videos

Table 3 gives the behavior signatures of videos.

The behavior *B_cvotTree*[6] returns the common video object tree of a video. For example, myVideo.*B_cvotTree* creates a CVOT tree from the clip set of myVideo. Our implementations of *B_cvotTree* is the GMCO algorithm discussed in Section 2.3. *B_search* searches a CVOT tree and returns a subtree which contains a salient object. The returned subtree can be one of the following:

- no nodes (empty subtree): the object does not appear in any clip of this video;
- one leaf node: the object appears in one clip, but not in its neighbors;
- both leaf nodes and internal nodes: the object appears in multiple clips.

In the CVOT model, a video knows the ordering of its clips. This ordering is defined by several video behaviors: *B_softlyOrdered*, *B_perfectlyOrdered* and *B_stronglyOrdered* which simply iterate over the time intervals in the clip set history and determine whether a clip history is softly ordered, perfectly ordered, or strongly ordered respectively.

A common question to myVideo would be its length (duration). This is modeled by the *B_length* behavior and it returns an object of type T_span. If a video is segmented into a perfectly ordered clip set, its length is equal to the total length of all the clips in this set. However, if the clip set is softly ordered or strongly ordered, the video length is not equal to the total length of all the clips because in such clip sets, clips may overlap.

Video information should also include metadata, such as the publishers, producers, publishing date, etc. A video can also be played by using *B_play*.

Each clip has a set of consecutive frames, which is modeled by T_history<T_frame>. Since a clip must be associated with a time interval, we treat clips as timestamped objects. Suppose myClip is a particular

[6] We assume the existence of type T_tree in TIGUKAT. Actually it is not difficult to define T_tree using TIGUKAT primitive types.

clip, then myClip is an instance (object) of type T_timeStampedObject< T_clip >. The content of myClip is myClip.*B_value* while the interval of myClip is myClip.*B_timeStamp*. Some behavior signatures of clips are shown in Table 4.

T_clip		
	B_frames:	T_history< T_frame >
	B_salientObjects:	T_collection<T_history<T_salientObject>>
	B_events:	T_collection<T_history<T_event>>
	B_affinity:	T_list<T_clip> → T_integer
	B_play:	T_boolean

Table 4: Behavior Signatures of Clips

All the salient objects within a clip are grouped by the behavior *B_salientObjects* which returns an instance of T_collection< T_history < T_salientObjects >>. Since a salient object, say john, can appear several times within a clip, such distinct appearances must be captured within the system. However, nothing about john has changed except the time interval. Therefore, history is a natural method to model this behavior.

It is legitimate to ask a clip's affinity (*B_affinity*) with other clips. *B_play* of T_clip is able to play a clip on an appropriate output device. Other related operations, such as *stop, pause, play backward*, etc., are omitted from the table because they are not important to our discussions. Such omissions are also applicable to the behaviors of T_video.

The basic building unit of a clip is the frame which is modeled by T_frame in Table 5. A frame knows its location within a clip or a video and such a location is modeled by a time instant (*B_location*), which can be a relative frame number. We model frames within a clip as a history which is identical to how we model clips within a video.

T_frame		
	B_location:	T_instant
	B_type:	T_videoType
	B_content:	T_image

Table 5: Behavior Signatures of Frames

Many different types of frames may exist, e.g. predicted frames, intracoded frames and bidirectional frames in MPEG videos [Gal91]. This is defined by the behavior *B_type* of T_frame. We assume the existence of a video type T_videoType. The content of a frame, *B_content*, is an image which defines many image properties such as width, height and color.

4.2 Modeling Video Features

The semantics or contents of a video is usually expressed by its *features* which include video attributes and the relationships between these attributes. Typical video features are salient objects and *events*. An *event* is a kind of activity which may involve many different salient objects over a time period, like holding a part, playing table tennis, and chatting with someone. Since we have discussed frames, clips and videos in the previous subsection, in this subsection we focus on salient objects and event definitions.

An event can occur in different places either within a clip or crossing multiple clips. For example, the event johnDrive may occur in multiple clips. Additionally, this event may occur several times within a clip. Therefore, an appropriate representation is necessary to capture the temporal semantics of general events. A simple and natural way to model the temporal behavior of events is to use historical structure. Thus, we model histories of events as objects of type `T_history< T_event >`. Instances, such as johnDrive, of `T_history< T_event >` consist of timestamped events. This allows us to keep track of all the events occurring within a video although an event may occur in multiple clips or just occur within one clip. In the interest of tracking all the events occurring within a clip, the behavior *B_events* is included in `T_clip`.

Similarly, since salient objects can also appear multiple times in a clip or a video, we model the history of a salient object as timestamped object of type `T_history< T_salientObject >`. The behavior *B_salientObjects* of `T_clip` returns all the salient objects within a clip. Using histories to model salient objects and events result in powerful queries as will be shown in the next subsection. Furthermore, it enables us to uniformly capture the temporal semantics of video data because a video is modeled as a history of clips and a clip is modeled as a history of frames.

`T_event`	*B_activity*:	`T_eventType`
	B_roles:	`T_collection<T_person>`
	B_inClips:	`T_video → T_history< T_clip >`
	B_eventObjects:	`T_collection<T_salientObject>`
`T_salientObject`	*B_boundingBox*:	`T_history< T_boundingBox >`
	B_centroid:	`T_point`
	B_inClips:	`T_video → T_history< T_clip >`
	B_status:	`T_statusType`

Table 6: Some Behavior Signatures of Events and Salient Objects

The behavior *B_activity* of `T_event`, shown in Table 6, identifies the type of events `T_eventType` and the behavior *B_roles* indicates all the persons involved in an event. *B_eventObjects* returns all the salient objects within an event. *B_inClips* indicates all the clips in which this event occurs. It is certainly reasonable to include other information, such as the location and time of an event, into type `T_event`, but they are not important to our discussion.

The behavior *B_boundingBox* of type `T_salientObject` defines a bounding box on an object so its spatial information can be recorded. The bounding box values of a salient object may change as the time goes. Again a history is excellent to capture such behaviors. *B_centroid* returns the centroid point of a salient object. The behavior *B_inClips* returns all the clips in which the salient object appears. This corresponds to the reverse function $\mathcal{F}'$ defined in Section 2.2. *B_status* may be used to define some other attributes of an salient object. For example, it is very useful to know whether an object is *rigid* or not if we want to track the motion of the object. Here we assume that `T_statusType` is such an enumerated type.

4.3 Applications of the CVOT model

A very brief discussion of how our CVOT model could be used in a real application is presented here. Figure 10 shows a video hierarchy. We let `T_playEvent` and `T_driveEvent` be the types of all objects that represent particular play and drive events respectively. For example, johnPlay (John plays table tennis) could be one of the objects of type `T_playEvent` while johnDrive could be one of the objects of type `T_driveEvent`. The class **C_event** maintains all the objects of type `T_event`. johnPlay and johnDrive are the objects of this class.

Now we explain how a driving event could be described in the CVOT model. The behavior *B_image* of `T_person` (see Table 7) returns the image of a person, which is useful in identifying a person. *B_driving* of `T_driveEvent` returns true if the driver is driving, otherwise it returns false. This behavior could be handled as follows: first we require the driver to be sitting at the driver seat, which can be done by checking the driver's bounding box against the driver seat (*B_driverSeat*) over a period time, making sure the driver is always within the vehicle (*B_transport*). If there exists any static object, checking the distance between the vehicle and the static object will determine the driving event; if there is no static object, we have to evaluate the distance of the vehicle over a period time and have this distance be greater than a predefined threshold value.

4.4 Query Examples

In this subsection we present some examples to show the expressiveness of our model. Since we are using TIGUKAT object calculus [Pet94], a brief introduction to it is necessary. The alphabet of the calculus consists of object constants (a, b, c, d), object variables (o, p, q, u, v, x, y, z), monadic predicates (C, P, Q), dyadic predicates $(=, \in, \notin)$, an n-ary predicate $(Eval)$, a function symbol (β) called *behavior specification* (Bspec), and logical connectives $(\exists, \forall, \wedge, \vee, \neg)$. The "evaluation" of a Bspec is accomplished by predicate

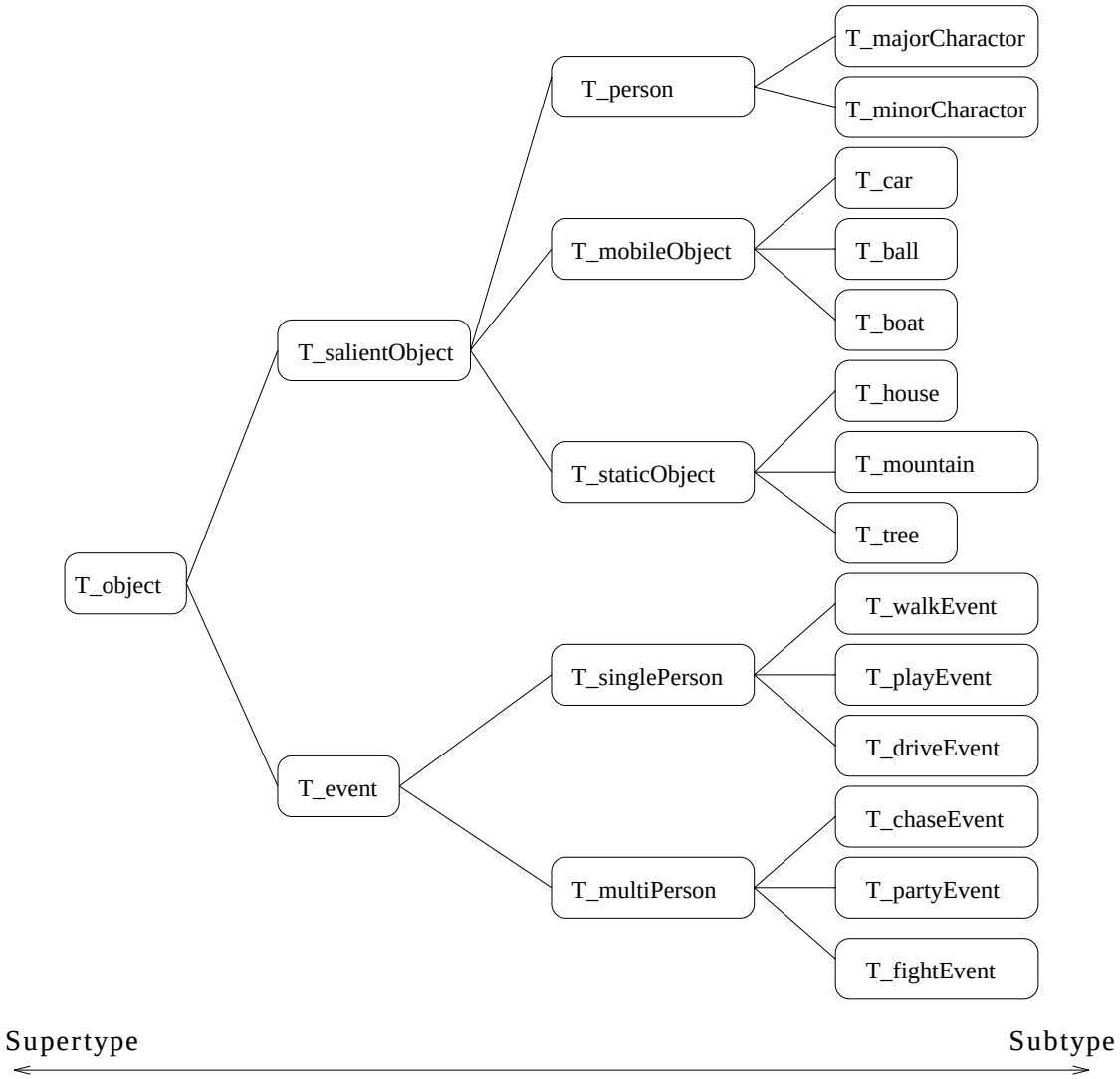

Figure 10: Video Hierarchy From Viewers' Point of View

T_person	*B_name*:	T_string
	B_birthDate:	T_date
	B_address:	T_string
	B_image:	T_image
T_driveEvent	*B_driver*:	T_person
	B_transport:	T_mobileObject
	B_driverSeat:	T_staticObject
	B_driving:	T_boolean
T_playEvent	*B_player*:	T_person
	B_playee:	T_collection<T_person>
	B_playing:	T_boolean

Table 7: Simple Behavior Signatures of Some Video Objects

Eval. A *term* is an object constant, an object variable or a Bspec. An *atomic formula* or *atom* has an equivalent Bspec representation. From atoms, *well-formed formulas* (WFFs) are built to construct the declarative calculus expressions of the language. WFFs are defined recursively from atoms in the usual way using the connectives $\wedge, \vee, \neg$ and the quantifiers $\exists$ and $\forall$.

A query is an object calculus expression of the form $\{t_1, \ldots, t_n | \phi(o_1, \ldots, o_n)\}$ where $t_1, \ldots, t_n$ are the terms over the multiple variables $o_1, \ldots, o_n$. ϕ is a WFF. Indexed object variables are of the form $o[\beta]$ where β is a set of behaviors defined on the type variable o. The semantics of this construct is to project over the behaviors in β for o, meaning that after the operation only the behaviors given in β will be applicable to o. A detailed description of the TIGUKAT object calculus is found in [Pet94].

We assume that all the queries are posted to a particular video instance myVideo and also salient objects and events are timestamped objects as discussed in Section 4. We also assume that all clips are timestamped clips and c $\in$ myVideo.*B_clips*.*B_history* where c is an arbitrary clip. myVideo.*B_clips* returns a history of all the clips in myVideo and myVideo.*B_clips*.*B_history* returns a collection of all the timestamped clips in myVideo. Since c is a timestamped clip, c belongs to the class **C_timeStampedObject** and the type of c is `T_timeStampedObject` < `T_clip` >. Since all the clips, salient objects, events belong to timestamped object class **C_timeStampedObject**, we omit them in the query calculus expressions.

Query 1 Our first query is to ask the duration of a clip c. It is simply c.*B_timeStamp*.*B_length*. Similarly, the duration of salient object a (or an event e) is a.*B_timeStamp*.*B_length*
(or e.*B_timeStamp*.*B_length*).

Query 2 This query asks whether a salient object is in a clip. For a given object a and clip c it could be expressed in TIGUKAT object calculus as:

$\{q \mid q = a.B\_timeStamp.B\_during(c.B\_timeStamp)\}$.

The query checks whether the time interval of object a is a subinterval of clip c. Another way to express the same query is to use clips associated with a:

$\{o \mid o = a.B\_value.B\_inClips(\text{myVideo}).B\_history.B\_elementOf(c)\}$.

Here, a.*B_value* returns the salient object a which indicates a.*B_value* $\in$ **C_salientObject**. Also a.*B_value*.*B_inClips*(myVideo) returns a history of all the clips containing a. Applying *B_history* to it returns the collection (set) of these clips. The behavior *B_elementOf*(c), defined in `T_collection`, checks whether c is an element of the collection.

For convenience, predicate *IN*(o, v) is used to denote that object o is in clip v. Similarly for a given event p, *INevent*(p, v) is the predicate denoting whether event e is in clip v.

Query 3 The query to find all the clips, in which *John* appears, could be either

$$\{\mathsf{v} \mid \exists \mathsf{p}(\mathsf{p}.B\_value.B\_name = \text{`}John\text{'} \wedge \mathsf{v} = \mathsf{p}.B\_value.B\_inClips.B\_history)\}$$

or

$$\{\mathsf{v} \mid \forall \mathsf{w}(\mathsf{p}.B\_value.B\_name = \text{`}John\text{'} \wedge IN(\mathsf{p}, \mathsf{w}) \wedge \mathsf{v} = \mathsf{w})\}$$

where p is an instance of timestamped T_person.

Query 4 To find the last clip in which person p appears:

$$\{v \mid \forall u(u \neq v \wedge IN(p, u) \wedge IN(p, v) \wedge u.B\_timeStamp.B\_precedes(v.B\_timeStamp))\}$$

where u and v are different clips. We compare time stamps of all the clips, in which p appears, with each other and choose the one which all others precedes it.

Query 5 "*In which clip does a play event last the longest?*":

$$\{v \mid \forall \mathsf{s} \exists \mathsf{w}(\mathsf{s} \neq \mathsf{w} \wedge \mathsf{w}.B\_timeStamp.B\_length.B\_greaterthaneqto(\mathsf{s}.B\_timeStamp.B\_length) \wedge IN(\mathsf{w}, \mathsf{v}))\}.$$

Suppose w is the longest play event which occurs in clip v, then w must satisfy the condition: the duration (*B_length*) of w's interval (w.*B_timeStamp*) is greater than or equal to (*B_greaterthaneqto*) the duration of any other play events. s and w are instances of type T_timeStampedObject < T_playEvent >.

Query 6 "*Are there any two clips in which object* x *simultaneously appears?*":

$$\{\mathsf{u}, \mathsf{v} \mid IN(\mathsf{x}, \mathsf{u}) \wedge IN(\mathsf{x}, \mathsf{v}) \wedge \mathsf{u} \neq \mathsf{v} \wedge \mathsf{x}.B\_timeStamp.B\_during(\mathsf{u}.B\_timeStamp.B\_intersection(\mathsf{v}.B\_timeStamp))\}.$$

The tricky part of this query is in finding an overlap part of two neighboring clips. The temporal intersection operation *B_intersection* is perfect to accomplish this operation. Of course, object x must be within such an overlap.

Query 7 "*Find a video clip in which* John *is driving a car after he walked out of the table-tennis room*":

$$\{\mathsf{u} \mid \exists \mathsf{x} \exists \mathsf{y}(\mathsf{p}.B\_value.B\_name = \text{`}John\text{'} \wedge \mathsf{x}.B\_value.B\_driver = \mathsf{p}.B\_value \wedge IN(\mathsf{p}, \mathsf{u}) \wedge \mathsf{y}.B\_value.B\_walker = \mathsf{p} \wedge \mathsf{y}.B\_value.B\_walkFrom(\mathsf{z}) \wedge \mathsf{x}.B\_value.B\_driving \wedge IN(\mathsf{x}, \mathsf{u}) \wedge INevent(\mathsf{y}, \mathsf{u}) \wedge \mathsf{x}.B\_timeStamp.B\_meet(\mathsf{y}.B\_timeStamp))\}$$

where p is an instance of timestamped T_person, x is an instance of timestamped T_driveEvent, y is an instance of timestamped T_walkEvent with one more behavior *B_walkFrom*, and z is an instance of timestamped T_room. Particularly object z represents table-tennis room. *B_walkFrom* describes a walker walking out from some place. The behavior *B_meet* at here guarantees that drive event occurs right after walk event.

5 Related Work

There is significant current interest in modeling video systems. Gibbs et. al. [GBT94] investigate timed streams as the basic abstraction for modeling temporal media using object-oriented technology. The *media element* in their model corresponds to video frames in ours. A *timed stream* is modeled by a finite sequence of tuples $< e_i, s_i, d_i >$, $i = 1, \ldots, n$, where e_i is a media element, s_i is the start time of e_i and d_i is its duration. Three general structuring mechanisms (interpretation, derivation, and composition) are used to model time-based media. We also model videos as time-based streams. However, the temporal operations are not well supported in their model.

AVIS (Advanced Video Information System) [ACC+95], which uses *association maps* to group salient objects and events, is quite close to our model. A video stream is segmented into a set of frame-sequences $[x, y)$, where x is the start frame and y is the end frame. Based on the association maps, a *frame segment tree* is built to capture objects and events occurring in the frame-sequences. Then two arrays are created: objectArray and eventArray. Each element of any array is an ordered linked list of pointers to nodes in the frame segment tree. It is shown that such a data structure results in efficient query retrieval. Although AVIS model is similar to CVOT, there are some fundamental differences:

- In CVOT, the segmentation of a video can be arbitrary in the sense that two neighboring clips could overlap as long as they are either softly or perfectly ordered. However, in AVIS two neighboring clips must be consecutive, i.e., they must be perfectly ordered. This extension in the CVOT model is important because an event may across multiple clips.

- Frame segment tree is a binary tree and, in practice, it is made up of many *empty nodes* (node without any common objects or events from its child nodes). This problem could result in deep binary trees. In CVOT, the tree is an arbitrary tree and only the root node's common object set is allowed to be empty. The major advantage of such a shallow tree is its small number of nodes which can result in significant space saving. The tradeoff could be the building cost of an arbitrary tree usually higher than that of a binary tree and more complex searching algorithms.

- We disallow events to be modeled in CVOT tree whereas AVIS allows their representation. The argument is that an event may cross multiple clips. This is particularly important if a video is segmented by a fixed time interval which is actually used in AVIS prototype system.

Video Semantic Directed Graph (VSDG) is a graph-based conceptual video model [DDI+95]. The most important feature of the VSDG model is an unbiased representation of the information while providing

a reference framework for constructing semantically heterogeneous user's view of the video data. Using this model along with the object-oriented hierarchy, a new video system architecture is proposed which can automatically handle video data. The video semantic directed graphs are more complicated than our common video object trees without introducing any more capability. Furthermore, the VSDG model does not directly support range queries, such as "*Find all the clips in which John appears*".

Little and Ghafoor [LG93] have described a temporal model to capture the timing relationships between objects in composite multimedia objects, and mapped it to a relational database. This model forms a basis for a hierarchical data model and for temporal access control algorithms to allow VCR-like capabilities. A generalized *n-ary* structure is used to model spatial-temporal semantics of multimedia data.

OVID (Object-oriented Video Information Database) [OT93] is an object-oriented video model. It introduces the notion of a *video object* which can identify an arbitrary video frame sequence (a meaningful scene) as an independent object and describe its contents in a dynamic and incremental way. However, the OVID model has no schema and the traditional class hierarchy of OBMSs is not assumed. An inheritance based on an *interval inclusion relationship* is introduced to share descriptional data among video objects. A major problem with OVID model is its heavy dependence on the *video description* which has to be done manually. We borrow the idea of modeling salient objects and events by the object-oriented technology from OVID model, then integrate multimedia temporal operations into this object-oriented model.

An architecture, called ViMod, for a video objectbase based on video features is proposed in [JH94]. The design of this model is the result of studying the metadata characteristics of queries and video features. The *algebraic video* data model [WDG94] allows users to model nested video structures such as shots, scenes and sequences and to define the output characteristics of video segments. A quite comprehensive set of temporal operators has been defined within the algebraic video system. Other video models include the OMEGA [Mas91], the motion-based semantic video [DG94], and the video-on-demand [LAF+93].

6 Conclusions

In this paper, a tree-based video model, called CVOT model, is proposed for specifying both the spatial and temporal semantics of video data although we only concentrate on temporal issues. The major advantages of the CVOT model are the flexibility for video segmentation and the feasibility of automatic video feature extraction. A unique way of integrating the CVOT model into an OBMS with rich temporal operations is presented. A new uniform approach of modeling video medium using histories is also introduced. End users are allowed to explore the video objectbase from their perception of video contents through the

object-oriented technology. Such a seamless integration brings a uniform interface to end users. The integrated video objectbase management system is extensible so any new technology can be easily added into the system. Furthermore, we show that our system supports a broad range of temporal queries and the combination of the CVOT model and object-oriented technology results in an elegant video OBMS.

There are two major directions for our future work on the CVOT model. One is to specify spatial relationships within the CVOT model, for which we are currently designing a spatial inference engine. The combination of the spatial and temporal relationships within a single model adds significant power and enables spatio-temporal reasoning. Another future work is to build a video query language based on the CVOT model. The queries can be translated into the TIGUKAT query calculus and then query algebra. Therefore, it is possible to optimize these queries using object query optimization techniques [MDZ93, ÖB95].

References

[ACC+95] S. Adali, K. S. Candan, S. S. Chen, K. Erol, and V. S. Subrahmanian. Advanced video information system: Data structures and query processing. In *ACM-Springer Multimedia Systems Journal (in press)*, 1995.

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

[CK95] S. Christodoulakis and L. Koveos. Multimedia information systems: Issues and approaches. In W. Kim, editor, *Modern Database Systems*, pages 318—337. Addison-Wesley, 1995.

[DDI+95] Y. F. Day, S. Dagtas, M. Iino, A Khokhar, and A. Ghafoor. Object-oriented conceptual modeling of video data. In *Proceedings of the 11th International Conference on Data Engineering*, pages 401—408, Taipei, Taiwan, 1995.

[DG94] N. Dimitrova and F. Golshani. Rx for semantic video database retrieval. In *Proceedings of the 2nd ACM International Conference on Multimedia*, pages 219—226, San Francisco, CA, October 1994.

[Gal91] D. L. Gall. MPEG: A video compression standard for multimedia applications. *Communications of ACM*, 34(4):46—58, 1991.

[GBT94] S. Gibbs, C. Breiteneder, and D. Tscichritzis. Data modeling of time-based media. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 91—102, Minneapolis, Minnesota, May 1994.

[Gha96] S. Ghandeharizadeh. Stream-based versus structured video objects: Issues, solutions, and challenges. In V. S. Subrahmanian and S. Jajodia, editors, *Multimedia Database Systems: Issues and Research Directions*, pages 215—236. Springer Verlag, 1996.

[GLÖS95] I. A. Goralwalla, Y. Leontiev, M. T. Özsu, and D. Szafron. A uniform behavioral temporal object model. Technical Report TR-95-13, Department of Computing Science, University of Alberta, May 1995.

[HJW95] A. Hampapur, R. Jain, and T. E. Weymouth. Production model based digital video segmentation. *Multimedia Tools and Applications*, 1(1):9—46, March 1995.

[JH94] R. Jain and A. Hampapur. Metadata in video database. *ACM SIGMOD RECORD*, 23(4):27—33, December 1994.

[Kim95] W. Kim. Multimedia information systems: Issues and approaches. In W. Kim, editor, *Modern Database Systems*, pages 5—17. Addison-Wesley, 1995.

[LAF+93] T. D. C. Little, G. Ahanger, R. J. Folz, J. F. Gibbon, F. W. Reeve, D. H. Schelleng, and D. Venkatesh. Digital on-demand video service supporting content-based queries. In *Proceedings of the First ACM International Conference on Multimedia*, pages 427—436, Anaheim, CA, 1993.

[LG93] T. C. C. Little and A. Ghafoor. Interval-based conceptual models for time-dependent multimedia data. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):551—563, August 1993.

[Mas91] Y. Masunaga. Design issues of OMEGA: An object-oriented multimedia database management system. *Journal of Information Processing*, 14(1):60—74, 1991.

[MDZ93] G. Mitchell, U. Dayal, and S. B. Zdonik. Control of an extensible query optimizer: A planning-based approach. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 517—528, Dublin, Ireland, August 1993.

[ÖB95] M.T. Özsu and J. Blakeley. Query optimization and processing in object-oriented database systems. In W. Kim, editor, *Modern Database Systems*, pages 146—174. Addison-Wesley, 1995.

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

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

[Pet94] R. Peters. TIGUKAT: A uniform behavioral objectbase management system. PhD thesis, Department of Computing Science, University of Alberta. Available as Technical Report TR-94-06, 1994.

[SZ94] S. W. Smoliar and H. J. Zhang. Multimedia information systems. *IEEE Multimedia Systems*, 1(2):62—72, Summer 1994.

[WDG94] R. Weiss, A. Duda, and D. K. Gifford. Content-based access to algebraic video. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 140—151, 1994.

[WK87] D. Woelk and W. Kim. Multimedia information management in an object-oriented database system. In *Proceedings of the 13th International Conference on Very Large Data Bases*, pages 319—329, Brighton, England, September 1987.