

Model Video Semantics with Constraints Considering Temporal Structure and Typed Events*

Yu Wang , Lizhu Zhou, Jianyong Wang

Tsinghua University, Beijing, China

wangyu02@mails.tsinghua.edu.cn, {[dcszlj](mailto:dcszlj@tsinghua.edu.cn), [jianyong](mailto:jianyong@tsinghua.edu.cn)}@tsinghua.edu.cn

Abstract

The advances of video technology and video-related applications demand appropriate video semantic models for representing video data and their semantics, and supporting powerful semantic queries on them. In this paper, we propose such a model named SemTTE. The model incorporates features of temporal structure and typed events of video contents. It organizes the whole video into a tree of events, and provides mechanisms for users to define domain-specific constraints. As a result, the contents and semantics of the video can be better represented and queried. For constraints enforcement, an efficient on-line method is proposed.

1. Introduction

The advances in video technology and the expressive and intuitive features of videos make more and more video data become an integral part of information systems. To manage those data and their semantics so users can find what they need efficiently and accurately, an appropriate semantic data model is needed.

However, most existing video semantic models simply view video data as a sequence of mutually independent segments with related annotations without exploring semantic relations between segments. No mechanism is provided to define and later check the mappings between semantic information and video segments, such as what events may occur in what part of a video, or what the order of events should be. However, this information is of great value for information validation. To overcome this weakness, we propose a model named SemTTE.

In SemTTE, these semantic considerations are concluded into two properties characterized by typed events and temporal structures, described as follows:

- ◆ Typed events: events in video can be abstracted to a finite set of types and the video can be viewed as a sequence of instances of these types. For example, “Foul” can be an event type in a NBA game and “O’Neal fouled YaoMing” is an instance of the type.
- ◆ Temporal structures: event instances are ordered temporally from the beginning to the end of a video, each one is related to a video segment to indicate its

occurrence. Constraints can be defined on the ordering and relationship of event instances of various types, for example, in a NBA game, there must be a first halftime followed by a second halftime, and the start score of the second halftime must equal the end score of the first halftime.

This paper is structured as follows. Section 2 provides a review of related work. Section 3 introduces SemTTE with its framework, schema, instance, and constraint mechanism. Section 4 is the conclusion. Due to the space limitation, this paper will only focus on the model framework and constraint mechanism.

2. Related Work

Although many models [1-10] have been proposed to manage video semantics, few of them has considered the two properties mentioned above as we do. [11] is a review and evaluation of these models.

Typed events and ISA relationship between types are considered in almost all models [1,2,4,6-10]. However, these event types are only used to distinguish event instances with different properties, not related to the temporal structure of video. For VideoGraph[3] and CoPaV²[5], no schema is needed, so new types can be added at any time. Schemaless makes them more flexible at the price of lacking of ability to manage types.

Videx[1], Extended ExIFO₂[7], and Temporal OO[4] have considered the temporal structure of video and partition videos into segments. However, these segments are just physical partitions, not related to semantics and no connection between partitions is explored. Videx, VideoGraph, Temporal OO, Ahmet Ekin’s model[9] and THVDM[10] represent events’ temporal relationships explicitly. However, these relationships are only optional features and no mechanism is provided to define constraints for them, and neither connections between event types nor video’s temporal structures are explored.

In summary, although temporal relationship between events has been considered in some existing models, they are only relationships between individual instances, not abstracted to event types and no mechanism is provided to constraint the temporal relationship between instances of various types in a specific domain.

* Supported by the National Natural Science Foundation of China under Grant No. 60473078

Compared to existing video semantic models, the distinguishing properties of SemTTE are:

- ◆ Temporal relationships between event instances are explicitly represented and mandatory. These relationships are abstracted to event types to represent the temporal structure of video.
- ◆ Mechanisms are provided for users to define constraints for temporal structure of videos in a specific domain, limiting the values, ordering, and relationships of event instances of various types. Besides, an efficient method for on-line constraint enforcement is proposed.

3. SemTTE

3.1 Framework of SemTTE

In SemTTE, information is represented by means of three primitive concepts:

- ◆ Entities: for perceivable objects or abstract objects appearing in or related to the video;
- ◆ Events: for video segments representing relationships between entities that occur in the video. Each event is located in the video by a temporal location. An event may be composed of other events;
- ◆ Entity and Event Types: entities and events sharing common properties are generalized by entity types and event types with a set of attributes (or roles) representing the common properties.

Besides these concepts, constraint mechanisms are provided for users to define constraints considering typed events and temporal structure for a specific domain, which will be described in more details in section 3.4.

3.2 Schema

The schema of SemTTE is composed of two sets of types for entities and events respectively. Type definitions are as following:

```
EntityType ::= (Name[,attributes: {Attribute}]
               [,constraints:{Intra-Constraint}])
EventType ::= (Name[,attributes: {Attribute}]
               [,roles: {Role}][,constraints: {Intra-Constraint}]
               [,components: {EventType}])
```

For both entity type and event type, constraints may be defined to limit values or relationships of attributes (or roles for event type).

3.3 Instance

Database instance of SemTTE contains two parts: a set of entities and a set of event trees.

```
Instance ::= ({EntityType}, {EventTree})
EventInstance ::= (TypeName {, AttName:AttValue})
```

```
EventTree ::= (root: TreeNode)
TreeNode ::= (event: EventInstance,
              [childs: listof(TreeNode)])
EventInstance ::= (TypeName{,AttName-AttValue}
                  {,RoleName-RoleValue})
```

The set of entities function as the background knowledge of a specific domain and is shared by multiple event trees. Each event tree corresponds to a video or several videos as a whole. Each node corresponds to an event whose roles refer to entities in the entity set. The parent-child relationship between nodes denotes composing relationship between event instances and ordering among sibling nodes denotes temporal ordering between event instances.

3.4 User-defined Constraints

There are three kinds of user-defined constraints: type constraints limit whether an instance is valid, composing relationship constraints limit whether an event instance may be component event of another event instance, and temporal constraints limit whether an event instance may occur immediately after another event instance. The previous two kinds of constraints are defined in type definition and is enforced when an instance is created or when a node is inserted into an event tree. In this section we will focus on temporal constraints.

Temporal constraints are the most important constraints in SemTTE. They are used to constraint the temporal structure of a video in a specific domain. Following are two examples of temporal constraints in NBA game domain:

- ✓ After the first halftime (an event instance of type “Halftime” with the attribute “halftimenum” equals 1), there should be a second halftime (an event instance of type “Halftime” with the attribute “halftimenum” equals 2), and the second halftime’s begin scores should equal to the first halftime’s end scores;
- ✓ The event occurs after shoot may be score, foul, shoot fail, or pause.

In SemTTE, temporal constraints are only applied to two successive event instances that are direct components of the same event instance, that is, neighboring siblings in the event tree. It limits whether an event instance may occur immediately after another event instance.

3.4.1 Definition

A temporal constraint for an event type pType is defined in following form:

```
AddTemCons( pType, pCondition, sType, sCondition)
```

pType and sType are types of the previous event and successive event, while pCondition and sCondition are constraints for them. A temporal constraint means that an instance of pType satisfying pCondition **may** be followed by an instance of sType satisfying sCondition. The union

of all temporal constraints for a type gives all the possibilities of its immediate successive events. If the union is empty, then its instance may be followed by any event instances. pCondition and sCondition are Boolean expressions formed by connecting following basic conditions by AND and NOT nestedly:

- ◆ Attribute value constraint (AVC): constraints for values of an attribute;
- ◆ Single event attribute relationship constraint (SRC): constraints for relationship of two attributes both coming from the previous event or the successive event;
- ◆ Multiple event attribute relationship constraint (MRC): constraints for relationship of two attributes, one of which comes from the previous event and the other comes from the successive event

Previous conditions may contain the first two kinds of basic conditions, while successive condition may contain all the three kinds of basic conditions.

3.4.2 Implementation

The set of temporal constraints is stored in following data structure:

```
TempConstraint ::= {ConstForEvent}
ConstForEvent ::= (pType, {pCondition, {SuccEvent}})
SuccEvent ::= (sType, sCondition)
```

TempConstraint is a set of ConstForEvent, each ConstForEvent considers an event type, maintaining all its possible immediate successive events (sType and sCondition) under various pConditions. This structure is constructed by performing following steps for each pType:

1. get all temporal constraints for pType and denote them as a set CEs of tuples (pCondition, sEventSet: {(sType, sCondition)});
2. for CEs
 - a) do following iteratively until no change can be made:

get two tuples t1 and t2 from CEs, if t1.pCondition is equivalent to t2.pCondition, then remove t1 and t2 from CEs and add a new tuple (t1.pCondition, Union(t1.sEventSet, t2.sEventSet)) to CEs;
 - b) for each tuple in CEs, do the following
 - i. if pCondition is always true, then set pCondition to null; if pCondition is always false, then remove this tuple;
 - ii. remove duplicated SuccEvent (with equal sType and equivalent sCondition) from sEventSet.
 - iii. for each SuccEvent e in sEventSet, if e.sCondition is always true, then set e.sCondition to null; if e.sCondition is always false, then remove e from the set.
3. Add (pType, CEs) to TempConstraint.

3.4.3 Enforcement

Enforcement of temporal constraints is depicted as following problem:

For two event instances pe and se (the type is denoted as pet and set respectively), determine whether se may be immediately successive event of pe according to the temporal constraints defined.

The process of enforcement includes two steps:

1. find all possible immediate successive events with type set of pe, denoted as a set PSE of sConditions, as shown in algorithm 1;
2. check whether se satisfies at least one sCondition in PSE, if so se can occur after pe, otherwise it can't.

The main problem of algorithm 1 is to find all pConditions satisfied by pe on-line efficiently. The straightforward approach is to check them one by one to see whether it is satisfied by pe. Suppose there are m conditions need to check, each contains k basic conditions and t Boolean operators on average, and there are totally n different basic conditions in the m conditions. In the worst case, this approach requires totally $m*(k+t)$ operations. However, a lot of them are redundant since if a basic condition appears in multiple pConditions, it will be evaluated multiple times which is a great waste of time.

Algorithm BuildPSE (pe, set, TempCons)

INPUT: pe: previous event; set: type of successive event; TempCons: Temporal constraints;

OUTPUT: the set PSE;

BEGIN

 consForPE = get the ConstForEvent corresponding to type of pe from TempCons;

 conditions = filter consForPE by sType, retaining those whose sType equals to set, then remove the element pType and sType to construct a set of (pCondition, {sCondition})

 PSE = an empty set;

 Add all sConditions in conditions into PSE whose related pCondition is satisfied by pe

 return PSE;

END

Algorithm 1. get possible successive event condition list

To avoid these redundant works, we propose another approach which can reduce the required operations dramatically. Following structures are constructed off-line for each event type to assist the on-line enforcement.

- ◆ PCList/SCList: the ordered list of all basic conditions in some previous/successive conditions in temporal constraints defined for the type. For each basic condition c, add "c" and "NOT c" into the list;
- ◆ PTrue-Mapping/STrue-Mapping: the mapping from a binary number to a previous/successive condition. The binary number is used to decide whether the corresponding condition is satisfied. Each bit of the number represents a basic condition in PCList/SCList, and if all basic conditions corresponding to 1s in the number are true, the

condition is true. The process of construction is shown in algorithm 2.

Algorithm BuildTrueMapping(*EventType*)

INPUT: *EventType*: the event type

OUTPUT: *tMap*: the true mapping

BEGIN

CSet = the set of all previous/successive conditions of *EventType*

NUM = number of elements in PCList/SCList of *EventType*;

FOR every condition *C* in *CSet*

Push all NOT in *C* into the deepest of the expression;

Generate a binary number BPC(or BSC) for *C* with *NUM* bits. If *C* contains the *i*th element of PCList(or SCList) then the *i*th bit of the number is 1, otherwise 0;

add (BPC, *C*) (or (BSC,*C*)) into *tMap*;

END FOR

Return *tMap*;

END

Algorithm 2. building true-mapping for an event type

With these assistant structures, now we can get all satisfied pConditions more efficiently , as shown in algorithm 3.

Algorithm getSatisfiedConditions (*ei*, *cons*)

INPUT: *ei*: event instance; *cons*: the set of conditions

OUTPUT: *scons*: all conditions satisfied by *ei* in *cons*

BEGIN

TMap = subset of PTrue-Mapping for *ei*, containing only mappings from binary number to conditions in *cons*;

Evaluate all Boolean expressions in PCList of *ei*;

BE = generate a binary number from *ei*. If *ei* satisfies the *i*th item in the corresponding PCList, then the *i*th bit of the number is 1, otherwise 0;

scons = empty set;

FOR every *BPC* in *TMap*

cover = (*BE XOR BPC*) AND *BPC*; //bit operation

IF *cover* == 0

Add the condition related to *BPC* to *scons*;

END IF

END FOR

Return *scons*;

END

Algorithm 3. get all satisfied conditions

With this approach, in the worst case, we first evaluate all basic conditions once, and then whether a condition is satisfied can be determined with one bit operation. Thus, totally $m+n$ operations are required, which is a great improvement compared to $m*(k+t)$. Actually by making a few modifications to algorithm3, not all basic conditions need to be evaluated. A binary OR operation for all *BPC*s in *TMap* may be got first, then only basic conditions corresponding to 1s in the resulted number need to be evaluated. This will reduce the number of operations furthermore.

The second step of enforcement is similar to finding all satisfied pConditions of *pe*, with the difference that the process terminates once a condition satisfied by *se* is found in *PSE*. Both straightforward approach and the binary approach can be used to solve this problem and the performance depends on the specific situation and is affected by the position of the first condition that *se* satisfies.

3.5.3 Violation Actions

When an instance violates the user-defined constraints, users can choose one of the following actions to be taken according to the type of the constraint violated:

- ✓ type constraint: the entity or event can not be created as an instance or modifications can not be made;
- ✓ composing relationship constraint: the event instance can not be added as another event instance's component event;
- ✓ temporal constraint: following actions are available
 - Alert: indicates users that there exist violations and show them the constraints violated. Then users can correct the errors interactively and store the modified version into the database or just ignore them;
 - Ignore: ignore all violations;
 - Forbidden: all event instances that violate temporal constraints can not be inserted into the event tree.

4. Conclusion

In this paper, a video semantic model SemTTE is proposed to manage semantic information in videos. It takes consideration of semantic structure of videos which may be concluded as two features: temporal structure and typed events. It organizes the whole video into a tree of events, thus, the structure and semantics of the video can be better represented and queried. It provides mechanisms for users to define domain-specific constraints, including type constraints, composing relationship constraints and temporal constraints. For the enforcement of temporal constraints, an efficient method is proposed by virtue of binary numbers and bit operation.

Reference

- [1] Adah, S.; S.Candan, K.; Chen, S.-s. The Advanced Video Information System: Data Structures and Query Processing. *ACM Multimedia Systems* 1996, 4, 172-186
- [2] Tusch, R.; Kosch, H.; Boszormenyi, L. VIDEX: An Integrated Generic Video Indexing Approach; *ACM Multimedia*, 2000; pp 448-451
- [3] Tran, D. A.; Hua, K. A.; Vu, K. VideoGraph:A Graphical Object-based Model for Representing and Querying Video Data; 2000; *In Proc. of the 19th*

International Conference on Conceptual Modeling (ER2000), pp 383-396

- [4] Combi, C. Modeling temporal aspects of visual and textual objects in multimedia databases; *Int Workshop on Temporal Representation and Reasoning 2000*; pp 59-86
- [5] Hacid, M.-S.; Declair, C. A database approach for modeling and querying video data. *IEEE Transactions on Knowledge and Data Engineering* 2000, 12, 729-750
- [6] Yong, C.; De, X. Hierarchical semantic associative video model; In proc of *IEEE International Conference on Neural Networks and Signal Processing*, 2003; pp 1217-1220
- [7] Aygun, R. S.; Yazici, A. Modeling and Management of Fuzzy Information in Multimedia Database Applications; Technical Report, 2002
- [8] Arslan, U. A Semantic Data Model and Query Language for Video Databases; Master thesis, 2002
- [9] Ahmet Ekin, Sports Video Processing for Description, Summarization, and Search (Chapter 2 Structural and Semantic Video Modeling), phd thesis, , 2003
- [10] Yu Wang, Chunxiao Xing, Lizhu Zhou, THVDM: A Data Model for Video Management in Digital Library, proceedings of the 6th *International Conference of Asian Digital Libraries*, 2003, pp 178-192
- [11] Yu Wang, Lizhu Zhou, Chunxiao Xing, An Evaluation Method for Video Semantic Models. *11th International Workshop on Multimedia Information Systems*, 2005, Springer LNCS 3665, pp. 207 – 220