EDF: A framework for Semantic Annotation of Video

Pradeep Natarajan, Ramakant Nevatia
Institute for Robotics and Intelligent Systems,
University of Southern California,
Los Angeles. CA 90089-0273
{pnataraj, nevatia}@usc.edu

Abstract

Semantic annotation of multimedia data is needed for various tasks like content based indexing of databases and also for making inferences about the activities taking place in the environment. In this paper, we present a top level ontology which provides a framework for describing the semantic features in video. We do this in three steps – First, we identify the key components of semantic descriptions like objects and events and how domain specific ontologies can be developed from them. Second, we present a set of predicates for composing events and for describing various spatio-temporal relationships between events/entities. Third, we develop a scheme for reasoning with the developed ontologies to infer complex *events from* simple *events using relational algebra. Finally, we have demonstrated the utility of our framework by developing an ontology for a specific domain. We conclude by analyzing the performance of the reasoning mechanism with simulated events in this domain.*

1. Introduction

There is an increasing need to design efficient methods to semantically annotate video to store, retrieve and manage the information captured in them. Such annotations would not only help human users to easily query and manage their digital libraries, but also enable automated applications performing complicated tasks like video surveillance to create, store, exchange and reason with the data. Events occurring in observed scenes are one of the most important semantic entities that can be extracted from videos of the scene. Many event recognition algorithms have been reported in recent literature [8, 9, 10]; however, the representations used in the various methods are quite diverse. Also, most systems represent video events as isolated entities rather than capture the relations between them in a structured ontology. In order to address these issues, we propose an event description framework (EDF) to capture event semantics that enables storage, inference and retrieval of events from lower level event observations. While we do not address the processing of video data to make the lower level inferences such as detection and recognition of objects or *primitive* events, we anticipate that the higher level reasoning and knowledge will also help guide the extraction of lower level primitives.

There have been many proposals to define a representation for describing video events. One significant challenge is to come up with a standard representation for describing multimedia content that is easily understood by a human reader but also is formal so that it can be manipulated by machines. [3] presents a set of tools standardized by MPEG-7 for describing semantics in multimedia. While this work identifies key semantic entities like objects and events and allows users to specify properties and relations between them, it does not make specific commitments regarding the structure of events and also does not provide mechanisms to reason with the annotations. There has been a significant amount of research devoted to the development of event representation and recognition in the computer vision community as well. [8] suggested the use of stochastic context free grammars, [9] advocated use of hierarchical decomposition events into separate "*threads"* related by temporal constraints while [10] presented similar ideas for representing scenarios by specifying the characters involved in the scenario, the sub-scenarios and the constraints combining the sub-scenarios. Apart from these, there have been a few works focused on the development of a formal language for describing an ontology of events. [7] presents a *first order logic* like syntax for describing composite events in terms of primitive events and also provides a set of predicates for describing temporal relationships between them. [2]

presents a similar representation scheme based on the CASE [5] representation used in natural languages.

In our work, we adopt the approach proposed by Davidson in [4]. In this scheme, we view an event instance as the assertion that there exits a particular event of a given type with various roles in which other individuals participate. While this representation scheme is equivalent to the ones in [2] and [7] in the sense that whatever that can be expressed in those schemes can also be expressed in ours and vice versa, our scheme has the advantage that it provides a single template predicate for representing all events instead of defining a predicate for each event type. At the same time, we have also incorporated several attractive features of these schemes into our framework. For example, we allow a particular event type to be defined as a subclass of another type like in [7]. We also allow a hierarchical decomposition of complex events into simpler ones like in [2] as well as [7]. Another important feature of our framework is a set of predicates for describing spatio-temporal relationships between events and entities. We built this list based on the previous work on temporal relationships presented in [1] and various works on spatial databases summarized in [6].

The rest of the paper is organized as follows – In section 2, we present our framework for describing events and also an ontology for describing spatio-temporal relationships. In section 3, we present a tool we have developed for creating ontologies using our framework and demonstrate the applicability of our framework by developing a domain ontology using it. Further, we populate the event database using simulated events in the domain and present an analysis of the performance of the reasoning mechanism. Finally we conclude with a discussion of the presented framework and future research directions we are pursuing.

2. Event Description Framework (EDF)

In this section, we describe our *Event Description Framework* in three subsections – In the first subsection, we identify a set of classes for semantic annotation of multimedia data, and describe their properties and relationships. Then, we present a set of predicates for describing various relationships between events and entities. Finally, we present a mechanism to reason with the concepts developed, to make inferences.

2.1. Basic Semantic Features

In EDF, we identify 3 main semantic features for describing multimedia data as follows –

1) *Entities:* These are basically objects like *Person*, *Door*, *Car* etc. that are involved in a particular domain. For example, while describing events occurring in a meeting room, the various entities involved could be people, table, chair etc Each of these entity classes are subclasses of the top-level entity class or of other classes. Thus, in each domain we will have a hierarchy of the entity classes involved. Further, each of these entity classes can have various features associated with them. For example, a car can have a feature called *color* with the possible values being *red, blue, green, black* etc. The set of features for a particular entity class is the union of its own specific features as well as its super classes. Thus, if *car* is a subclass of the top-level *entity* class, and if *size* is a feature of *entity*, it is also a possible feature for *car*. An important point to note here is that while specifying the features for a particular instance of an entity class, it is not necessary to specify values for all possible features of that entity class. For example, if X is an instance of car, it is not necessary to specify its color. Another point to note here is that we do not allow for multiple inheritances while developing entity class hierarchies and instead encourage choosing sufficient features to distinguish between classes.

2) *Actions:* This class refers to various actions like run, walk etc. that takes place in a domain. For example, going back to the meeting domain, the various actions can be *speak*, *raise-hand*, *walk* etc. The action classes can be organized in a hierarchy similar to the entity classes and also have features associated with them (like *speed, tone* etc.). Each action has a *timestamp*, which specifies the *start time* and *end time* of the action. Further, they have a *patient specification* which describes the thing towards which the action is directed to, such as *X gave the book to Y* where Y is the patient. Finally, they can have various prepositional phrases associated with them such as the *Instrument* used to perform the action etc.

3) *Events:* We define two classes of events, namely –

a) *Primitive Events:* These are basically (Actor*, Action*) tuples, where *Actor** is a set of entities that initiate the event, and *Action** is a set of actions performed during the course of the event. For example, in the event *Jack and Jill went up the hill*, Jack and Jill are the actors and "*went up the hill*" is the action.

b) *Composite Events:* Composite events are compositions of other primitive/composite events. Composite event definitions are specified using the predicate *PROCESS* whose first argument is the event being defined and whose second argument is the composition of other events. To define compositions of events, we have the following predicates –

- *Sequence* – represents a set of events which happen one after another in a temporal sequence.
- *AND* – represents a set of events with no particular temporal relationship between them.
- *OR* – represents a set of *alternate* events of which at least one should occur.

Apart from these basic predicates, we also have a set of predicates to describe various spatial and temporal relationships between events and entities. These predicates are presented in subsection 2.2. Figure 1 shows the hierarchy of semantic features described in this subsection. Here, we would like to point out the close relationship between our proposed ontology and the structure of English sentences. The *entity* descriptions basically specify the *Noun Phrases*, while the *action* descriptions specify *Verb Phrases* and the simple events, which are (*Actor, Action*) pairs, are equivalent to sentences.

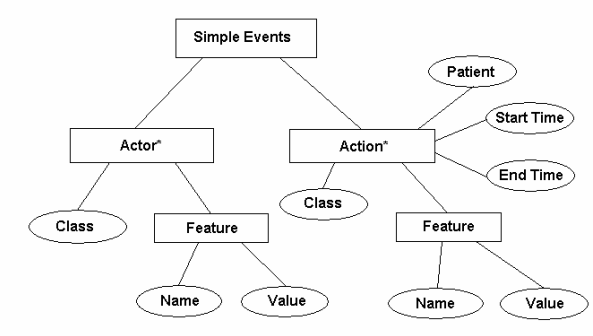

Figure 1: EDF Hierarchy

2.2. Ontology for describing Spatio-Temporal Relationships

In this section, we present a set of predicates for describing *spatio-temporal* relationships, based primarily on our study of various previous works. In order to define temporal relationships between two events, we defined the following predicates – *metby, meets, finishes, finishedby, startedby, starts, during, after, before, overlaps, overlappedby, contains, simultaneous*.

In each of these, the predicates have two arguments and they can be either *time intervals* or *events*. These predicates are based on Allen's *Interval Algebra* [1], and interested readers may refer to it for further details.

In order to define spatial relationships between events/entities, we present a set of predicates based on previous work in spatial databases as well as our own research on prepositions in English. [6] gives a good introduction to the issues involved in designing spatial databases. In general, spatial relationships can be classified into three types –

1) *Topological Relationships:* These are invariant under translation, rotation and scaling.

2) *Directional Relationships:* These describe relationships like "*above*", "*below*" etc. involving the relative direction between two events/entities.

3) *Metric Relationships:* These involve relationships like *distance* < *100* involving constraints on spatial metrics.

Of these, topological relationships are well studied. Studies [6] have shown that all these relationships can be expressed using 6 basic relationships (*"touch", "in", "cover", "equal", "overlap", "disjoint"*) and 3 operators *b* (which when applied on an area returns the boundary), *f* and *t* which return the end points of a line. Thus, we have included these as part of EDF.

Next, in order to define a set of predicates for describing directional relationships, we conducted a study of prepositions in English. The intuition behind this is that all directional relationships can be expressed in English using prepositions. Based on our study, we have defined the following predicates –

over, upon, opposite, behind, in-front-of, left-of, right-of, above, below.

In each of these, the arguments to the predicates can be *events/entities/regions*. We need to specify the coordinate system while using these directional relationships. Further, we have also defined the predicates *near* and *far* for metric relationships and the predicate *at* to specify the location of an entity or event. Apart from these basic predicates, we also allow the user to specify other predicates depending on the domain of interest.

2.3. Reasoning with EDF

In this section, we present our scheme for reasoning with the developed ontologies in order to infer composite events and relationships from the primitive events. We do this by organizing the entity, action and simple event definitions into relational tables and translating the complex event definitions into

SQL/XQuery style queries. We have defined the following relational tables –

1) *Thing:* This is a top-level table that contains two fields – *ThingNo* and *Type*. *ThingNo* is the key field and the keys of all the other tables must be one of the values of *ThingNo* in this table. *Type* can have 3 values – *Entity/Action/Event* depending on whether the "*Thing*" is an *Entity* or *Action* or *Simple Event* respectively.

2) *Entity:* This table stores the basic information about various entity instances. It consists of 3 fields – *ThingNo* which basically points to the *Thing* table, the *Type* field which specifies which entity class, this instance belongs to (like *car*, *person* etc) and the *Name* field which can be a default name for the instance or a user specified one.

3) *EntityFeature:* This table stores information about the features of various entities. It consists of 4 fields – the *fieldNo* which is basically an index of the table, the *entityNo* which contains the *ThingNo* of the entity whose feature is being described, the *featureName* and *featureValue* fields which contain the name of the feature being specified and its value, the *startTime* and *endTime* fields which specify the time period during which the entity had the feature. The last two fields were introduced to take into account the temporal nature of some features. If the entity has the feature always, the startTime and endTime fields can be left as NULL.

4) *Action:* The action table stores information related to the actions taking place in a stream and contains the following fields – *ThingNo* which points to the *Thing* table, the *Type* field which specifies the action class (like *run, move* etc), the *startTime* and *endTime* fields which specify the time interval during which the action takes place and finally the *Patient* field which contains a list of *ThingNo*s of entities that are patient to this action.

5) *ActionFeature:* This table stores information about the features of various actions and has a structure which is exactly equivalent to the *EntityFeature* table.

6) *SimpleEvents:* This table stores the simple events that can be directly inferred from the entities and actions seen in a stream. It consists of 3 fields – the *ThingNo* field which points to the *Thing* table, the *Actor* field which contains a list of *ThingNo*s of entities that initiate the event and *Action* field which contains a list of actions performed by the actor(s).

With this representation, it is now easy to represent composite event definitions in terms of primitive events and features using SQL style queries. This is because, the predicates *AND* and *OR* used for describing complex events, directly map to the AND and OR of SQL/XQuery, and the predicate *SEQUENCE* can be easily represented using constraints on the *startTime* and *endTime* fields of actions/features. Also, for each of the spatio-temporal predicates, we translate them to conditions on the time/location of the things. For example, if event *e1* is *before* event *e2*, it gets translated to *e1.startTime* < *e2.startTime* and so on. This approach to reasoning is equivalent to using a first order logic theorem prover, since relational algebra is directly based on first order logic. Further, we can use a standard database management system like Oracle/PSQL to store event data. These systems have several inbuilt query optimization techniques to store and retrieve data more efficiently. In the next section, we present a tool that facilitates the storage and retrieval of event data in the format we have described above and a specific domain ontology developed using it.

3. EDF Tool and Applications

In the previous section, we presented our framework for developing event ontologies and our scheme for storing and reasoning with event data. One issue with the presented framework is that it is hard for the user to insert data into the database, while maintaining the integrity constraints. In order to make it easier for the user to develop event ontologies and to insert, search and retrieve event data, we have developed a tool which has a simple easy-to-use interface and also automatically takes care of integrity constraints. Figures 2 and 3 show screenshots of this tool. It basically consists of 3 tabs – An *Entity Tab*, an *Action Tab* and an *Event Tab*. The entity and action tabs contain interfaces for describing class hierarchies, inserting individual instances, defining features and inserting feature values. The event tab contains interfaces for defining simple events as well as complex events and an area to see the results of complex event queries. Also, in our implementation, we store the tables as XML files and retrieve complex events using XQuery which for our purposes is equivalent to SQL.

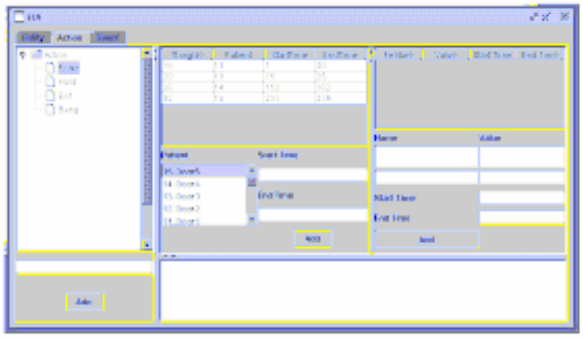

Figure 2: EDF Tool – Action Tab

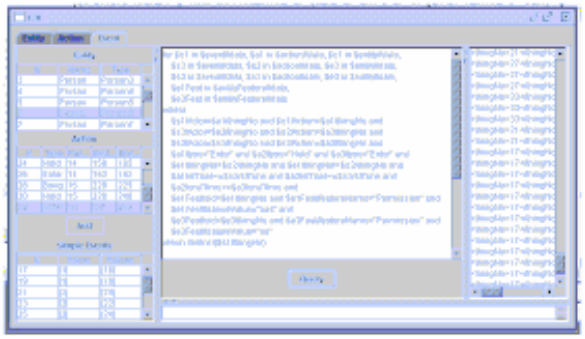

Figure 3: EDF Tool – Event Tab

We will now illustrate the utility of the framework by developing an ontology for a specific domain – namely, the *sneak-in* domain that describes possible ways in which an intruder can sneak into a protected facility. Based on our study, we classified the various sneak-in methods into the following categories –

1) Methods of sneaking into doors with locks.
2) Methods for doors which are open and guarded.
3) Methods involving non standard entrances not carefully guarded or locked.

We then proceeded to define subclasses for each of these categories and then defined the events in each sub category using our framework. Due to space limitations, we will only discuss a subset of the events belonging to category 1 in the following discussion. We identified two major subclasses of events and specific event patterns in category 1 as follows-

1) Intruder gets building user to let them in.
   a) Building user is going in, holds door, lets intruder in.
   b) Building user is exiting, holds door, lets intruder in.
   c) Intruder bangs on door, asks building user inside to let them in.
2) Door opened by building user not aware of the intruder.
   a. *Tailgating:* intruder hides and waits for entering building user, then intruder closely tails behind building user and enters right behind them.
   b. Intruder hides very close to the door, waits for someone to exit, intruder quickly catches the door and enters.

Next, we proceeded to describe these patterns formally using our framework. In particular, the 3 events in the case where the intruder gets a building user to let them in were described in a syntax similar to [7] as follows –

1) *PROCESS(SneakInEvent(Person Violator, Person Helper, Time StartTime, Time EndTime), AND(Permission(Violator,"no"), Persmission(Helper,"yes"), SEQUENCE(Event(e1),Event(e2),Event(e3)), Actor(e1, Helper), Action(e1, "Enter", Door d, null, null), Actor(e2, Helper), Action(e2, "Hold", d, null, null), Actor(e3, Violator), Action(e3, "Enter", d, null, null)))*
2) *PROCESS(SneakInEvent(Person Violator, Person Helper, Time StartTime, Time EndTime), AND(Permission(Violator,"no"), Persmission(Helper,"yes"), SEQUENCE(Event(e1),Event(e2),Event(e3)), Actor(e1, Helper), Action(e1, "Exit", Door d, null, null), Actor(e2, Helper), Action(e2, "Hold", d, null, null), Actor(e3, Violator), Action(e3, "Enter", d, null, null)))*
3) *PROCESS(SneakInEvent(Person Violator, Person Helper, Time StartTime, Time EndTime), AND(Permission(Violator,"no"), Persmission(Helper,"yes"), SEQUENCE(Event(e1),Event(e2),Event(e3)), Actor(e1, Helper), Action(e1, "Bang", Door d, null, null), Actor(e2,Violator), Action(e2, "Hold", d, null, null), Actor(e3, Violator), Action(e3, "Enter", d, null, null)))*

The above statements have a simple interpretation in English. For example, in the first statement, in the first argument of the PROCESS statement, we are saying that in a *SneakInEvent* we are interested in two entities of type *Person*, namely the violator/intruder and the helper and also the time interval during which the sneak-in happened. In the second argument, we are

saying that a sneak-in happens every time the following conditions are met –

- There is a person *Violator* who does not have permission to enter the building and there is another person *Helper* who has permission to enter.
- The following events happen in sequence one after another-
    - *Helper* enters through door d.
    - *Helper* holds the same door d.
    - *Violator* enters through door d.

Next, we translated each of these statements into a query on the relational tables described in section 3. For example, statement 1 above gets translated into the following XQuery –

```
<SneakInEvents>

{for $s1 in $event//data, $a1 in $action//data[thingNo=$s1/Action], $e1 in $entity//data[thingNo=$s1/Actor], "$a2 in $action//data[number(stTime)>=number($a1/stTime) and number(stTime)<number($a1/endTime)], $s2 in $event//data[Action=$a2/thingNo], $e2 in $entity//data[thingNo=$s1/Actor], $a3 in $action//data[number(stTime)>number($a2/stTime) and number(stTime)<number($a2/endTime)], $s3 in $event//data[Action=$a3/thingNo], $e3 in $entity//data[thingNo=$s3/Actor], $e1Feat in $entityFeature//data[eid=$e1/thingNo], $e3Feat in $entityFeature//data[eid=$e3/thingNo]

where $a1/type="Exit"and $a2/type="Hold" and $a3/type="Enter" and $e1/thingNo=$e2/thingNo and $e1/thingNo!=$e3/thingNo and $e1Feat/featureName="Permission" and "$e1Feat/featureValue= "has" and $e3Feat/featureName= "Permission" and "$e3Feat/featureValue="no" and number($a1/Patient) = number($a2/Patient) and number($a2/Patient) = number($a3/Patient)

return

<SneakInEvent>
        <Violator>{$e3/Name/text()}</Violator>
        <Helper>{$e1/Name/text()}</Helper>
        <StartTime>{$a1/stTime/text()}</StartTime>
        <EndTime>{$a3/endTime/text()}</EndTime>
</SneakInEvent>

}</SneakInEvents>
```

Again, while the syntax above looks complicated, it has a fairly simple interpretation in English and can be learned quickly. The most important thing to note here is the way the statement *Sequence(s1,s2,s3)* gets translated into XQuery. We are effectively saying that two events *s1* and *s2* are in sequence if their time intervals overlap, i.e. s1's start time is before s2's and s1's end time is after s2's start time. We can extend this definition of sequence to more than 2 events where s1 overlaps with s2, s2 with s3 and so on. While the term *sequence* has a more general (though imprecise) interpretation in normal conversation, we chose to use this definition since it was often encountered in our test cases. The more complicated relationships between the events can be expressed using the other spatio-temporal predicates we have described in section 2.2.

In the discussion so far, our focus had been on defining an exhaustive list of ways in which an intruder can sneak into a protected building and then inferring a sneak-in every time one of these patterns is seen. However, this may not be necessary in many applications. For example, we might just want to know all the sneak-in's that happened during some time interval and who helped it, rather than the specific type of sneak-in that happened. In this case, we need to look for the following-

1) Find all events in which a person without permission enters the building. He is the violator.
2) Then for each event identified in (1), find the event in which a building user held the door open for the violator to enter. The actor in this event is the helper. Further, this event should precede the event in (1) and also overlap with it.

This can be expressed using the following query –

```
<SneakInEvents>

{for $s1 in $event//data, $a1 in $action//data[thingNo=$s1/Action], $e1 in $entity//data[thingNo=$s1/Actor], $a2 in $action//data[number(stTime)<=number($a1/stTime) and number(endTime)>=number($a1/stTime)], $s2 in $event//data[Action=$a2/thingNo], $e2 in $entity//data[thingNo=$s2/Actor], $e1Feat in $entityFeature//data[eid=$e1/thingNo]

where $e1Feat/featureName= "Permission" and $e1Feat/featureValue= "no" and $a1/type="Enter" and $a2/type="Hold" and number($a1/Patient) = number($a2/Patient) return <SneakInEvent>
<Violator>{$e1/Name/text()}</Violator>
<Helper>{$e2/Name/text()}</Helper>
</SneakInEvent>}</SneakInEvents>
```

Besides these queries for identifying composite events, our representation can also be used to make a variety of queries on the event database such as to find all events in which a particular entity took part etc. In the next section, we present the results obtained by applying the

ontology developed in this section to a database of events.

4. Experiments and Results

To test our proposed framework, we populated a database of events by simulating the events that take place in a typical building that is secured. We defined two entity classes of interest, namely *Person* and *Door*, where *Person* includes all the people involved and *Door* includes all the points of entry/exit into the building. Each person has a feature "*Permission*" which can have values *has/no*. Here, we make the assumption that there is an independent mechanism to determine if a person has permission or not, such as card readers, keys etc. Further, we defined four action classes – *Enter, Exit, Hold* and *Bang*. Each of these actions is performed by a person with a door as the patient. We then populated the database with information on the people using the building and the events in which they participated. We did this in such a way that the total number of events was about five times the number of persons involved. We varied the number of persons involved from 20 to 500 and tested the performance of 4 queries for each of these cases – 3 of the queries corresponded to the complex event definitions of the sneak-in events where the intruder gets a building user to open the door for them and the fourth one was the general sneak-in query described at the end of section 3. Since the accuracy of the composite events recognized is always 100% provided the queries are formulated correctly, we decided to focus on the complexity of the inference procedure rather than the accuracy to measure performance. This also made sense, since in an earlier implementation we had used a first order logic theorem prover (OTTER), which had exponential complexity in many cases. Figures 4-7 show the variation in the time to execute the queries with the size of the database.

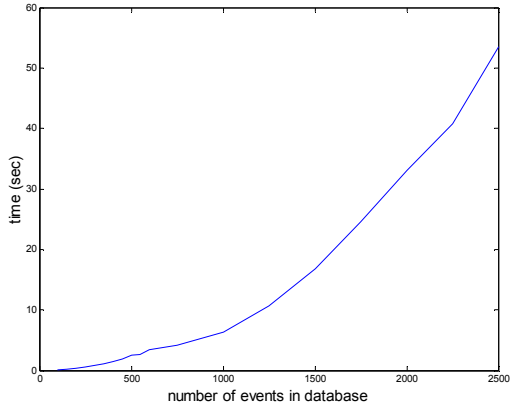

Figure 4: Performance of query 1 with the size of database.

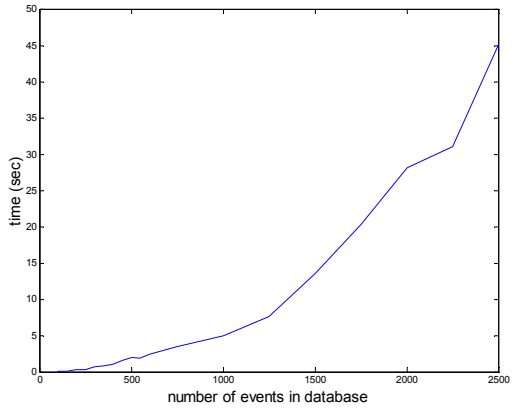

Figure 5: Performance of query 2 with the size of database.

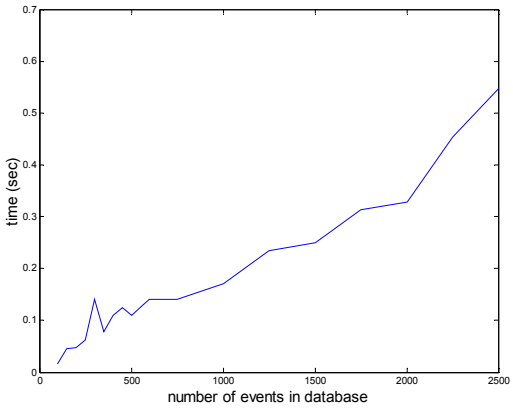

Figure 6: Performance of query 3 with the size of database.

As can be seen, the queries in general take only a few milliseconds per event in the database and thus are real time. However, a more careful analysis of the statistics shows that the time taken for the first two queries is $O(n^2)$ where n = number of events, while the

last two queries are nearly linear. This is primarily due to the fact that the join operations performed in the first two queries take $O(n^2)$ time. However, this operation can be converted into O(n) by using *hash join* [11] and thus optimized further.

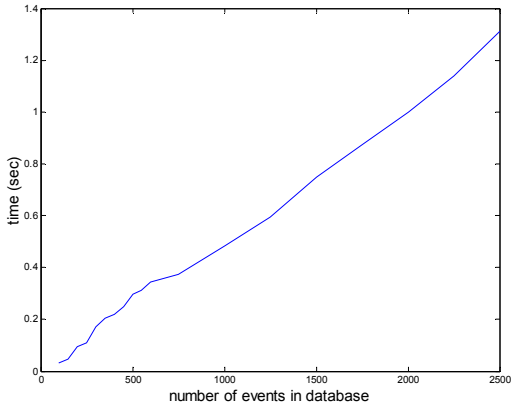

Figure 7: Performance of query 4 with the size of database.

From these results, we can conclude that our representation scheme can be used to recognize composite events from primitive event data in real time. All these tests were performed on Windows PC: CPU 2GHz and 512MB RAM.

5. Discussion and Conclusion

In this paper, we presented a framework for developing ontologies for semantic annotation of multimedia and also a method to store, retrieve and infer complex events. We developed a specific domain ontology using our framework and demonstrated its utility. We are currently exploring the possibility of importing a large existing knowledge base like CYC [13] to our framework and also the possibility of developing a media level annotation tool that can produce the event annotations in a semi automated manner.

6. Acknowledgements

This research was partially funded by the Advanced Research and Development Activity of the U.S. Government under contract MDA-904-03-C-1786. The sneak-in ontology described in this paper is based on the work done by Dr.Brian Burns at SRI [12].

7. References

[1] Allen, James F., and George Ferguson, "Actions and Events in Interval Temporal Logic," in O. Stock (ed.), "*Spatial and Temporal Reasoning*", Kluwer Academic Publishers, Dordrecht, Netherlands, 1997, pp. 205-245.

[2] Asaad Hakeem, Yaser Sheikh, Mubarak Shah, *"CASEE: A Hierarchical Event Representation for the Analysis of Videos"*, The Nineteenth National Conference on Artificial Intelligence (AAAI), San Jose, USA, Jul 2004, pp. 263-268.

[3] A. B. Benitez, H. Rising, C. Jrgensen, R. Leonardi, A. Bugatti, K. Hasida, R. Mehotra, A. M. Tekalp, A. Ekin, and T.Walker, *"Semantics of multimedia in MPEG-7"*, in *Proc. IEEE Int Conf. Image Process.,* vol. 1, 2002, pp. 137-140.

[4] Davidson, D. "*The Logical Form of Action Sentences*". In Davidson, D., Essays on Actions and Events. Oxford: Oxford University Press, 1980, pp. 105-148

[5] Fillmore.C.J. *"The Case for CASE"*, In *Universals in linguistic theory*, ed. E. Bach and R. Harms, New York: Holt, Rinehart, and Winston, 1968, pp. 1-88.

[6] Ralf Hartmut Guting, *"An Introduction to Spatial Database Systems",* VLDB Journal 3(4): 357-399, 1994.

[7] Nevatia.R, Bolles B, Hobbs.J. *"An Ontology for Video Event Representation"*, Event Mining Workshop, CVPR, Washington D.C, USA, June 2004, pp 119-119.

[8] Y.A. Ivanov, A.F. Bobick, *"Recognition of Visual Activities and Interactions by Stochastic Parsing"*, IEEE Trans. On PAMI, no. 8, August 2000, pp. 852-872.

[9] S. Hongeng, R. Nevatia, *"Multi-agent event recognition"*, Proceedings of IEEE ICCV, vol. 2, Vancouver, Canada, Jul 2001, pp. 84-91.

[10] T. Vu, F. Bremond, M. Thonnat, *"Automatic Video Interpretation: A Novel Algorithm for Temporal Scenario Recognition"*, The Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, 9-15 August 2003.

[11] Ramakrishnan, Gehrke, *"Database Management Systems"*, McGraw Hill, 2003.

[12] J.Brian Burns, "An ontology of a video event: Sneaking in", Technical report, 2004.

[13] http://www.cyc.com