

A SEMANTIC DATA MODEL AND QUERY LANGUAGE FOR VIDEO DATABASES

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BİLKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Umut Arslan

January, 2002

I certify that I have read this thesis and that in my opinion it is fully adequate,
in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Uğur Güdükbay(Co-supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate,
in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Cevdet Aykanat

I certify that I have read this thesis and that in my opinion it is fully adequate,
in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Attila Gürsoy

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

A SEMANTIC DATA MODEL AND QUERY LANGUAGE FOR VIDEO DATABASES

Umut Arslan

M.S. in Computer Engineering

Supervisors: Assoc. Prof. Dr. Özgür Ulusoy and

Assist. Prof. Dr. Uğur Güdükbay

January, 2002

Advances in compression techniques, decreasing cost of storage, and high-speed transmission have facilitated the way video is created, stored and distributed. As a consequence, video is now being used in many application areas. The increase in the amount of video data deployed and used in today's applications not only caused video to draw more attention as a multimedia data type, but also led to the requirement of efficient management of video data. Management of video data paved the way for new research areas, such as indexing and retrieval of videos with respect to their spatio-temporal, visual and semantic contents. In this thesis, semantic content of video is studied, where video metadata, activities, actions and objects of interest are considered within the context of video semantic content. A data model is proposed to model video semantic content, which is extracted from video data by a video annotation tool. A video query language is also provided to support semantic queries on video data.

The outcome of this thesis work will be used in a project, which proposes a video database system architecture with spatio-temporal, object-trajectory and object-appearance query facilities so as to build a complete video search system to query video data by its spatio-temporal, visual and semantic features.

Keywords: video databases, semantic video modeling, annotation of video data, semantic querying of video data.

ÖZET

VIDEO VERİLERİ İÇİN ANLAMSAL VERİ MODELİ VE SORGULAMA DİLİ

Umut Arslan
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticileri: Doç. Dr. Özgür Ulusoy ve
Yrd. Doç. Dr. Uğur Güdükbay
Ocak, 2002

Sıkıştırma tekniklerindeki gelişmeler, veri saklama maliyetlerinin düşmesi ve yüksek hızda veri transferi, videonun oluşturulması, saklanması ve dağıtılmasını kolaylaştırmıştır. Bundan dolayı, video günümüzde birçok uygulama alanında kullanılmaktadır. Uygulama alanlarında oluşturulan ve kullanılan video sayısında ve veri miktarındaki artış, hem videonun sadece bir mültimedya (çoklu iletişim aracı) veri tipi olarak daha çok ilgi çekmesini sağlamış, hem de video verilerinin daha verimli bir şekilde kullanılması gereğini doğurmuştur. Video verilerinin yönetimi, videonun yer-zaman, görsel ve anlamsal içeriklerine göre endekslenmesi ve yeniden edinilmesi gibi araştırma alanlarının oluşmasına olanak sağlamıştır. Bu tezde videonun anlamsal içeriği üzerine çalışılmıştır. Videonun anlamsal içeriği video ile ilgili bibliyografik verileri, video içinde geçen etkinlikleri, aksiyonları (hareketleri) ve ilginç nesneleri kapsar. Videonun anlamsal içeriğini modellemek için bir video veri modeli önerilmektedir. Videonun anlamsal içeriği bir çıkarım aracı ile elde edilir. Video verileri üzerinde anlamsal sorgular yapabilmek amacıyla bir video sorgulama dili sağlanmıştır.

Bu tezin sonuçları, yer-zaman, nesne yörüngesi, nesne görünüm sorgulamalarını destekleyen video veritabanı mimarisinin önerildiği bir projede kullanılacaktır. Buradaki amaç yer-zaman, görsel ve anlamsal sorguları destekleyen eksiksiz bir video arama sistemi oluşturmaktır.

Anahtar sözcükler: video veritabanları, anlamsal video modelleme, video verilerinden çıkarımlar yapma, video verilerini anlamsal sorgulama.

Acknowledgement

I would like to express my gratitude to my supervisors Assoc. Prof. Dr. Özgür Ulusoy and Assist. Prof. Dr. Uğur Güdükbay for their instructive comments in the supervision of the thesis.

I would like to express my special thanks and gratitude to Prof. Dr. Cevdet Aykanat and Assist. Prof. Dr. Attila Gürsoy for showing keen interest to the subject matter and accepting to read and review the thesis.

I would like to acknowledge the support of Turkish Scientific and Technical Research Council (TÜBİTAK) under the grant number EEEAG-199E025.

I would like to express my special thanks to Mehmet Emin Dönderler for reading and reviewing the thesis and for his support and patience.

I thank to my family and to my friends Feyzal, Engin, Sengör, and Ediz for their support.

Finally, I thank to my managers İlker Kuruöz and Fatih Bektaşoğlu for their support.

Contents

1	Introduction	1
1.1	Organization of the Thesis	5
2	Related Work	6
3	Video Database System	10
3.1	Video Database System Architecture	10
3.2	Video Data Model	12
3.3	Query Language	12
3.4	Query Processing	13
4	Semantic Video Model	16
4.1	The Data Model	17
4.1.1	Hierarchical Structure	18
4.1.2	Temporal Management	19
4.2	Database Design	19

5	Annotation Tool	21
5.1	Initialization Process and the Main User Interface	23
5.2	The Order of Annotation	24
5.3	Utility Data Annotation	26
5.4	Video Metadata Annotation	26
5.5	Hierarchical Video Tree	27
5.6	Object Annotation	30
5.7	Event Annotation	31
5.8	Subevent Annotation	36
5.9	Hierarchical Video Tree Functionality	38
5.10	Implementation Details	40
6	Semantic Query Language	43
6.1	Features of the Language	43
6.2	Types of Conditions	44
6.2.1	Metadata Conditions	45
6.2.2	Event Conditions	45
6.2.3	Object Conditions	50
6.3	Semantic Conditions	50
6.3.1	Semantic Conditions for Output Type Videos	51
6.3.2	Semantic Conditions for Output Type Sequences	51

6.3.3 Semantic Conditions for Output Type Scenes	51
7 Conclusions and Future Work	52
Bibliography	55
Appendices	58
A Semantic Query Grammar Specification	58
B Java Class Definitions	66
C Database Table Specifications	70

List of Figures

3.1	Video Database System Architecture.	11
3.2	Web client - query processor interaction.	13
3.3	Query processing phases.	14
4.1	Database design of the semantic video model.	20
5.1	The main user interface of the annotation tool.	23
5.2	The main user interface with a video file opened.	24
5.3	Utility window.	26
5.4	Video metadata annotation process.	27
5.5	Video window with data inserted.	28
5.6	Hierarchy tree of a video.	29
5.7	The user interface with hierarchy tree after video metadata annotation.	29
5.8	Object Window with data inserted.	30
5.9	The user interface with video tree hierarchy after object annotation.	31
5.10	Event annotation, event specification process.	32

5.11	Event annotation, object selection process.	33
5.12	Event annotation, role definition process.	34
5.13	The user interface with hierarchy tree after event annotation. . . .	35
5.14	Subevent annotation, subevent specification process.	36
5.15	Subevent annotation, object selection process.	37
5.16	The hierarchy after subevent annotation process.	38

Chapter 1

Introduction

Advances in compression techniques, decreasing cost of storage, and high-speed transmission have facilitated the way video is created, stored and distributed. These improvements in technology created new application areas, where large amounts of video data is used, such as digital libraries, distance learning, public information systems, electronic commerce, video-on-demand systems, and so on. Consequently, more and more videos are created each day, and this fact leads to an enormous growth in the number of videos to be dealt with. The fast increase in the amount of video data caused video to draw more attention as a multimedia data type and revealed an important problem; new methods should be developed to manage them because existing data management techniques do not provide sufficient support for video data type.

Traditional database systems are not suitable to manage videos since video data has its own characteristics, which differentiates it from simple textual or numerical data. A video consists of a sequence of frames, which are nothing but images. Therefore, video data possesses all attributes of image data such as color, texture, object layout, shape of objects, etc. Moreover, huge volumes of data, audio content and temporal structure can be listed as attributes that differentiate video data from image data. The volume of data contained in a video is huge compared to image data since a typical video clip may contain thousands of images. Temporal structure of video provides events and motions

of objects to be specified as attributes particular to video. To summarize, video data has rich audio-visual content, and thereby, new methods for managing video data are required.

Some terms need to be defined before the explanation of the concepts in management of video data: *Video* consists of *frames*. *Shot* is a set of frames recorded in a single camera action. *Keyframe*, which is one frame of the shot, can be used to identify the shot. *Scene* is a sequential collection of shots unified by a common event or locale. *Sequence* is a collection of semantically related scenes that need not be consecutive in time, and sequences build up the video. To conclude, video has a hierarchical structure, where sequences, scenes, shots and frames constitute the levels in the hierarchy.

As stated in [10], data related with video can be divided into two groups. The first group is metadata about video. Data, which is not extracted from video content like video name, production year, etc., can be modeled as metadata about video. Extracted data from video content constructs the second group of data and is also categorized into two groups: *physical data* and *semantic data*. Low-level features like color, texture, shape and spatial relationships of objects can be specified as physical data. Semantic data covers events, actions, attributes and relations of objects.

Management of video data paved the way for new research areas, such as indexing and retrieval of videos with respect to their spatio-temporal, visual and semantic contents. Traditional database systems must be enhanced with new capabilities to handle video data, as a data type to query, and the first step in achieving this goal would be to create a video model and incorporate it into the existing database architectures. The defined model can then be used for retrieval and querying of video with the extracted information. Video indexing is the process of segmentation of video for efficient management and retrieval. Video modeling can be defined as the process of translating raw video data into an efficient internal representation. Video retrieval is the process of retrieving video according to its content. There are many works dealing with these aspects, some of which will be described in Chapter 2.

Video can be modeled in many different ways. Since video consists of sequential images (frames) in a time-line, models applied to image files can also be applied to video files. Color, texture of video and objects appearing in video can be modeled by the same methods applied to images.

Spatial, temporal and semantic models of videos have also been provided. Spatial models deal with the relative locations of objects in video frames whereas temporal models deal with the locations of objects between video frames on a time-line. Spatial and temporal models are merged to construct spatio-temporal models. Semantic models deal with the meaning perceived by humans when a video is watched.

Video indexing mechanisms can be categorized into three groups: *automatic*, *semi-automatic* and *manual*. Computer vision techniques are used to automatically index video with respect to physical features like color and texture. In semi-automatic indexing, computer vision techniques are used with human interaction in the process. In manual indexing, indexing process is performed by end-users. Indexing algorithms are used to segment videos into smaller units for efficient management. Shot boundary detection algorithms are used to divide videos into shots. Scene change detection algorithms decide where the scene changes occur in a video file. The output of such algorithms is also used for browsing and retrieval of videos. Clustering methods have been used to index shots into scenes.

Feature extraction algorithms have been proposed to analyze video data. Audio streams, color histograms and textures of frames are analyzed to extract semantic information from video. Compressed video streams are examined to annotate motions of objects that appear in video, where *annotation* is the name given to the process of analysis of video files. Automatic feature extraction techniques cannot directly extract semantic information from a video, but a number of systems have been proposed that model high-level data like events in video. However, these systems are generally domain specific and cannot be used to model every type of video. News and sports are examples of some domains where automatic extraction of high-level data is possible. Objects that appear in a video

can be annotated using object extraction algorithms.

Summary of a video can be created by specifying a few important scenes and the other important scenes are detected automatically, which are the ones that are similar to the specified scenes with respect to a given similarity constraint. Heuristics are defined for similarity measurements. For classification of videos, reasoning methods have been defined. For example, if there are more close-ups in a movie, then the movie can be of type romance.

For querying video data, indexes created by video indexing algorithms are used. Moreover, video is queried by image content. Given an image, the most similar frames in video are returned with respect to spatio-temporal attributes of objects, color histograms, and texture analysis. Query-by-example, iconic-based query, query by annotated text, query by motion and trajectory of objects, query by semantic content are some of the proposed methods for retrieval of video.

In this thesis, we concentrate on the extraction, modeling, and querying of semantic information in video databases. The content of the semantic information includes bibliographic data about video and events, actions, and objects of interest taking place in video.

We propose a semantic video model which models semantic information in video. Video is modeled in a hierarchy of *events*, *subevents* and *objects* of interest. Video consists of events and events consist of subevents. Moreover, objects are modeled in every level in the hierarchy. An event is an instance of activity, which may involve many different objects over a time period. Subevents are used to detail the activity (event) into actions, and to model relations between objects of interest. The hierarchical model provides many semantic levels that facilitate understanding of video content. A database model is constructed to have proper database management support for the semantic video model.

We have implemented an annotation tool in Java to extract the semantic information from videos, and to view and update semantic information that has already been extracted. The annotation tool is a database application that is used to insert and update semantic information extracted from videos to the database

defined for the semantic video model.

Finally, we have designed a query language to facilitate video retrieval according to semantic conditions. Three types of queries can be issued by the end-users. *Video metadata queries* are used to retrieve videos with respect to bibliographic data. *Event queries* are used to retrieve videos according to activities and actions that take place in video. *Object queries* are used to query videos according to objects of interest, their attributes and roles in activities.

The work done in this thesis constitutes a part of a video database system architecture, proposed by Dönderler et al. in [5, 6, 7, 8]. Therein, a rule-based spatio-temporal model for videos and a video query processor, which can answer spatial, temporal, trajectory, motion and object queries for videos, are proposed. The work done in this thesis will be integrated into that video database architecture. Thus, the extended query processor will be able to answer semantic queries as well.

1.1 Organization of the Thesis

The remainder of the thesis is organized as follows. In Chapter 2, related work about semantic modeling of video is discussed. The video database architecture, into which the proposed semantic model will be integrated, is described in Chapter 3. In Chapter 4, the semantic data model that we propose is defined. The annotation tool developed, which is used to extract semantic information from videos, is introduced in Chapter 5. In Chapter 6, the query language that is used to retrieve semantic content from videos is described. Conclusions of our work and future research directions are given in Chapter 7. Grammar specification of our semantic query language is presented in Appendix A. Java class definitions used in the implementation of the annotation tool are given in Appendix B. Database table specifications for the semantic video model are presented in Appendix C.

Chapter 2

Related Work

In the literature, there are numerous works about indexing, modeling, and retrieval of the semantic content of videos. As stated in [13], semantic conceptualization can be performed at several levels of information granularity. At the finest level of granularity, video data can be indexed based on low-level features such as color, texture, shape, and objects. At a coarser level of granularity, indexing of video data can be focused on activities, actions which are higher level abstractions. Automatic indexing of video data is desirable since manual indexing is hard and indexes that are created may differ with respect to the indexers. Low-level features, which can be extracted from video data without user intervention, have been used in automatic indexing of video data [2, 3, 9, 17, 18, 19, 21]. However low-level features are not sufficient enough to index video data based on higher level abstractions. There are a number of works [2, 9] where low-level features are used to index higher level abstractions but these works are specific to some domain of videos like news videos or sports videos.

Many works try to index video data based on low-level features. In [16], an interactive workbench is provided supporting the researcher in the development of new movie content analysis algorithms. Low-level features of video are supplied to the researcher and by combining these low-level features, he/she tries to design an algorithm, which can be used for indexing high level video data.

A browseable/searchable paradigm is proposed in [3] for organizing video databases. Automatic indexing approach is used to classify shots into pre-determined semantic classes. The semantic classes are broadly defined semantic categories that can be derived from low-level features. The semantic classes correspond to head and shoulders, man-made versus natural, indoor versus outdoor which cannot model high level semantic content of video.

A probabilistic framework for semantic indexing and retrieval of video is proposed in [17, 18]. The framework uses multijects and multinet where multijects are multimedia objects representing semantic features and concepts, and a multinet is a probabilistic network of multijects that accounts for the interaction between concepts. Video is segmented into shots and shots are segmented into regions, which are labeled by a person. Then, low-level features of the regions are extracted. The regions (multijects) are then related to other multijects by a factor graph framework that uses sum-product algorithm in a tree. In this work importance is given to the relations between semantic concepts. For example, detection of ‘beach’ increases the chances of detecting ‘water’. Detection of ‘speech’ in the audio content and detection of a ‘face’ in the video frame may infer the event ‘human talking’. The multijects represent general concepts and they are not capable of supporting activities and objects in detail. In the ‘human talking’ event, data such as the name of the person or the subject of the talk are not supported.

A video browsing system is proposed in [11, 12] which accepts conceptual queries from users. Conceptual terms are extracted from low-level features. The retrieval engine calculates relevance values for the results of a conceptual query by feature aggregation on video shot granularity to offer conceptual, content-based access. For example, ‘artificial light’ is specified by the user to retrieve indoor shots. High-level semantic information is not addressed by this work.

An algebraic video model is proposed in [22] which assigns content information to logical video segments. In this way, the same data can have multiple annotations. A video algebra is also proposed that consists of operations for temporally combining video segments to generate a desired video stream. In this

work, no model is defined for activities, actions, and objects of interest taking place in video.

Close-captioned text has been used to index and model semantic information. In [23], a basic keyword set is annotated from the close-caption of video. An electronic lexical system (Word-Net) is used to provide semantic association. The authors also apply TOC (Table of Content) and index logic to videos in [21]. Video is structured into a hierarchy based on low-level features and keywords are associated with the levels in the hierarchy. The usage of a lexical system is a good idea in query processing especially when finding words with the same meaning. However, this work does not provide a semantic video model that will abstract activities, actions and interesting objects.

A spatio-temporal model is proposed in [4] to model semantic information of video. Modeling events by using spatio-temporal attributes of objects is performed but this can only be used for specific domains like sports videos. A ‘pass event in a soccer game’ can be modeled by using spatio-temporal attributes but a ‘party event’ cannot be modeled in this way.

A metadata database and a query interface for video is proposed in [20], where bibliographic, structural, and content indexes are identified. Bibliographic data stores metadata about video. Structural data stores information about shot, scene, and segments of video. Content data stores keywords and objects. In this video model, objects are associated with actions but activities are not supported. Attributes cannot be defined for objects.

A semantic video model is proposed in [1]. Video entities are specified as activities, events, and video objects. In this model, video is divided into fixed-time duration frame sequences. Activities, events and objects are related to the frame sequences and these relations are modeled by using a frame segment tree and arrays that store activities, events and video objects. For retrieval of the semantic data, different query types are defined such as object, activity, event, and compound queries. The proposed data model and the algorithms for handling different types of queries were implemented within a prototype, called Advanced Video Information System (AVIS). In this model, objects have no attributes other

than role in the event. Dividing video into fixed-size time intervals is not a good solution for temporal modeling of video. The query language proposed cannot answer temporal queries.

In [14, 15], video is modeled using spatial attributes of objects. Common Video Object Tree (CVOT) model is proposed. In this model, all common objects among video clips are found and video clips are grouped according to these common objects. This data model is integrated into a temporal object model to provide concrete object database management support for video data. The salient objects and shots are recognized by a video analyzer. Spatio-temporal queries are supported by this system. Temporal attributes for events and objects are supported by storing history of events and objects. Semantic attributes for objects and roles for activities are not addressed by the system.

In [10], a data model and a rule-based query language is developed for video content-based indexing and retrieval. The data model is designed around the object and constraint paradigms. The data model consists of feature and content layer and semantic layer. The semantic layer includes objects, attributes of objects, and relations between objects. The query language can be used to infer relationships about information represented in the model. Queries can refer to both the feature-content layer and the semantic layer.

Chapter 3

Video Database System

In this chapter, a Web-based video database system proposed in [5, 6, 7, 8] to which the work in this thesis will be integrated, is described. The organization of this chapter is as follows: The architecture of the video database system is given in Section 3.1. The spatio-temporal data model proposed for video data type is described in Section 3.2. Types of queries that can be answered by the system are given in Section 3.3. The query processing approach is explained in Section 3.4.

3.1 Video Database System Architecture

Figure 3.1 illustrates the overall architecture of the target Web-based video database system. The proposed system is built on a client-server architecture. Users access the video database on the Internet through a Java client Applet.

Spatio-temporal relations of objects in video frames are extracted from video clips by *Fact-Extractor* and these relations are stored in *Knowledge-Base* as Prolog-type predicates. *Feature Database*, which stores data about semantic content of videos, is populated by *Video Annotator*. Feature Database and Video Annotator constitute part of this thesis work and detailed explanation on these

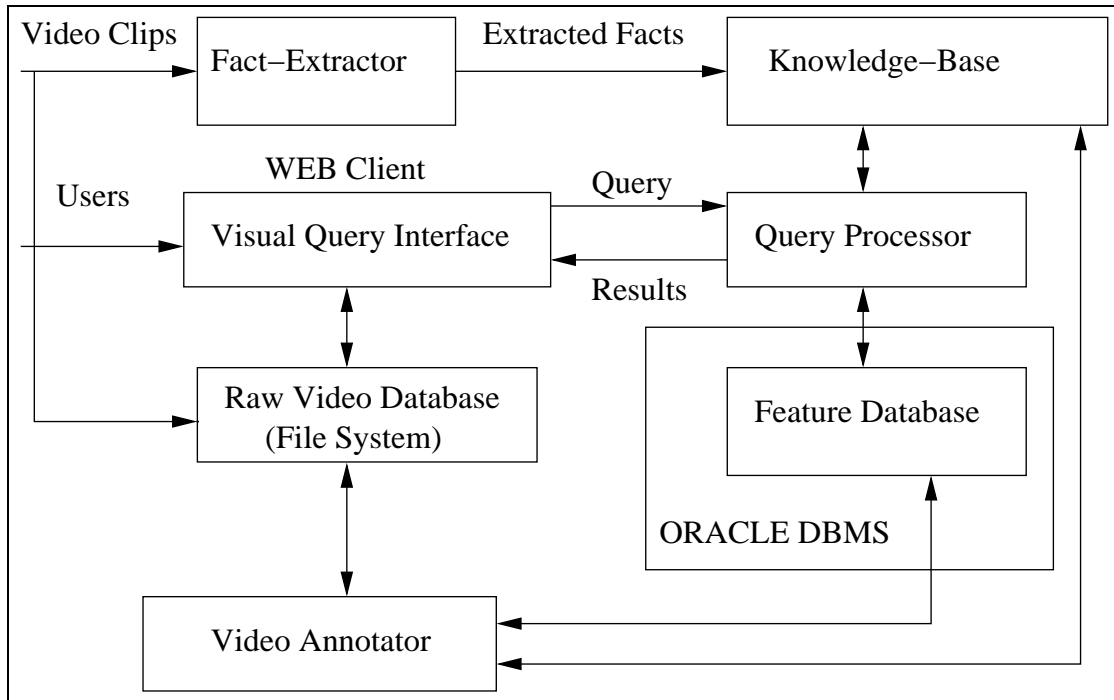


Figure 3.1: Video Database System Architecture.

components are given in Chapters 4 and 5, respectively. In the heart of the system lies the query processor, which is responsible for answering user queries in a multithreaded environment. Spatio-temporal, relative object-motion, object-appearance and similarity-based object-trajectory queries are supported by this system, which uses the knowledge-base to answer these types of queries. By using spatio-temporal relations, a restricted set of events can also be specified as query conditions. Users may query the system with sketches. A visual query is formed by a collection of objects with different attributes including object-trajectory with similarity measure and spatio-temporal ordering of objects in time. Motion is sketched as an arbitrary polygonal trajectory for each query object. Users are able to browse the video collection before posing complex and specific queries. A text-based SQL-like query language is also available for the experienced users.

3.2 Video Data Model

A new approach is proposed for segmentation of video into shots where segmentation is done using spatial relations of objects in video frames. Spatial relations can be grouped into mainly three categories: *topological relations* that describe neighborhood and incidence, *directional relations* that describe order in space, and *distance relations* that describe range between objects. Spatial relations are extracted using the minimum bounding rectangles of salient objects that are specified manually by a Java application developed, called *Fact-Extractor*.

In a video, a shot is detected when the set of spatial relations between objects changes. The frames where the change occurs are used as an index for the shots and are considered as keyframes. Spatial relations are called spatio-temporal relations because they do have a time component represented by frame numbers of keyframes. Spatio-temporal relations for the shots are stored, on a keyframe basis, as Prolog facts in a knowledge-base. Inference rules are used to reduce the number of facts stored in the knowledge-base since some facts could be derived using other facts stored. The benefit of storing facts in a knowledge-base is the reduction of the computational cost of relation computation during the time of query processing since computation of relations are done a priori.

3.3 Query Language

The query language is a text-based SQL-like language designed and implemented to support spatio-temporal queries on video data. Different query types that can be specified by the query language are as follows:

- *Object queries*: this type of queries may be used to retrieve objects, along with video segments where the objects appear.
- *Spatial queries*: this type of queries may be used to query videos by spatial properties of objects defined with respect to each other. Supported

spatial properties for objects can be grouped into mainly three categories: topological relations that describe neighborhood and incidence in 2D-space, directional relations that describe order in 2D- space, and 3D-relations that describe object positions on z-axis of the three dimensional space.

- *Similarity-based object-trajectory queries:* this type of queries may be used to query videos to find paths of moving objects.
- *Temporal queries:* this type of queries is used to specify the order of occurrence for conditions in time. Supported temporal operators, which are used in temporal queries, are before, meets, overlaps, starts, during, finishes and their inverse operators.
- *Aggregate queries:* this type of queries may be used to retrieve statistical data about objects and events in video data. The three aggregate functions are *average*, *sum*, and *count*. These functions may be very attractive in collecting statistical data for such applications as sports event analysis systems.

3.4 Query Processing

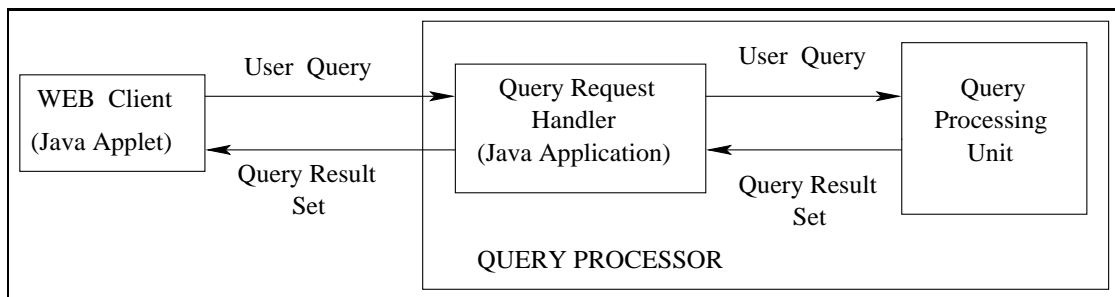


Figure 3.2: Web client - query processor interaction.

Figure 3.2 illustrates how the query processor communicates with Web clients and the underlying system components to answer user queries. Web clients send user queries, transformed into SQL-like text-based query language expressions if visual queries are given, to the query processor. Query processor is responsible for retrieving and responding to user queries in a multithreaded environment. The

queries are reconstructed by the query processor as Prolog-type knowledge-base queries. Results returned from the knowledge-base are sent to Web clients. The phases of query processing for spatio-temporal queries are shown in Figure 3.3 and can be briefly described as follows.

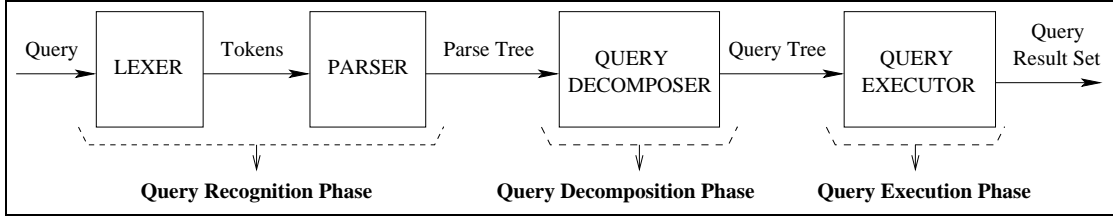


Figure 3.3: Query processing phases.

1. *Query parsing*: The lexical analyzer partitions a query into tokens, which are passed to the parser with possible values for further processing. The parser assigns structure to the resulting pieces and creates a parse tree to be used as a starting point for query processing. This phase is called *query recognition phase*.
2. *Query decomposition*: The parse tree generated after the query recognition phase is traversed in a second phase, which is called *query decomposition phase*, to construct a query tree. Queries are decomposed into three basic types of subqueries: *plain Prolog subqueries* or *maximal subqueries* that can be directly sent to the inference engine Prolog, *trajectory-projection subqueries* that are handled by the trajectory projector, and *similarity-based object-trajectory subqueries* that are processed by the trajectory processor.
3. *Query execution*: The query tree is then traversed in preorder in the next phase of query processing, *query execution phase*, executing each subquery separately and performing interval processing so as to obtain the final set of results.
4. *Interval processing*: Results returned by subqueries are further processed in the internal nodes of the query tree to compute the final result set for user queries.

In order for the system to support semantic video queries, a semantic video model and semantic extensions to the query language are needed. Extraction of semantic information from video data and querying this data are the focuses of the work done in this thesis. This work will be used in the video database system explained so that the resulting system is able to answer semantic video queries in addition to spatio-temporal, relative object-motion, object appearance, and similarity-based object-trajectory queries on video data.

Chapter 4

Semantic Video Model

Modeling is necessary for efficient management and retrieval of video. Semantic video modeling is the translation of video data into an internal representation, which captures the semantic content of video and creates indexes for efficient retrieval.

In this chapter, a video model is proposed to model video semantic content. Video has two layers; the first one is the *feature and content layer* that deals with the low level details of video, and the second one is the *semantic layer* that deals with the meaning perceived by humans from a video. A semantic video model should capture events, subevents, objects of interest and bibliographic data about video. An event is an instance of an activity that may involve many different objects over a time period. Subevents are used not only to model relations between objects but also to detail the activity (event) into actions. Actions are the acts performed by living objects. Data that is related to video in general, but not related to video content, such as name, year of production, producer and etc., is specified as bibliographic data about video.

This chapter is organized as follows: In Section 4.1, our semantic video model is defined. The database model, which is used to support the semantic video model, is described in Section 4.2.

4.1 The Data Model

Video consists of sequence of images in a time-line. The main difference between video and image is the temporal structure of video. Temporal structure provides video with semantic content that cannot be derived from single images. While two people dancing in a ballroom can be depicted by an image, how long they have danced and which figures they have played can only be captured by a video. For example, a goal event in a soccer game can only be described by a sequence of images, which corresponds to video.

Video consists of events. Events are the instances of activities taking place in video. In other words, activities are the abstractions of events. For example, wedding is an activity, but wedding of Richard Gere and Julia Roberts in a movie, is an event. Activities can be thought of as classes, and events can be thought of as the instances of these classes. For each activity type, a number of roles are defined. For example, murder is an activity. Murder activity has two roles defined for it: *murderer* and *victim*. The murder of Richard Gere by Julia Roberts is an event where Richard Gere has the role ‘victim’ and Julia Roberts has the role ‘murderer’.

Subevents are used to detail events and to model the relations between objects of interest. The difference between events and subevents can be better explained with an example. Assume that a party is depicted in a video. The party is modeled as an event that may contain a number of subevents. Subevents may be specified as follows:

- drinking, a group of people may drink,
- eating, a group of people may eat,
- dancing, some couples may dance, and
- talking, another group may talk about a subject.

Several objects of interest can take place in this party event. These objects

are assigned roles, which may be defined as ‘Host’ and ‘Guest’ for the party event. Actions represented by subevents, such as drinking or eating, are performed by living objects. These imply that objects are not only assigned roles defined for the event, but also they are associated with subevents, where they perform the actions represented by subevents. To sum up, events, subevents, objects, and bibliographic data form the abstraction of video semantic content, which can be categorized in three groups as follows:

- *bibliographic or metadata*: data about video,
- *object data*: data about the objects of interest in video,
- *event data*: activities and actions taking place in video.

Video name, duration, producer, director, video type, audience and subject of video are classified as bibliographic data. The attributes of interesting objects and values for the attributes can be stored in object data whereas data related to events and subevents can be stored in event data. Type of activity, begin and end times, objects that take part in event, roles for objects, location and time can be described as event specific data. Subevent specific data can be given as follows: type of subactivity, begin and end times, and objects that appear in subevent.

4.1.1 Hierarchical Structure

A video is modeled as a hierarchy of events, subevents and objects of interest. A hierarchical model provides many semantic levels that facilitate understanding of video content. Video consists of events and events consist of subevents. Moreover, objects are modeled in every level in the hierarchy. In the semantic video model, segmentation of video into sequences and scenes is performed by specifying events and subevents of video since events are associated with sequences and subevents are associated with scenes.

4.1.2 Temporal Management

Temporal management of video segments can be categorized into three groups: *segmentation*, *stratification* and *temporal cohesion*. Segmentation splits video into independent and contiguous time segments, which allows one level of segmentation to be specified. Stratification allows overlapping of time segments, which provides many levels of segmentation to be performed. In temporal cohesion, a time segment is defined as a set of non-overlapping time intervals and this provides many levels of segmentation and accurate representation of video segments. Temporal cohesion, which allows accurate temporal representation of time segments, is used in our semantic video model. Events and subevents are the time segments in our model. Video consists of events, which may overlap. Events consist of subevents, which may not be contiguous in time. Scenes may also overlap. These features provide flexibility in modeling activities and actions in video.

4.2 Database Design

A database model is required to have proper database management support for the semantic video model. With respect to the specifications of the semantic video model, a database model, which consists of fifteen database tables, is constructed to store the semantic data of videos. The conceptual design of the database is presented in Figure 4.1. The specification of the tables can be found in Appendix C.

TVIDEO table stores the bibliographic information about videos. The remaining database tables can be categorized into three groups:

1. *Utility tables:* TAUDIENCE, TVIDEOTYPE, TACTIVITY, TACTIVITYROLE, TSUBACTIVITY, and TATTRIBUTE can be grouped as utility tables. Utility data such as audiences, video types, activity types, roles for activity types, subactivity types and object attributes are stored in these tables.

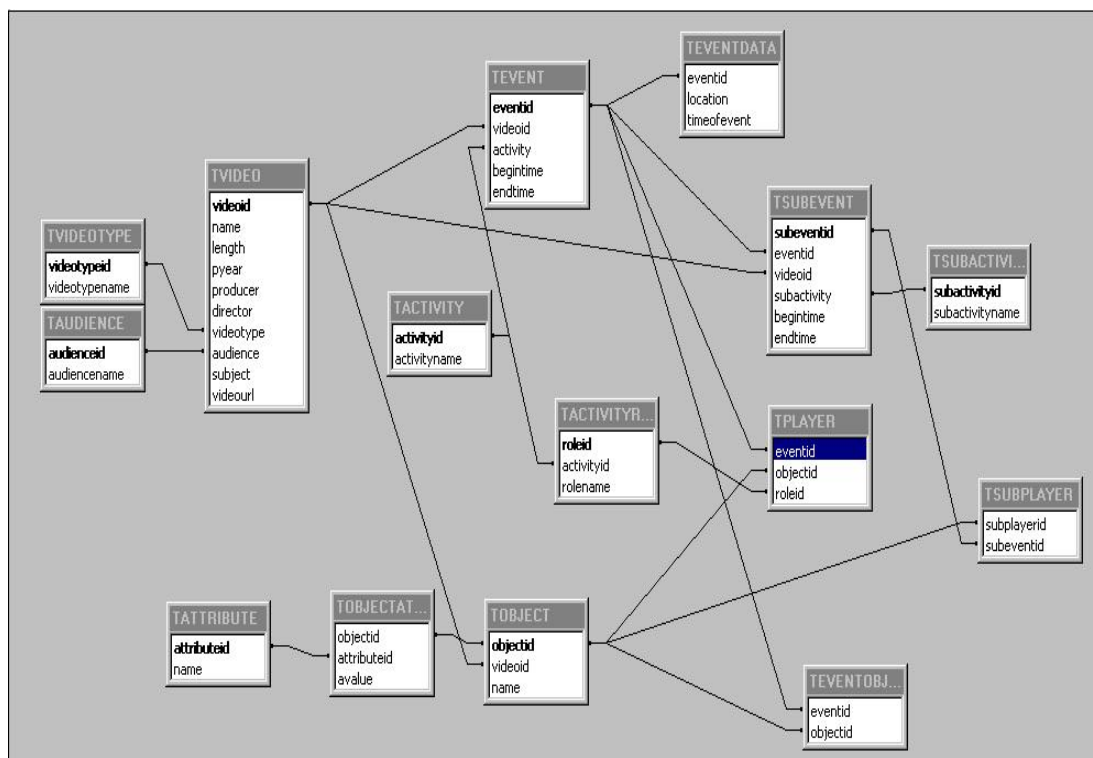


Figure 4.1: Database design of the semantic video model.

2. *Object tables*: Data about the interesting objects in videos are stored in **TOBJECT** and **TOBJECTATTRIBUTE** tables.
3. *Event tables*: Database tables that store data related to events and subevents are **TEVENT**, **TEVENTDATA**, **TEVENTOBJECT**, **TPLAYER**, **TSUBEVENT**, and **TSUBPLAYER**.

Chapter 5

Annotation Tool

The video annotation tool developed is a database application coded in Java, which is used for annotating video clips according to the semantic video model designed. The tool has two main functionalities. First of all, it is a movie player, which can play video files in different movie formats. Java Media Framework Application Programming Interface (JMF API) has been used for the implementation of the player part of the tool. JMF API allows video, audio and other time-based media to be used within Java applications and applets. In the annotation tool, opening a video for playing, closing a video, navigating inside a video and viewing the details of a video, such as length of a video and format of a media stream, could be done by using the functionalities provided by JMF API. The second main functionality provided by the tool is annotating videos by their semantic content. The tool is used to extract semantic data about videos for annotation as well as to view, update and delete the semantic data that has been extracted before. Semantic data extracted from a video may be categorized into five groups as follows:

1. *Metadata about a video*: Metadata contains the data specific to video, such as video name, length of video, year of production, etc. The annotation of this data is referred as *video metadata annotation* and is explained in Section 5.3.

2. *Object data*: Object data is formed by items of interest in video. The annotation of object data is referred as *object annotation* and is described in Section 5.6.
3. *Event data*: Data related to activities that take place in video is considered as event data, and annotation of this data is called *event annotation*, which is described in Section 5.7.
4. *Subevent data*: Data related to actions that take place in activities is considered as subevent data and annotation of this data is called subevent annotation, which is described in Section 5.8.
5. *Utility data*: Utility data consists of audiences, video types, activities, activity roles, sub-activities and object attributes. The annotation of the utility data is explained in Section 5.3.

The tool is explained through the annotation of a sample video clip in Sections 5.1-5.8. The rest of this chapter is organized as follows: In Section 5.1, initialization of the tool and general information about the main user interface is given. The order of annotation is discussed in Section 5.2. In Section 5.3, video metadata annotation is explained. Section 5.4 describes the annotation of utility data. In Section 5.5, hierarchical video tree that is used to show the annotation status is explained. Section 5.6 describes the annotation of video objects, and event annotation is explained in Section 5.7, which is followed by the discussion of subevent annotation in Section 5.8. Section 5.9 describes the functionality of the buttons used with hierarchical video tree. Section 5.10 gives the implementation details of the tool. Java class definitions of the video components used in the tool are given in Appendix B.

5.1 Initialization Process and the Main User Interface

An initialization procedure is executed as soon as the video annotation tool is started. During the initialization, JDBC (Java Database Connectivity) driver class is loaded into the Java Virtual Machine (JVM). JDBC is an API that manages database connections and database operations (queries) within the Java programming language. After the driver is loaded, ‘utility’ data is read from the video database and then main program window, shown in Figure 5.1, is launched.

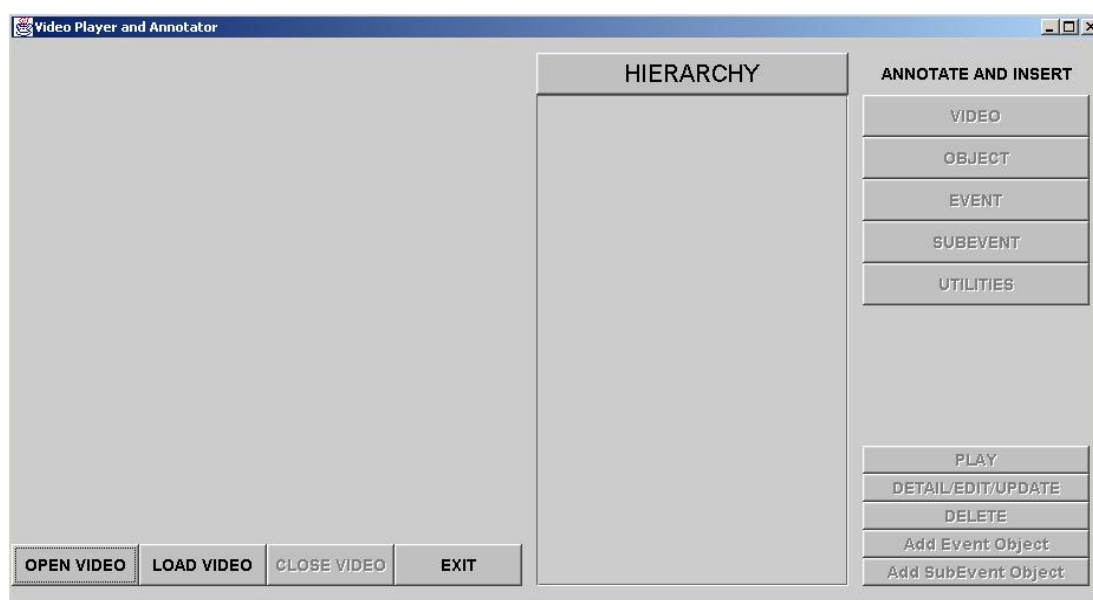


Figure 5.1: The main user interface of the annotation tool.

Video is loaded and played in the video player area at the left-hand-side of the main window. To the right of this, there is a rectangular area, where the hierarchy of an annotated video is displayed as a hierarchical video tree. Moreover, there are also a number of buttons placed in two groups at the right-hand-side of the main window. The upper group consists of ‘video’, ‘object’, ‘event’, ‘subevent’ and ‘utilities’ buttons, which are used to open new windows to annotate corresponding video components. For example, when ‘video’ button is clicked, another window of the tool, referred to as ‘video window’, is launched. Video window is used to annotate video metadata. When ‘object’ button is clicked, yet another window

referred to as ‘object window’ is launched for the user to annotate interesting objects in video. The functionality of the lower group of buttons is explained in Section 5.9.

Initially, ‘open video’, ‘load video’ and ‘exit’ buttons are enabled. A video is opened and played by using the ‘open video’ button. Main window with a video playing in the video player area is demonstrated in Figure 5.2.



Figure 5.2: The main user interface with a video file opened.

As soon as a video is opened, the upper group of buttons is enabled. The annotation of the video is done using these buttons in a pre-determined order that is described in Section 5.2.

5.2 The Order of Annotation

The order of annotation should follow the hierarchical semantic model of video from top to bottom. Hence, video is annotated first according to the hierarchy. Annotation of events with their corresponding subevents may be accomplished afterwards. During the annotation process, annotation of objects may be carried

out whenever needed. However, the annotator must comply with the following restrictions:

- Utility data annotation is required for video metadata, event, subevent and object annotations.
- Event and object annotation cannot be done before video metadata annotation.
- Event annotation cannot be done before the annotation of objects of that event.
- A subevent cannot be annotated before the annotation of the event to which it is associated.

The annotation of utility data can be done at any time. However, utility data is required for the annotation of video metadata, events, subevents and objects. For example, video type and audience information are required during video metadata annotation. The next annotation that should be done is video metadata annotation because event and object annotations depend on video metadata annotation. Event annotation cannot start immediately after video metadata annotation since for an event annotation to be complete, with event objects and roles of the objects specified, annotation of objects in the event must be done before the annotation of that event. Subevent annotation must be associated with an event annotation as well; if the event annotation is not done, then subevent annotation is not possible.

With the above details, the annotation is performed as follows. First of all, utility data and video metadata annotation is done. Then, for an event ‘e1’, object annotation for the objects in event ‘e1’ is performed followed by event annotation for ‘e1’. Subevent annotations for ‘e1’ are performed last. Subsequent annotations of other events should follow the same order of annotation (object \rightarrow event \rightarrow subevent). The order of upper group of buttons in the main window from top to bottom also reflects this order of annotation.

5.3 Utility Data Annotation

The ‘Utility Window’, where all the utility data can be seen and updated, is shown in Figure 5.3.

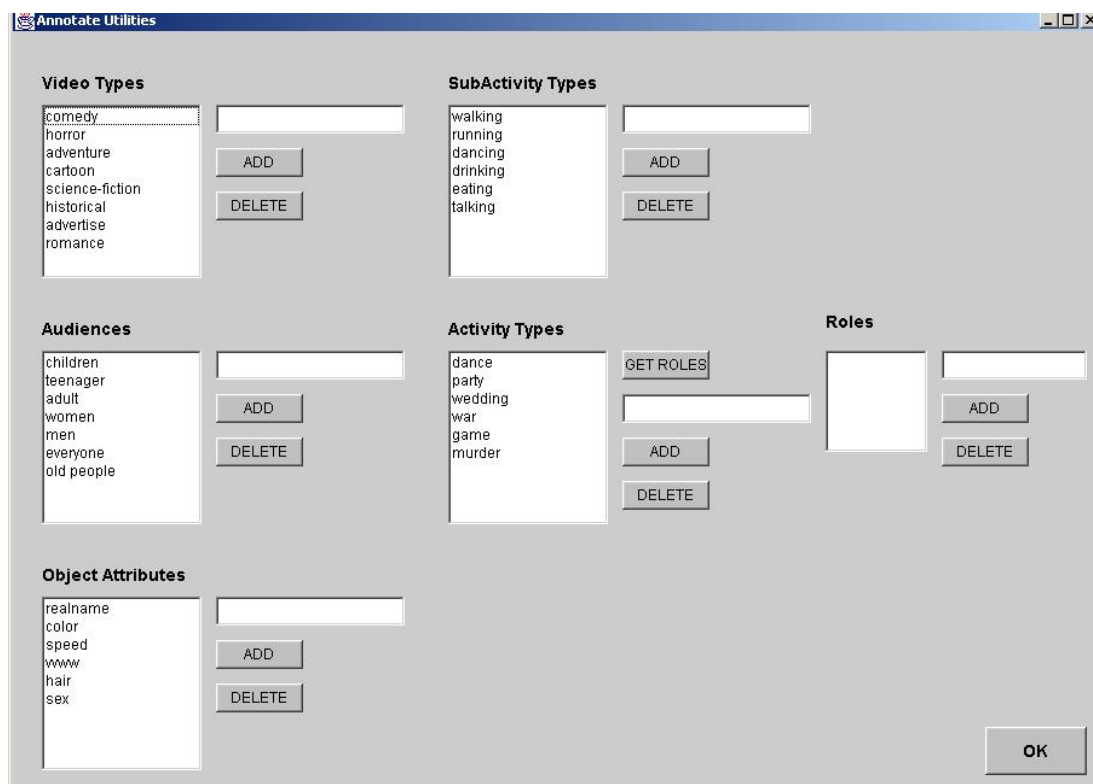
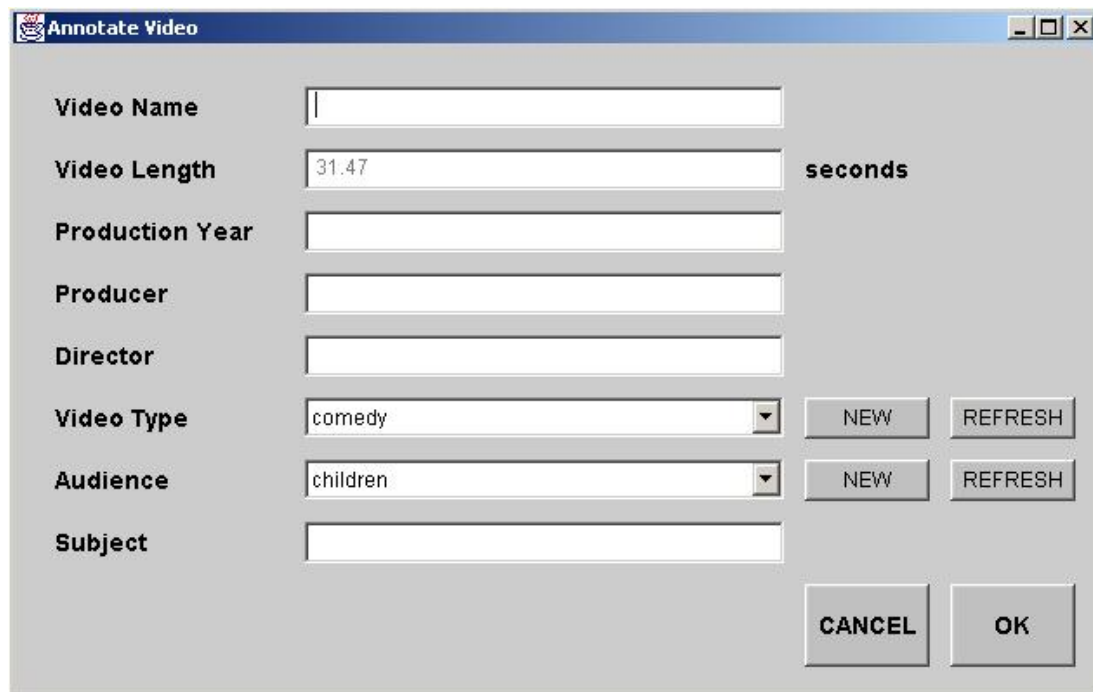


Figure 5.3: Utility window.

Video types, audiences, object attributes subactivity types, activity types and roles for activity types can be added and deleted by the annotator using this window.

5.4 Video Metadata Annotation

Semantic annotation of a video starts with video metadata annotation. This annotation is done by the ‘video window’ that appears when ‘video’ button is clicked. The video metadata annotation process is given in Figure 5.4.



The screenshot shows a window titled "Annotate Video" with a standard Windows-style title bar. Inside the window, there are several labeled input fields and buttons. The fields are: "Video Name" (a text box), "Video Length" (a text box containing "31.47" with the unit "seconds" to its right), "Production Year" (a text box), "Producer" (a text box), "Director" (a text box), "Video Type" (a dropdown menu showing "comedy"), "Audience" (a dropdown menu showing "children"), and "Subject" (a text box). To the right of the "Video Type" and "Audience" dropdowns are "NEW" and "REFRESH" buttons. At the bottom right of the window are "CANCEL" and "OK" buttons.

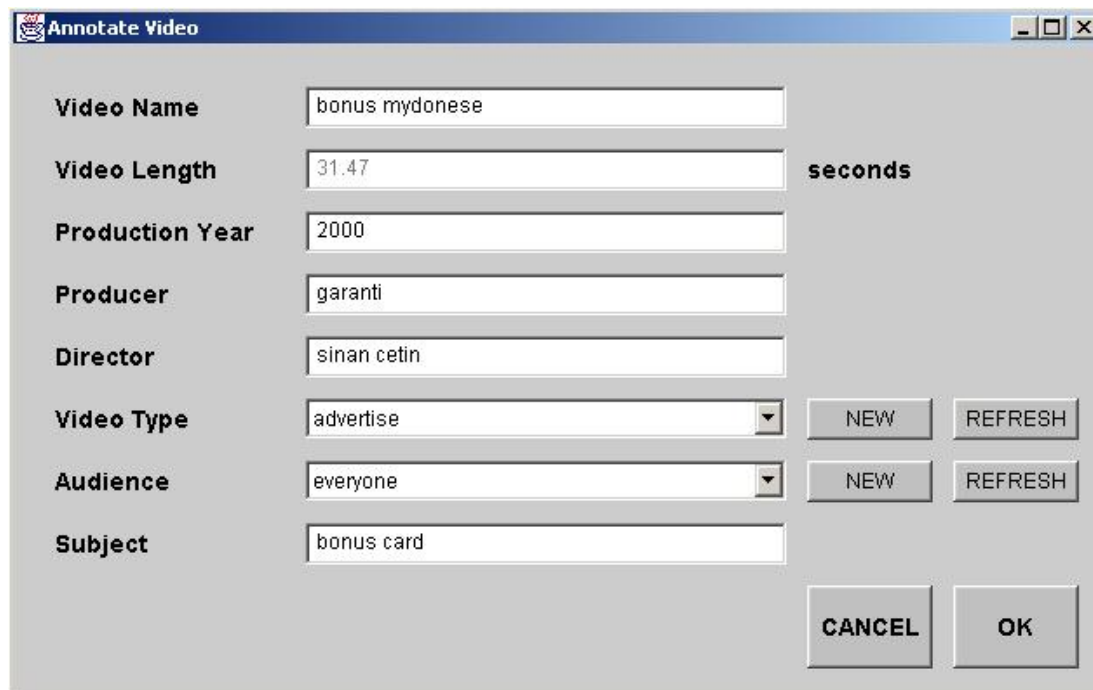
Figure 5.4: Video metadata annotation process.

Using this window, video specific data can be annotated. Video length is automatically retrieved from video player. ‘Video Type’ and ‘Audience’ fields can be selected from a list of choices. The ‘New’ button, when clicked, shows the ‘Utilities window’, where utility data is annotated. ‘Refresh’ buttons refresh the items in the lists of video type and audience. The ‘Video Window’, when the data specific to a video is inserted, is shown in Figure 5.5.

5.5 Hierarchical Video Tree

Video is modeled as a hierarchy in the semantic video model. The hierarchical video tree is used to show the current annotation status. The following rules define the hierarchical video tree (see Figure 5.6):

- Root of the tree is a video entry.
- The leaves of a video entry are events and video objects.



The screenshot shows a window titled "Annotate Video" with a standard Windows-style title bar (minimize, maximize, close buttons). The window contains several input fields and buttons:

- Video Name:** A text box containing "bonus mydonese".
- Video Length:** A text box containing "31.47", followed by the label "seconds".
- Production Year:** A text box containing "2000".
- Producer:** A text box containing "garanti".
- Director:** A text box containing "sinan cetin".
- Video Type:** A dropdown menu showing "advertise". To its right are two buttons: "NEW" and "REFRESH".
- Audience:** A dropdown menu showing "everyone". To its right are two buttons: "NEW" and "REFRESH".
- Subject:** A text box containing "bonus card".
- At the bottom right, there are two large buttons: "CANCEL" and "OK".

Figure 5.5: Video window with data inserted.

- The leaves of an event are event objects and subevents.
- The leaves of a subevent are subevent objects.
- The leaves of a video object are the attributes and their values.
- The leaves of an event object are the roles of the event object in the activity.
- Subevent objects have no children.

When video metadata is inserted, the root of the hierarchy tree is constructed. Hierarchical video tree and the current state in the annotation can be viewed by clicking the 'hierarchy' button. Figure 5.7 shows the 'Main Window' with the current hierarchy tree.

The hierarchy tree has only the root node, which is for the video entry. For identification, video name is rendered near the video entry as well. Besides, the lower group of buttons is enabled. These buttons are 'play', 'detail/edit/update', 'delete', 'add event object' and 'add subevent object'. The functionalities of these buttons are explained in Section 5.9.

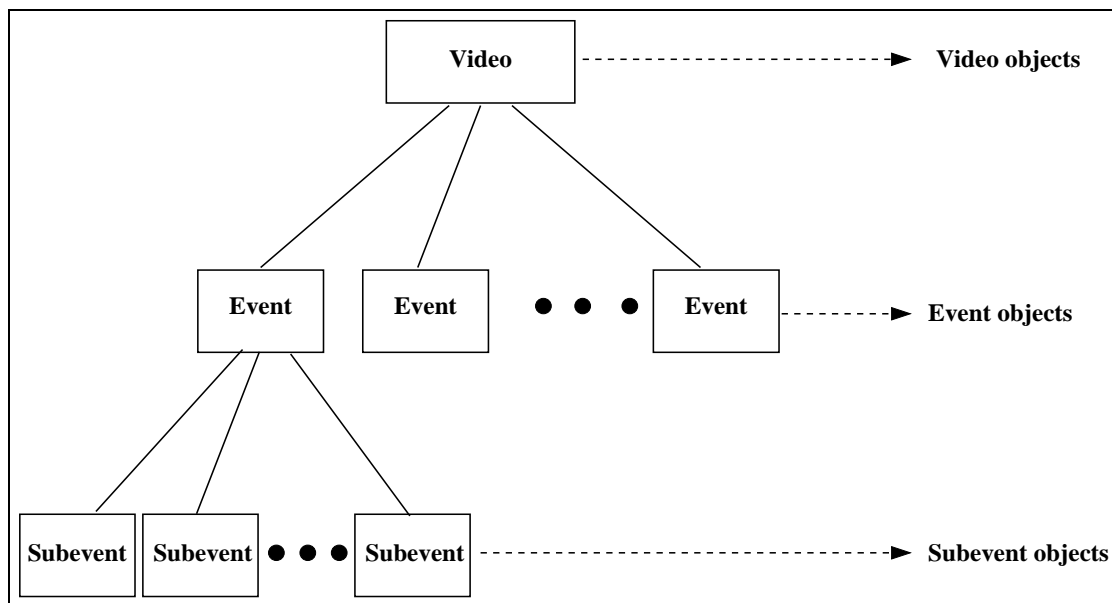


Figure 5.6: Hierarchy tree of a video.



Figure 5.7: The user interface with hierarchy tree after video metadata annotation.

5.6 Object Annotation

For an event annotation, the objects that appear in that event should be annotated first. ‘Object window’, which appears when the object button is clicked, is shown in Figure 5.8. In this figure, four objects are inserted into the video, and attributes and values of ‘object1’ are listed as well.

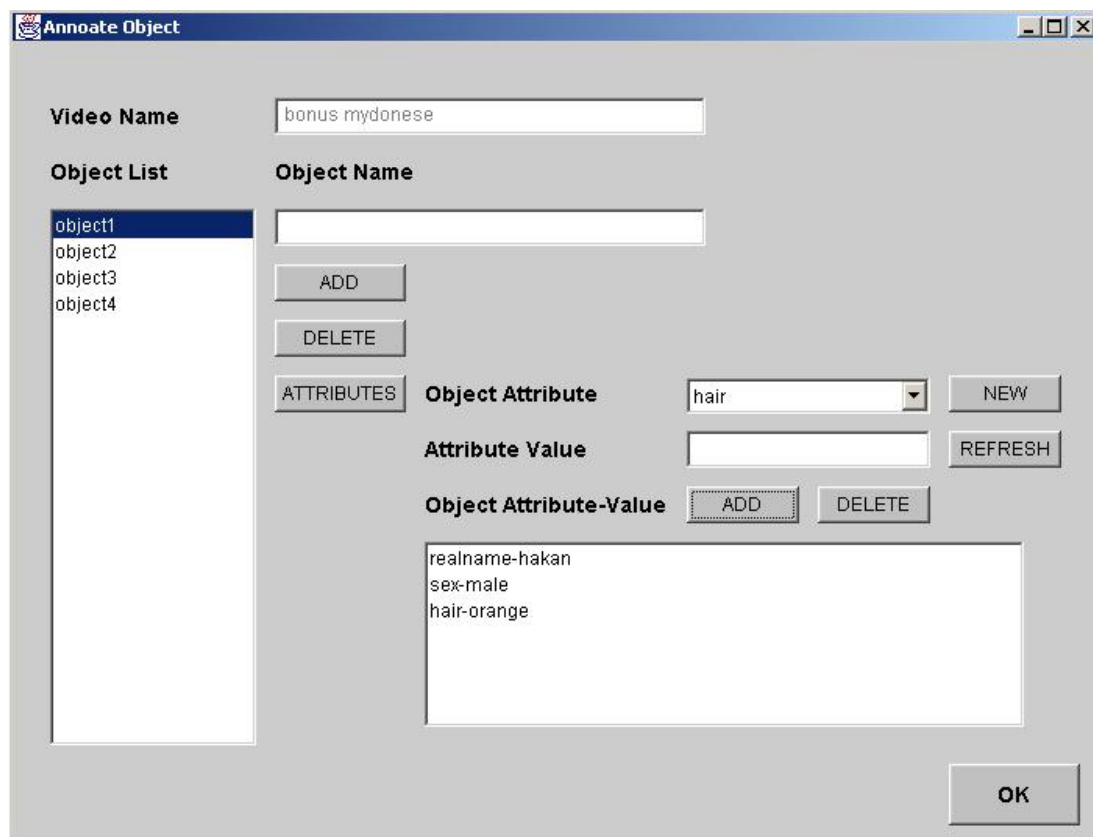


Figure 5.8: Object Window with data inserted.

Using the ‘object window’, objects can be added to and deleted from the video. Attributes for objects and values for the attributes can be defined or deleted. Only one value can be defined for each attribute. In the ‘video name’ field, the name of the current video, that is being annotated, is displayed for information. Selecting an object from the object list and clicking the ‘attributes’ button lists all the attributes and values defined for the attributes. Clicking the ‘new’ button brings up the ‘utilities window’, where the annotator can define new object attributes. ‘Refresh’ button refreshes the items in object attribute

list, which is required as new attributes are defined. The hierarchy tree of video, after some video objects are defined, is shown in Figure 5.9.



Figure 5.9: The user interface with video tree hierarchy after object annotation.

In Figure 5.9, the hierarchy tree with video entry as the root node and four objects as four children of the root node is displayed. Object1 has three attributes defined, namely `realname`, `sex` and `hair`. Values for the attributes are 'hakan', 'male' and 'orange'. All data that has been annotated up to that point is displayed in the hierarchy tree, which provides a friendly user interface to the annotator.

5.7 Event Annotation

The next step following the object annotation is the annotation of an event. For an event annotation, three windows are used. In the first window, event specific data is entered, as shown in Figure 5.10.

In Figure 5.10, party event has been defined, whose time frame spans from second 0.08 to second 19.74 in the sample video clip. To obtain the time of a location in a video, video is sought to the desired location and paused. Time of that location in the video stream is retrieved from the video player by using the

Annotate Event

Video Name

Event Type

Role list

Start Time

End Time

Event Location

Time of Event

Figure 5.10: Event annotation, event specification process.

‘Get Time’ button. Event type is selected from a list of choices. Roles of the selected event can be retrieved and listed by clicking the ‘Show Roles’ button. New event types (activities) and roles can be defined in the utilities window, which appears when ‘new’ button is clicked. Location and time of an event are also defined. When ‘Ok’ button is clicked, event data is inserted into the database and the second window of event annotation is launched while the current window is closed. In the second window, selecting from a list of video objects specifies objects that appear in the event. This process is demonstrated in Figure 5.11.

Video name and event type are displayed for information. In this annotation example, there are four objects defined for the video and two of which have been selected to appear in the event. New objects for the video can be defined by clicking ‘new’ button, which launches the ‘object window’ for object annotation. When ‘Ok’ button is clicked, this window is closed and the last window for the event annotation is shown. The last window is used to define roles for the event objects, as demonstrated in Figure 5.12.

In this window, video name and event type are given, and roles defined for

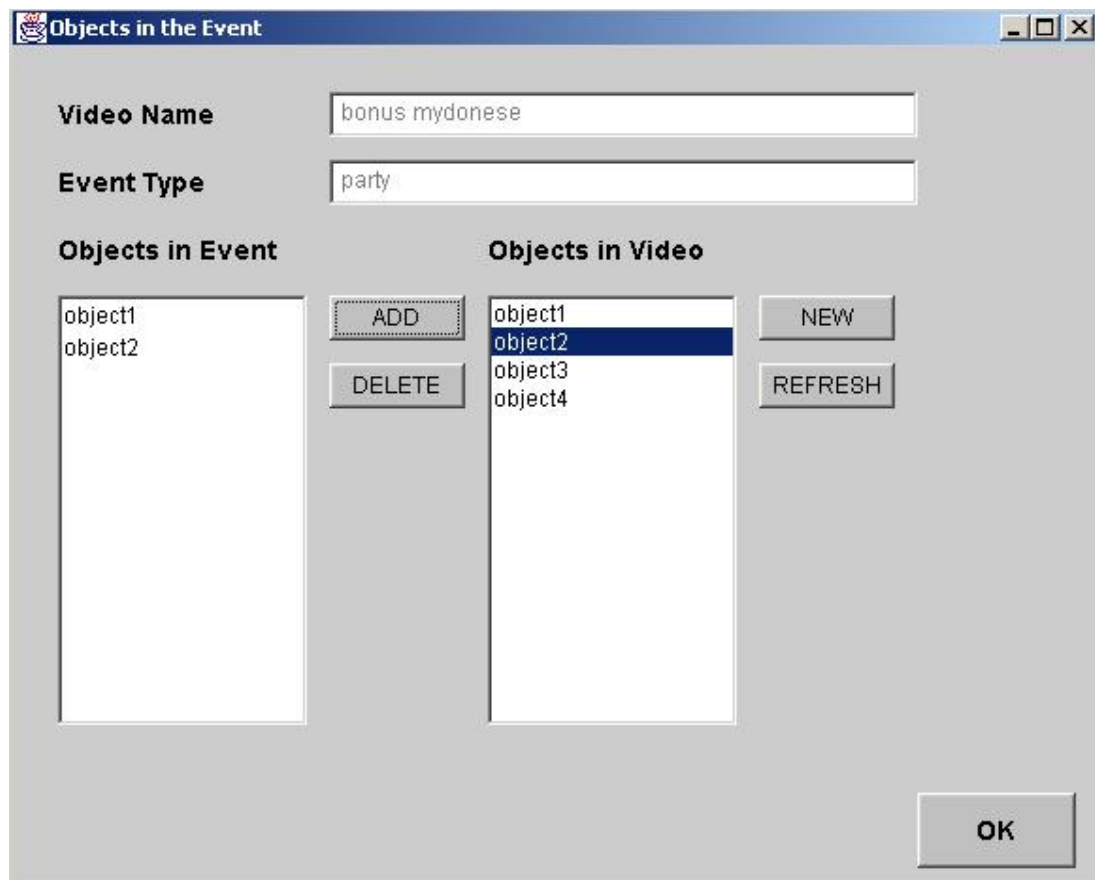


Figure 5.11: Event annotation, object selection process.

The screenshot shows a window titled "Annotate Team" with a standard Windows-style title bar. Inside the window, there are two text input fields at the top: "Video Name" containing "bonus mydonese" and "Event Type" containing "party". Below these are three list boxes. The "Players-Roles" list box contains "object1-host" and "object2-guest". To its right are "ADD" and "DELETE" buttons. The "Objects in Event" list box contains "object1" and "object2", with "object2" selected. Below it are "NEW" and "REFRESH" buttons. The "Role List" list box contains "host" and "guest", with "guest" selected. Below it are "NEW" and "REFRESH" buttons. An "OK" button is located at the bottom right of the window.

Video Name	Event Type	Players-Roles	Objects in Event	Role List
bonus mydonese	party	object1-host object2-guest	object1 object2	host guest

Figure 5.12: Event annotation, role definition process.

the event type (activity) are also displayed together with the objects appearing in the event. The annotator has to match the objects with roles. One object may be associated with more than one role in the event. The ‘New’ button below the role list brings up the ‘utilities window’, and the ‘New’ button below the object list launches the second window of the event annotation, where the objects that appear in the event are specified. The hierarchy tree after the event annotation is completed is displayed in Figure 5.13.

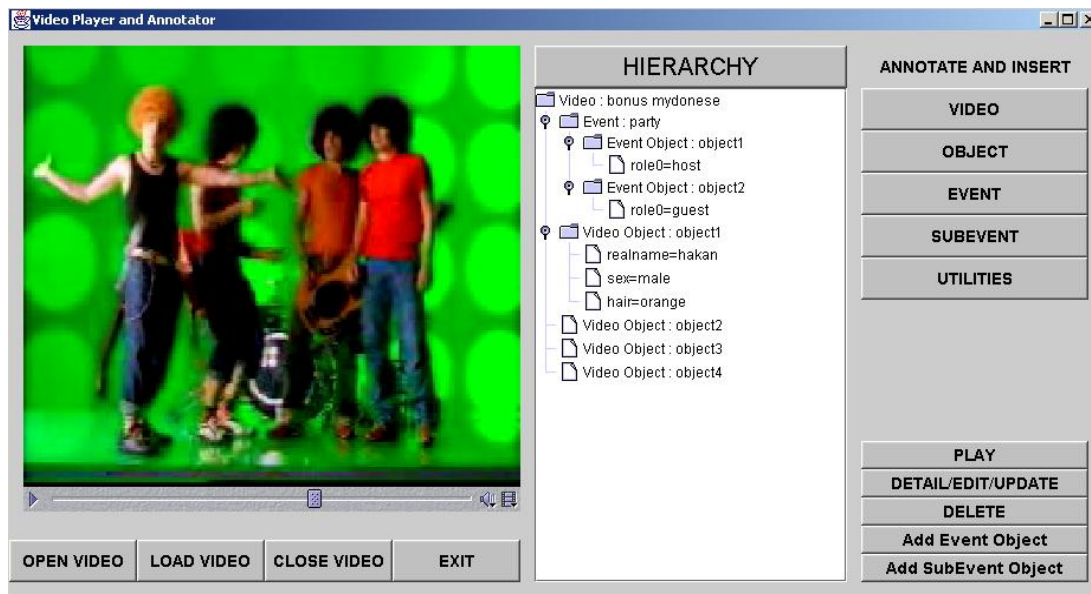
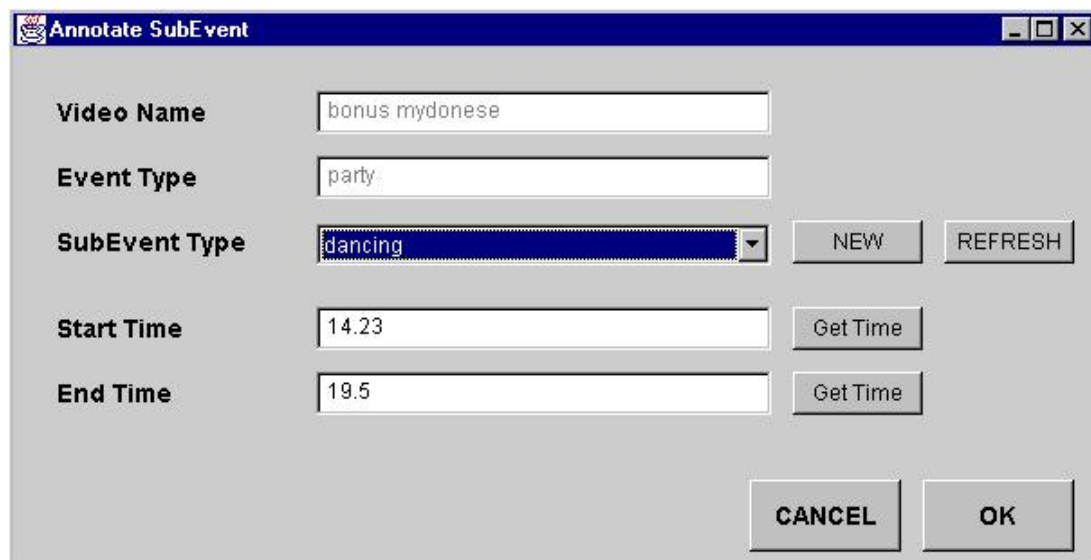


Figure 5.13: The user interface with hierarchy tree after event annotation.

In the hierarchy tree, the root node (video entry) has five children. The party event is also inserted as a child of the video entry. This event has two children, which are event objects, and these event objects have their role names as their children. Since event objects may have many roles defined for them, roles are displayed as ‘role0’, ‘role1’, and ‘role2’ if there is more than one role defined for an event object.

5.8 Subevent Annotation

The ‘Subevent’ button must be clicked to start the annotation of subevents. Two windows are used in subevent annotation. The first window is used to insert subevent specific data whereas the second window is used to select the objects appearing in the subevent. Figure 5.14 shows the first subevent window.



The screenshot shows a window titled "Annotate SubEvent". It contains the following fields and controls:

- Video Name:** A text box containing "bonus mydonese".
- Event Type:** A text box containing "party".
- SubEvent Type:** A dropdown menu with "dancing" selected. To its right are "NEW" and "REFRESH" buttons.
- Start Time:** A text box containing "14.23". To its right is a "Get Time" button.
- End Time:** A text box containing "19.5". To its right is a "Get Time" button.
- At the bottom right are "CANCEL" and "OK" buttons.

Figure 5.14: Subevent annotation, subevent specification process.

Video name and event type are displayed to provide information to the annotator. Event type is used to show into which event the subevent is being inserted. Subevent type is selected from a list of choices, and the ‘new’ button is used to define new subevent types. Start and end times are specified in the same way as it is done in the event annotation. When ‘Ok’ button is clicked, this window is closed and the second window of the subevent annotation is launched, which is demonstrated in Figure 5.15.

Video name and subevent type are displayed for information. Subevents are associated with events. Therefore, objects that can appear in subevents are only those objects that appear in the event, with which the subevent is associated. In the window shown in Figure 5.15, event objects are displayed for the annotator to select which of them appear in the subevent. In the example annotation, both of the event objects appear in the subevent ‘dancing’. The ‘New’ button will pop

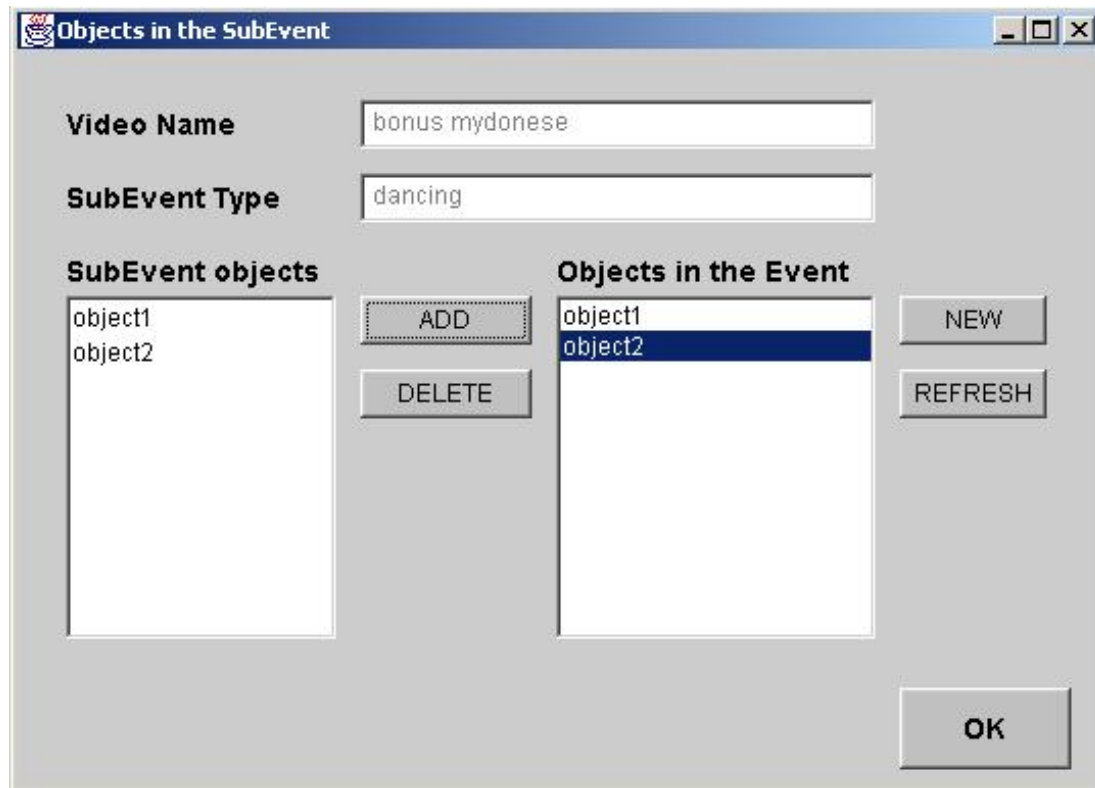


Figure 5.15: Subevent annotation, object selection process.

up the second window of the event annotation, where objects for the event are defined. When ‘Ok’ button is clicked, annotation of ‘dancing’ subevent, which is associated with the party event, is completed. The hierarchy after this annotation is displayed in Figure 5.16.

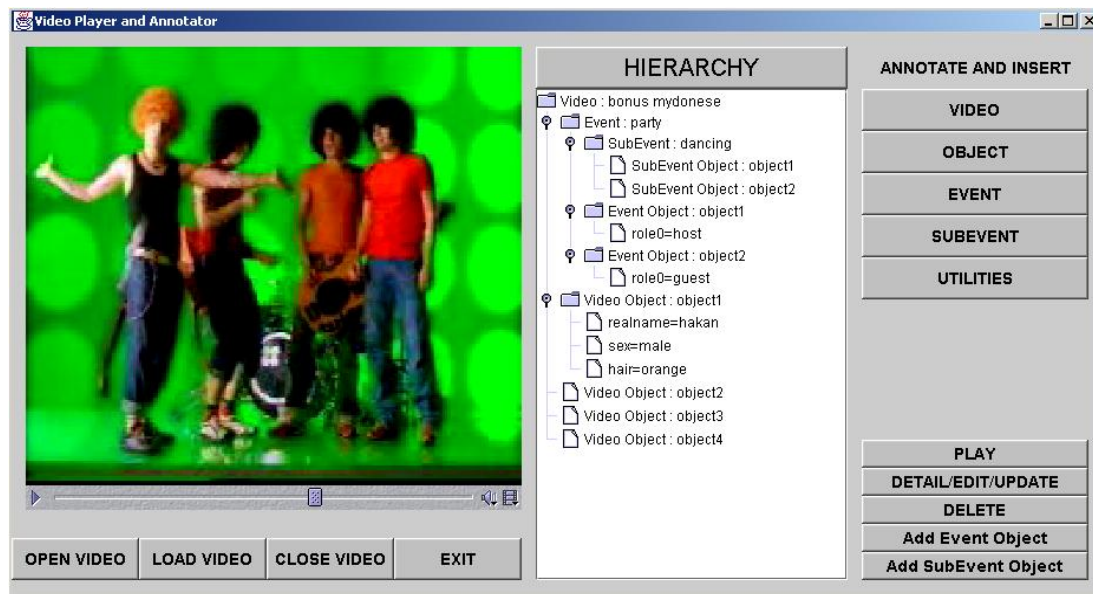


Figure 5.16: The hierarchy after subevent annotation process.

In the hierarchy tree, party event has a new child, which is a subevent whose subevent type is dancing (cf. Figure 5.16). This subevent has two subevent objects. New events, objects and subevents can be annotated in the same way.

5.9 Hierarchical Video Tree Functionality

Annotation of all video components has been explained in the preceding sections. In this section we describe the functionality of the lower group of buttons, which work with the hierarchical video tree. These buttons are ‘play’, ‘detail/edit/update’, ‘delete’, ‘add event object’ and ‘add subevent object’.

1. *Play button:* This button is used to play a selected video component in the video tree if it is playable. Playable video components are video, event and

subevent. The play button makes the video player seek to the beginning location of the playable video component and starts to play it from that location until the end of the playable video component. If video is selected in the hierarchy and play button is clicked, video is sought to the beginning of the video. If an event is selected, video player seeks to the beginning location of the event and plays it and stops at the ending location of it.

2. *Detail/Edit/Update button*: This button is used to show the details of a selected video component in the video tree. Details of video components are shown in the same windows, where they are annotated. If a video is selected and the detail button is clicked, then the video window will appear with the video specific data inserted in the corresponding fields. If an event is selected, then the event window will appear with the event specific data for that event. In the detail windows, data editing can be done, and when the ‘Ok’ buttons are clicked, the data updated in the detail windows is updated in the database as well. Therefore, detail windows are used for editing/updating of annotation data. The windows designed for the annotation of video components are used for the annotation, detail and update operations. This provides uniformity and standardization in the annotation tool.
3. *Delete button*: This button deletes a selected video component both from the hierarchy and the database. If a video is selected and deleted, all the annotation data that is entered for that video is lost.
4. *Add Event Object button*: This button is a quick link to the second window of the event annotation, where objects for the event are specified. The data for the event selected in the video tree is retrieved into the second window.
5. *Add Subevent Object button*: This button is also a quick link to the second window of the subevent annotation, where objects for the subevent are defined. The data for a subevent selected in the video tree is retrieved into the second window.

The ‘add event object’ and ‘add subevent object’ buttons are used because addition of an object to an event requires the annotator to pass the first window of the event annotation, where event specific data is defined. Using the update button, a new event object can be defined. Nonetheless, the annotator must first see the first window of the event annotation, after which an event object can be defined in the second window. To eliminate the viewing of the first windows in the event and subevent annotations, these quick link buttons are defined.

Up to now, functionality of the annotation tool used for the annotation of a new video is explained. The next part to discuss about is the loading of the annotated video. The annotator need not complete the annotation of a video when he/she starts to annotate it. The annotator can stop the annotation process at any time, and can continue to annotate after. This functionality is also provided in the annotation tool.

When the load video button is clicked, a list of video names that have been annotated before is shown to the annotator. One video is selected, and the program loads that video from the database. When the hierarchy button is clicked, the hierarchical video tree for that video appears. From then on, the annotator can continue the annotation and can update the existing annotation.

5.10 Implementation Details

There are a number of implementation details that have not been explained so far about the annotation tool. First of all, this tool works with one video file at a time. However, since this program is an application, multiple copies can be executed at the same time.

For the connection to the database, JDBC is used. When the program is initialized, the JDBC connection is established, and this connection is used until the execution of the program ends. For each database access, the same connection is used, which eliminates the need for reconnecting to the database. This provides better performance for the database operations.

When an exception occurs in the execution, an error window is launched. For example, when begin and end times of an event are the same, it prompts the annotator to correct the times. When a subevent's begin and end times are out of the range for the event that it is associated with, then it prompts the annotator stating that subevent begin and end times must be between the event begin and end times.

There is a window controller. Only one copy of one type of window can be open at any time. This means that the annotator cannot open multiple copies of the same window at the same time. Hierarchical video tree is displayed as follows:

- For videos, video name is displayed.
- For events, activity type is displayed.
- For subevents, subactivity type is displayed.
- For subevent objects, object name is displayed.
- For event objects, object name is displayed.
- For video objects, object name is displayed.
- For roles of event objects, role name is displayed.
- For attributes of video objects, attribute name and attribute value are displayed.

Annotated events and subevents are listed in a sorted order in the hierarchical video tree. Events of video are sorted, and so are subevents of all events. Sorting criterion adopted takes into account the beginning times of events and subevents. When an update occurs to the beginning time of an event or subevent changing the order, this change is reflected in the video tree as well.

Annotation of subevents, event objects, and subevent objects need not follow an order. While annotating a subevent, normally it is associated with the last

event that was annotated. However, the annotator can select the event with which the subevent will be associated. This gives some flexibility to the annotator. The annotator can select an event or a subevent in the hierarchical video tree, and add an object to that event or subevent by using the ‘add event object’ and ‘add subevent object’ buttons.

When an annotation is completed, it is immediately reflected in the video database. The annotation tool also stores the current annotated video in a Java class called ‘Video’. The object instantiated from the video class is a mirror of the database and is used in the construction of the hierarchical video tree as well as in the execution of the program. This video object is used because retrieving data from the database for each operation is unnecessary. Instead, the annotation tool makes queries to the database only when it is really needed. For example, if the annotator annotates a video, and then, wants to see the details of the video, the program retrieves this information from the video object instead of retrieving from the database since video object and the entries in the database for that video are the mirrors of each other. This provides better performance in the execution of the program. There are some cases when database connection is necessary. When the annotator annotates the video specific information, the program first queries the database and creates an entry in the database for the video. If no problem is encountered in this operation, then a video object is created and the video specific data is inserted into the video object. In this way, both video object and the database contain the same data. Here, the synchronization problem is crucial: if some problem occurs in the database operation, this operation must not be executed by the video object. If video object is thought as a local copy of the database for that video, then the above operation can be summarized as follows: First the database operations must be performed. After that, if there is no problem, local operations could be performed. The Java class definitions for video class and related classes are given in Appendix B.

Chapter 6

Semantic Query Language

Having defined semantic video model and implemented video annotation tool to extract semantic data from video clips, we should provide retrieval methods to query video data by its semantic content. A query language similar to SQL was designed to specify semantic queries on video. Semantic queries specified by this language can be used to retrieve videos/video segments based on video metadata, events, actions and objects of interest in video. The organization of this chapter is as follows: In Section 6.1 an introduction to the query language is provided. Explanation of condition types for semantic queries is provided in Section 6.2, where metadata, event and object conditions are described in detail. Grammar specification of the language is provided in Appendix A.

6.1 Features of the Language

A query statement format similar to SQL was defined for retrieval of video data by its semantic content. The format of the query statement is as follows:

```
select target from range where condition;
```

In the above statement, **target** in **select** clause represents the output type of a query, which can be specified as video or segments of video. In the semantic

video model, segments of video are defined as sequences or scenes, where sequences are associated with events and scenes are associated with subevents of video. The range of a query, which is specified by **range** in **from** clause, can be a video or a list of videos. Conditions for a query are specified in **where** clause. Conditions can be categorized in three groups: *metadata conditions*, *event conditions* and *object conditions*. Output types of queries depend on the conditions specified.

Supported operators are logical and temporal operators, which are used to join conditions. Logical operators are ‘and’, ‘or’ and ‘not’. Temporal operators join conditions on a time basis and they are explained in Section 6.2.2.4.

Semantic queries are defined as follows:

```
<query> := select videos from <range> where <condition1> ';'
| select sequences from <range> where <condition2> ';'
| select scenes from <range> where <condition3> ';'

```

```
<range> := all | <vidlist>

```

```
<vidlist> := <vid> | <vid> ',' <vidlist>

```

The set of conditions valid for each output type (video, sequence, scene) are different. Before the explanation of this difference, condition types are given in Section 6.2.

6.2 Types of Conditions

Condition types for semantic queries are categorized into three groups as follows: *metadata conditions*, *event conditions*, and *object conditions*. Metadata conditions query video according to the metadata specified for video such as video name, video length, production year, etc. Event conditions specify queries related to events/activities and subevents/actions. Object conditions are used to

specify conditions about interesting objects in video. The types of conditions are examined in Sections 6.2.1-6.2.3, where the output types for the queries with these conditions are also mentioned.

6.2.1 Metadata Conditions

These conditions correspond to the metadata defined for videos such as the length of a video, title of a video, type of a video, etc. Since these attributes are associated with video in the semantic video model, video is the output type for the queries with these types of conditions. An example query with a metacondition is given in as follows:

“Find all videos from the database, which are comedies produced in 2000 that are less than 90 minutes of length and directed by either Yilmaz Erdogan or Sinan Cetin”.

```
select videos
from all
where meta(vtype:comedy and
           pyear:2000 and
           length:90 and
           director:Yilmaz Erdogan or director:Sinan Cetin))
```

6.2.2 Event Conditions

Event conditions correspond to events and subevents that take place in a video. Events consists of subevents where events are associated with sequences and subevents are associated with scenes. The output type for queries with event conditions can be video, sequence or scene according to the type of the event condition specified. Event conditions can be categorized in three groups:

1. event conditions without subevent condition,

2. event conditions with subevent conditions, and
3. subevent conditions.

6.2.2.1 Event Conditions without Subevent Condition

This condition type is used to specify event conditions without any subevent condition. Event type, event metadata (time, location) and role conditions can be given. Role conditions are used to query events with the specified object in the given role. An example query for this condition type is listed as follows:

“Find all videos that has a wedding event with two objects (players) with roles bride and groom where bride is Julia Roberts and groom is Hugh Grant and the event takes place in London in 2000”.

```
Select videos
from all
where etype:wedding with
    (object1:role=bride and object1(name:Julia Roberts) and
    object2:role=groom and object2(name:Hugh Grant) and
    location:London and
    time:2000)
```

The output type for queries specifying event conditions without subevent conditions are video and sequence since events are associated with sequences. Videos that consist of sequences can also be specified as the output type.

6.2.2.2 Event Conditions with Subevent Conditions

This condition type is used to specify event conditions with subevent conditions. Event type, event metadata (time, location), role conditions and subevent conditions can be given. The query in Section 6.2.2.1 revised by joining one subevent condition is given as follows:

“Find all videos that has a wedding event with two objects (players) with roles bride and groom who are dancing where bride is Julia Roberts and groom is Hugh Grant and the event takes place in London in 2000”.

```
Select videos
from all
where etype:wedding with
    (object1:role=bride and object1(name:Julia Roberts) and
     object2:role=groom and object2(name:Hugh Grant) and
     location:London and
     time:2000 and
     etype:dancing with object1,object2)
```

The output type for queries specifying event conditions with subevent conditions are video, sequence and scene since events are associated with sequences and subevents are associated with scenes. Videos that contain these scenes and sequences can also be specified as the output type.

6.2.2.3 Subevent Conditions

Subevent conditions specify actions in events. A list of players (objects) that take place in the action can also be specified in the condition. Users may query the video database by specifying subevent conditions without specifying event conditions. An example of a subevent condition may be given as follows:

“Find all videos where Julia Roberts and Hugh Grant are dancing”.

```
Select videos
from all
where etype:dancing with object1,object2 and
    object1(name:Julia Roberts) and
    object2(name:Hugh Grant)
```

The output type for queries specifying subevent conditions are video, sequence and scene since subevents are associated with scenes and sequences, videos associated with these scenes can also be specified as output type.

6.2.2.4 Temporal Conditions

Temporal queries are used to specify the order of occurrence for event conditions in time. The three groups of event conditions described previously can be temporally joined. The query language implements all temporal relations as temporal operators defined by Allen's temporal interval algebra. Temporal conditions are defined as follows:

`<eventcondition> <temporalconnector> <eventcondition>`

`<temporalconnector> := overlap | before | during |
 starts | finishes | meets |
 ioverlap | ibefore | iduring |
 istarts | ifinishes | imeets`

The meanings implied by temporal operators are given as follows:

- *overlap*: two events overlap in a video.
- *before*: an event comes before the other in a video.
- *during*: an event happens during another event.
- *starts*: an event starts another event.
- *finishes*: an event finishes another event.
- *meets*: an event finishes and another event starts at the same time.

The temporal operators that start with 'i' are the inverse operators. The output types for temporal queries are decided according to the event condition types that are temporally joined.

6.2.2.5 Event Conditions for Videos

All types of event conditions can be specified when output type for queries is videos. The event conditions can also be temporally joined. To explain event conditions more accurately, ‘general event conditions’ is defined as event conditions with or without subevent conditions.

6.2.2.6 Event Conditions for Sequences

All types of event conditions can be specified as event conditions for output type sequences; however, not all types of event conditions can be joined temporally. Subevent conditions can be temporally joined to other subevent conditions or general event conditions. Yet, general event conditions cannot be joined temporally with other general event conditions since the output type of the query is videos. This restriction may be demonstrated with a query example in written English as follows:

“Find all videos where there is a wedding and a party event such that wedding event comes before party event”.

In this query, two general event conditions are specified, which are connected temporally, and the output type of the query is videos. Sequences or scenes cannot be the output type because the event conditions point to two different sequences and these sequences can be modeled in the hierarchal semantic model at video level since videos contain sequences.

However there is one exception to this restriction. General events can be joined temporally only if the temporal operator is ‘during’. Events can overlap, and so can sequences. A general event can be defined inside another general event. This condition is clarified with the following query.

`<generaleventcondition> during <generaleventcondition>`

“Find all sequences where there is a robbery event during a wedding event”.

During a wedding event, someone steals something. Stealing is represented by a general event since it has nothing to do with the wedding event. Wedding sequence can be returned as the result of the query.

6.2.2.7 Event Conditions for Scenes

When the return type is scene, only subevent conditions can be specified.

6.2.3 Object Conditions

These conditions correspond to the objects (living or nonliving) that are of interest in video. Object attributes can be specified in object conditions. A query example for object conditions is given as follows:

“Find all scenes where Brad Pitt, Julia Roberts and a Dalmatian dog appears”.

```
Select scenes
from videos
where odata (object1 (type:man, name:Brad Pitt) and
              object2 (type:woman, name:Julia Roberts) and
              object3 (type:dog , kind:dalmatian))
```

6.3 Semantic Conditions

Semantic conditions are constructed by joining meta, event and object conditions logically and/or temporally. Specification of semantic conditions differentiate with respect to different output types which is depicted as follows:

```
<query> := select videos from <range> where <condition1> ' ;'
```

```
| select sequences from <range> where <condition2> ';'
| select scenes from <range> where <condition3> ';'

```

```
<range> := all | <vidlist>

```

```
<vidlist> := <vid> | <vid> ',' <vidlist>

```

6.3.1 Semantic Conditions for Output Type Videos

All conditions can be specified and joined. A recursive structure has not been used for specifying conditions. Instead, conditions are defined as combinations of groups. Metaconditions stay as a group and they are connected to object or event group of conditions. This makes the queries structured and easy to read.

6.3.2 Semantic Conditions for Output Type Sequences

When the output type of queries is sequences, metaconditions cannot be specified alone since metaconditions are modeled in video level in the hierarchical semantic model and the output type for metaconditions is videos. When the output type is video, there is no restriction. Nevertheless, when the output type is sequence, queries cannot be joined to return video as their output type. Event conditions can be specified when their return type is sequence.

6.3.3 Semantic Conditions for Output Type Scenes

When the output type of queries is scenes, metaconditions cannot be specified alone. Event conditions that can be specified are the ones that return scene as their result.

Chapter 7

Conclusions and Future Work

Video data type has attracted more attention in the recent years and the number of application areas where video is used has increased. Consequently, more and more videos are created each day, and this fact leads to an enormous growth in the number of videos to be dealt with. Management of video has become an important problem and traditional database systems were not able to provide sufficient solutions.

Video has its own characteristics that differentiate it from other types of data. As a consequence, new methods have been proposed to manage video data efficiently. Management of video covers modeling, indexing and retrieval of video. In this thesis, management of video semantic information has been investigated, where semantic information is referred to as the meaning understood by human from video data. We have defined the content of the semantic information to include bibliographic data about video and events, actions, and objects of interest taking place in video.

We have proposed a semantic video model which models video semantic information in a hierarchy. Video consists of events, and events consist of subevents. Moreover, objects are modeled in every level in the hierarchy. A hierarchical model provides many semantic levels that facilitate understanding of video content. Temporal cohesion approach has been used to model time segments of

video, which provides flexibility and accuracy in modeling events and subevents. A database model has been constructed to have proper database management support for the semantic video model.

We have implemented an annotation tool in Java to extract the semantic information from videos, and to view and update semantic information that has already been extracted. The annotation tool is a database application that is used to insert and update semantic information extracted from videos to the database defined for the semantic video model. Since manual annotation of video content is a difficult process, extraction of information automatically is desirable, however automatic information extraction techniques are not powerful enough to model video semantic content. Human assistance is required in modeling video semantic information. The annotation tool simplifies the manual annotation process by providing simple and easy-to-understand user interfaces. The tool enables the annotator to see the current status of annotation in a hierarchical tree abstraction. Annotator is free to stop the annotation at any time and continue after without loss of data.

Finally, we have designed a query language to facilitate video retrieval according to semantic conditions. Three types of queries can be issued by the end-users. Video metadata queries are used to retrieve videos with respect to bibliographic data. Event queries are used to retrieve videos according to activities and actions that take place in video. Object queries are used to query videos according to objects of interest, their attributes and roles in activities.

Processing of semantic queries is the next step to complete the retrieval process of video semantic information. Query processor of the video database system described in Chapter 3 will be extended to handle semantic queries. Query processor will first separate semantic query conditions in a query from those conditions that could be handled by the knowledge-base. The semantic conditions will be organized and sent as regular SQL queries to the feature database where annotated semantic content is stored. Intermediate results returned from the feature database and the knowledge-base will be integrated by the query processor, and then, will be sent to the web-clients.

Web-clients send their queries by a Java client applet, which is used to access the video database system. The query interface of this applet will be extended to handle semantic data about video.

Our semantic model can be improved by extending the data stored for video, events and subevents. Video metadata can be extended to store other kinds of bibliographic data that we have not handled. Event specific data can be extended to store more information about events and subevents. For example for a ‘war’ event, name of the war can be stored to identify the war depicted in video. For a ‘talking’ subevent, ‘about’ field can be defined to mention the topic of the conversation. In short, data fields specific to event and subevent types can be defined. The semantic query language should be extended in parallel with the improvements of the semantic video model.

Annotation tool can be extended to facilitate the annotator with more functionality. One of these functionality can be a visual presentation, which shows the time segments of video components (event, subevent, object) graphically.

Finally, a library can be constructed that consists of video types, audiences, activity and subactivity types, roles for activities, and object attributes to facilitate the annotator in the annotation process. A dictionary can be merged to the system to handle words with the same or similar meaning.

Bibliography

- [1] S. Adalı, K.S. Candan, S. Chen, K. Erol, V.S. Subrahmanian. Advanced Video Information System: Data Structures and Query Processing. *Multimedia Systems Journal*, Vol.4 No.4, pages:172-186, ACM-Springer, August 1996.
- [2] G. Ahanger, T.D.C. Little. Data Semantics for Improving Retrieval Performance of Digital News Video Systems. *IEEE Transactions on Knowledge and Data Engineering*, Vol.13 No.3, pages:352-360, May-June 2001.
- [3] J. Chen, C. Taşkıran, E.J. Delp, C.A. Bouman. VIBE: A New Paradigm for Video Database Browsing and Search. *Proc. of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 12-15 1998, Seattle, WA.
- [4] Y.F. Day, A. Khokhar, A. Ghafoor. A Framework for Semantic Modeling of Video Data for Content-Based Indexing and Retrieval. *Multimedia Systems*, Vol.7 No.5, pages:409-423, 1999.
- [5] M.E. Dönderler, Ö. Ulusoy and U. Güdükbay, A Rule-Based Approach To Represent Spatio-Temporal Relations in Video Data, *Lecture Notes in Computer Science*, Vol.1909, pages:409-418, October 2000.
- [6] M.E. Dönderler, Ö. Ulusoy and U. Güdükbay, A Rule-Based Video Database System Architecture (Accepted for publication in the *Journal of Information Sciences* and also available as a Technical Report (BU-CE-0111) at Bilkent University Computer Engineering Department).

- [7] M.E. Dönderler, Ö. Ulusoy and U. Güdükbay, Rule-Based Spatio-Temporal Query Processing for Video Databases (Submitted to the VLDB journal).
- [8] M.E. Dönderler, A Rule-Based Spatio-Temporal Query Language for Video Databases, Ph. D. Progress Report #5, January 2002.
- [9] B. Günsel, M. Ferman, M. Tekalp. Video Indexing Through Integration of Syntactic and Semantic Features. *Proc. of the 3rd IEEE Workshop on Applications of Computer Vision (WACV)*, 1996.
- [10] M.S. Hacid, C. Declair, J. Kouloumdjian. A Database Approach for Modeling and Querying Video Data. *IEEE Transactions on Knowledge and Data Engineering*, Vol.12 No.5, pages:729-750, September-October 2000.
- [11] S. Hollfelder, A. Everts, U. Thiel. Designing for Semantic Access: A Video Browsing System. *Proc. of IEEE Int. Conf. on Multimedia Computing and Systems (ICMCS)*, pages:7-11, June 1999, Florence, Italy, *IEEE Computer Society*, Vol.1, pages:394-399, Los Alamitos, 1999.
- [12] S. Hollfelder, A. Everts, U. Thiel. Concept-based Browsing in Video Libraries. *Proc. of the IEEE Forum on Research and Technology Advances in Digital Libraries (IEEE ADL 99)*, May 19-21, 1999, Baltimore, MD, USA, *IEEE Computer Society*, pages:105-115, Los Alamitos, 1999, ISDN 0-7695-0219-9.
- [13] W. Khatib, Y.F. Day, A. Ghafoor, P.B. Berra. Semantic Modeling and Knowledge Representation in Multimedia Databases. *IEEE Transactions on Knowledge and Data Engineering*, Vol.11 No.1, pages:64-80 January-February 1999.
- [14] J.Z. Li, M.T. Özsu, D. Szafron. Modeling of Video Spatial Relationships in an Object Database Management System. *In Proc. of the International Workshop on Multimedia DBMSs*, pages:124-133, Blue Mountain Lake, NY, USA, 1996.

- [15] J.Z. Li, I.A. Goralwalla, M.T. Özsu, D. Szafron. Modeling Video Temporal Relationships in an Object Database Management System. *In Proc. of Multimedia Computing and Networking*, pages:80-91, San Jose, CA, USA, Feb 1996.
- [16] R. Lienhart, S. Pfeiffer, W. Effelsberg. The Moca Workbench. Support for Creativity in Movie Content Analysis. *Proc. of the 1996 International Conference on Multimedia Computing and Systems (ICMCS'96)*.
- [17] M.R. Naphade, I. Kozintsev, T.S. Huang, K. Ramchandran. A Factor Graph Framework for Semantic Indexing and Retrieval in Video. *Proc of the IEEE Workshop on Content-based Access of Image and Video Libraries (CBAIVL'00)*.
- [18] M.R. Naphade, I. Kozintsev, T.S. Huang, K. Ramchandran. A Probabilistic Framework for Semantic Indexing and Retrieval in Video. *IEEE International Conference on Multimedia and Expo*, New York, 31 July-2 August 2000.
- [19] J. Oh, K.A. Hua. Efficient and Cost-effective Techniques for Browsing and Indexing Large Video Databases. *SIGMOD Conference* pages:415-426 2000.
- [20] L.A. Rowe, J.S. Boreczky, C.A. Eads. Indexes for User Access to Large Video Databases, Storage and Retrieval for Image and Video Databases II, *IS&T/SPIE Symp. On Elec. Imaging Sci and Tech*, San Jose, CA February 1994.
- [21] Y. Rui, T.S. Huang, S. Mehrotra. Constructing Table-of-Content for Videos. *ACM Multimedia Systems Journal , Special Issue Multimedia Systems on Video Libraries*, Vol.7 No.5, pages:359-368 Sept 1999.
- [22] R. Weiss, A. Duda, D.K. Gifford. Content-Based Access to Algebraic Video. *Proc. IEEE First International Conference on Multimedia Computing and Systems*, Boston, MA, May 1994.
- [23] Y. Zhuang, Y. Rui, T.S. Huang, S. Mehrotra. Applying Semantic Association to Support Content Based Video Retrieval. *Content-based video retrieval Proc. of IEEE VLBV98 workshop*, pages:45-48, Urbana, IL.

Appendix A

Semantic Query Grammar Specification

```
<query> := select videos from <range> where <condition1> ';'
| select sequences from <range> where <condition2> ';'
| select scenes from <range> where <condition3> ';'

```

```
<range> := all | <vidlist>

```

```
<vidlist> := <vid> | <vid> ',' <vidlist>

```

```
// Condition specifications for return type videos

```

```
<condition1> := <metaconditions>
| <eventconditions1>
| <objectconditions>
| <metaconditions> <connector> <eventconditions1>
| <metaconditions> <connector> <objectconditions>
| <eventconditions1> <connector> <metaconditions>
| <eventconditions1> <connector> <objectconditions>
| <objectconditions> <connector> <metaconditions>
| <objectconditions> <connector> <eventconditions1>

```

```

| <metaconditions> <connector> <eventconditions1>
    <connector> <objectconditions>
| <metaconditions> <connector> <objectconditions>
    <connector> <eventconditions1>
| <eventconditions1> <connector> <metaconditions>
    <connector> <objectconditions>
| <eventconditions1> <connector> <objectconditions>
    <connector> <metaconditions>
| <objectconditions> <connector> <metaconditions>
    <connector> <eventconditions1>
| <objectconditions> <connector> <eventconditions1>
    <connector> <metaconditions>

// Condition specifications for return type sequences

<condition2> := <eventconditions2>
| <objectconditions>
| <metaconditions> <connector> <eventconditions2>
| <metaconditions> <connector> <objectconditions>
| <eventconditions2> <connector> <metaconditions>
| <eventconditions2> <connector> <objectconditions>
| <objectconditions> <connector> <metaconditions>
| <objectconditions> <connector> <eventconditions2>
| <metaconditions> <connector> <eventconditions2>
    <connector> <objectconditions>
| <metaconditions> <connector> <objectconditions>
    <connector> <eventconditions2>
| <eventconditions2> <connector> <metaconditions>
    <connector> <objectconditions>
| <eventconditions2> <connector> <objectconditions>
    <connector> <metaconditions>
| <objectconditions> <connector> <metaconditions>
    <connector> <eventconditions2>
| <objectconditions> <connector> <eventconditions2>
    <connector> <metaconditions>

```

```
// Condition specifications for return type scenes
```

```
<condition3> := <eventconditions3>
| <objectconditions>
| <metaconditions> <connector> <eventconditions3>
| <metaconditions> <connector> <objectconditions>
| <eventconditions3> <connector> <metaconditions>
| <eventconditions3> <connector> <objectconditions>
| <objectconditions> <connector> <metaconditions>
| <objectconditions> <connector> <eventconditions3>
| <metaconditions> <connector> <eventconditions3>
    <connector> <objectconditions>
| <metaconditions> <connector> <objectconditions>
    <connector> <eventconditions3>
| <eventconditions3> <connector> <metaconditions>
    <connector> <objectconditions>
| <eventconditions3> <connector> <objectconditions>
    <connector> <metaconditions>
| <objectconditions> <connector> <metaconditions>
    <connector> <eventconditions3>
| <objectconditions> <connector> <eventconditions3>
    <connector> <metaconditions>
```

```
// Metadata condition specifications
```

```
<metaconditions> := meta '(' <metaconditionlist> ')'
```

```
<metaconditionlist> := '(' <metaconditionlist> ')',
| not '(' <metaconditionlist> ')',
| <metaconditionlist> and <metaconditionlist>
| <metaconditionlist> or <metaconditionlist>
| <metacondition>
```

```
<metacondition> := vtype ':' <strvalue> // video type
```

```

| audience ':' <strvalue>
| title ':' <strvalue>
| length ':' <intvalue> // Maximum length (sthreshold) in minutes
| pyear ':' <intvalue> // production year
| producer ':' <strvalue>
| director ':' <strvalue>
| subject ':' <strvalue>

// Event condition specifications
// Event condition specification for videos

<eventconditions1> := '(' <eventconditions1> ')'
| not '(' <eventconditions1> ')'
| <eventconditions1> and <eventconditions1>
| <eventconditions1> or <eventconditions1>
| <temporalcondition1>
| <generaleventcondition1>

<generaleventcondition1> :=
    <generaleventconditions> | <subeventcondition>

<generaleventconditions> :=
    <generaleventconditionWsub> | <generaleventconditionW0sub>

<temporalcondition1>=<generaleventconditions>
    <temporalconnector><generaleventconditions>
| <generaleventconditions> <temporalconnector>
    <subeventcondition> [ with <where> event ]
| <subeventcondition> <temporalconnector>
    <generaleventconditions> [ with <where> event ]
| <subeventcondition> <temporalconnector>
    <subeventcondition> [ with <where> event ]

<where> := same | different

```

// Event condition specification for sequences

```

<eventconditions2> := '(' <eventconditions2> ')'
| not '(' <eventconditions2> ')'
| <eventconditions2> and <eventconditions2>
| <eventconditions2> or <eventconditions2>
| <temporalcondition2>
| <generaleventcondition2>

```

```

<generaleventcondition2> :=
    <generaleventconditions> | <subeventcondition>

```

```

<temporalcondition2> := <generaleventconditions>
                        during <generaleventcondition>
| <generaleventconditions> <temporalconnector>
    <subeventcondition> with same event
| <subeventcondition> <temporalconnector>
    <generaleventconditions> with same event
| <subeventcondition> <temporalconnector>
    <subeventcondition> with same event

```

// Event condition specification for scenes

```

<eventconditions3> := '(' <eventconditions3> ')'
| not '(' <eventconditions3> ')'
| <eventconditions3> and <eventconditions3>
| <eventconditions3> or <eventconditions3>
| <subeventcondition>

```

// General event condition specification with Subevents

```

<generaleventconditionWsub> := etype ':' <strvalue> with
    <subeventcondition> <connector> <eventconditionlist1>
| etype ':' <strvalue> with <eventconditionlist1>
    <connector> <subeventcondition>

```



```

<eventconditionlist1> := '(' <eventconditionlist1> ')'
| not '(' <eventconditionlist1> ')'
| <eventconditionlist1> and <eventconditionlist1>
| <eventconditionlist1> or <eventconditionlist1>
| <eventcondition1>

<eventcondition1> := location ':' <strvalue> // event location condition
| time ':' <generalvalue> // event time condition
| <objectid> ':' role '=' <strvalue> // role condition
| <objectcondition> // object condition
| <subeventcondition> // sub event condition
| <subeventcondition> <temporalconnector> <subeventcondition>

// General event condition specification without Subevents

<generaleventconditionW0sub> :=
    etype ':' <strvalue> [ with <eventconditionlist1> ]

<eventconditionlist1> := '(' <eventconditionlist1> ')'
| not '(' <eventconditionlist1> ')'
| <eventconditionlist1> and <eventconditionlist1>
| <eventconditionlist1> or <eventconditionlist1>
| <eventcondition1>

<eventcondition1> := location ':' <strvalue> // event location condition
| time ':' <generalvalue> // event time condition
| <objectid> ':' role '=' <strvalue> // role condition
| <objectcondition> // object condition

// Subevent condition specifications

<subeventcondition> := etype ':' <strvalue> [ with <playerlist> ]

<playerlist> := [ <playerlist> , ] <objectid>

```

```

// Object condition specifications

<objectconditions>      :=  odata '(' <objectconditionlist> ')'

<objectconditionlist> := '(' <objectconditionlist> ')'
                        | not '(' <objectconditionlist> ')'
                        | <objectconditionlist> and <objectconditionlist>
                        | <objectconditionlist> or <objectconditionlist>
                        | <objectcondition>

<objectcondition> := <objectid> '(' <attriblist> ')'

<attriblist> := <attriblist> ',' <attriblist>
              | <attributevalue> ':' <generalvalue>

// Utilities

<vid> := [1-9][0-9]*

<connector> := and | or

<temporalconnector> :=
    overlap | before | during | starts | finishes | meets
    | ioverlap | ibefore | iduring | istarts | ifinishes | imeets

<objectid> := <variable> | <atom>

<variable> := [A-Z][A-Za-z0-9]*

<atom> := [a-z][A-Za-z0-9]*

<attributevalue> := <strvalue>

<generalvale> := [0-9a-zA-Z]+

```

`<strvalue> := [a-zA-Z]+`

`<intvalue> := [0-9]+`

Appendix B

Java Class Definitions

Video

Video class holds metadata about video and a list of events and a list of objects of interest. Video class has all the information about video.

```
public class Video implements VideoComponent
{
    private String videoname;
    private double videolength;
    private int productionyear;
    private String producer;
    private String director;
    private String videotype;
    private String audience;
    private String subject;

    private TypedVector events;
    private TypedVector objects;

    private String videourl;
}
```

Event

```
public class Event implements VideoComponent
{
    private int eventid;

    private String activity;
    private EventData eventdata;
    private double begintime;
    private double endtime;
    private TypedVector subevents;
    private TypedVector objects;
}
```

EventData

```
public class EventData {
    private String time;
    private String location;
}
```

Event class holds data specific to events. It has a list of subevents and a list of objects appearing in that event. EventData class is the metadata part of events.

SubEvent

Subevent class holds data about each subevent. It has a list of objects appearing in that subevent.

```
public class SubEvent implements VideoComponent
{
    private int subeventid;
    private int eventid;    //which event does this subevent belong

    private String subactivity;
    private double begintime;
    private double endtime;
    private TypedVector objects;
}
```

VideoObject

VideoObject class holds the attributes and values for the attributes of the objects.

```
public class VideoObject implements VideoComponent
{
    private String name;
    private Vector attributes;
    private Vector values;
}
```

EventObject

EventObject class stores the roles of VideoObject in that event.

```
public class EventObject extends VideoObject implements VideoComponent
{
    private Vector roles;
    private int parentid;
}
```

SubEventObject

```
public class SubEventObject extends EventObject
{
    private int grandparentid; //event to which this object belongs
}
```

Appendix C

Database Table Specifications

TAUDIENCE:

```
audienceid    = integer
audiencename   = string
```

This table is used to store audience names like, teenager, children, adult, everyone, etc.

TVIDEOTYPE:

```
videotypeid    = integer
videotypename   = string
```

This table is used to store video type names, like adventure, horror, science-fiction, romance, etc.

TACTIVITY:

```
activityid     = integer
activityname    = string
```

This table is used to store activity names, like party, wedding, murder, war, etc.

TACTIVITYROLE:

```
roleid      = integer
activityid  = integer
rolename    = string
```

This table is used to store rolenames for each activity. For example, for the murder activity the role names are murderer and victim; and for the party activity the role names are host and guest.

TSUBACTIVITY:

```
subactivityid  = integer
subactivityname = string
```

This table is used to store actions such as talking, eating, dancing, fighting, etc.

TATTRIBUTE:

```
attributeid  = integer
name         = string
```

This table is used to store attribute names for video objects like realname, sex, color, speed, etc.

TVIDEO:

videoid	= integer
name	= string
length	= double
pyear	= integer
producer	= string
director	= string
videotype	= integer
audience	= integer
subject	= string
videourl	= string

This table stores bibliographic information about video. `videotype` and `audience` values are references to TVIDEOTYPE and TAUDIENCE tables.

TOBJECT:

objectid	= integer
videoid	= integer
name	= string

This table stores the object names for each video. `videoid` field is a reference to TVIDEO table.

TOBJECTATTRIBUTE:

objectid	= integer
attributeid	= integer
avalue	= string

This table stores the attribute values for each attribute defined for each object. `objectid` and `attributeid` are references to TOBJECT and TATTRIBUTE tables.

TEVENT:

```
eventid      = integer
videoid      = integer
activity     = integer
begintime    = double
endtime      = double
```

This table stores data about events, like activity type of event and start and end times of event. `videoid` field is a reference to `TVIDEO` and `activity` field is a reference to `TACTIVITY` table.

TEVENTDATA:

```
eventid      = integer
location     = string
timeofevent  = string
```

This table stores metadata about events. The data stored in this table can be extended. For the time being, location and time of event is stored. `eventid` field is a reference to `TEVENT` table.

TEVENTOBJECT:

```
eventid      = integer
objectid     = integer
```

This table stores the objects that appear in each event. `eventid` field is a reference to `TEVENT` table and `objectid` field is a reference to `TOBJECT` table.

TPLAYER:

```
eventid      = integer
objectid     = integer
roleid       = integer
```

This table stores the objects that appear in events plus their roles in the events. Objects can have many roles in an event. `eventid`, `objectid` and `roleid` fields are references to `TEVENT`, `TOBJECT`, and `TACTIVITYROLE` tables, respectively.

TSUBEVENT:

```
Subeventid   = integer
Eventid      = integer
Videoid      = integer
Subactivity   = integer
Begintime    = double
Endtime      = double
```

This table stores data about subevents. Subactivity, begin and end times are stored. `eventid` and `videoid` are references to `TEVENT` and `TVIDEO` tables.

TSUBPLAYER:

```
subplayerid  = integer
subeventid   = integer
```

This table stores the objects that appear in subevents. `subeventid` field is a reference to `TSUBEVENT` table and `subplayerid` field is a reference to `TSUBPLAYER` table.