DATA MODELING AND QUERYING FOR VIDEO DATABASES

A DISSERTATION 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
DOCTOR OF PHILOSOPHY

By
Mehmet Emin Dönderler
July, 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 dissertation for the degree of doctor of philosophy.

Assoc. Prof. Dr. Özgür Ulusoy (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 dissertation for the degree of doctor of philosophy.

Asst. 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 dissertation for the degree of doctor of philosophy.

Asst. Prof. Dr. Attila Gürsoy

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

Asst. Prof. Dr. Uğur Doğrusöz

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

Prof. Dr. Adnan Yazıcı

Approved for the Institute of Engineering and Science:

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

ABSTRACT

DATA MODELING AND QUERYING FOR VIDEO DATABASES

Mehmet Emin Dönderler
Ph.D. in Computer Engineering
Supervisors: Assoc. Prof. Dr. Özgür Ulusoy and
Asst. Prof. Dr. Uğur Güdükbay
July, 2002

With the advances in information technology, the amount of multimedia data captured, produced and stored is increasing rapidly. As a consequence, multimedia content is widely used for many applications in today's world, and hence, a need for organizing this data and accessing it from repositories with vast amount of information has been a driving stimulus both commercially and academically. In compliance with this inevitable trend, first image and especially later video database management systems have attracted a great deal of attention since traditional database systems are not suitable to be used for multimedia data.

In this thesis, a novel architecture for a video database system is proposed. The architecture is original in that it provides full support for spatio-temporal queries that contain any combination of spatial, temporal, object-appearance, external-predicate, trajectory-projection and similarity-based object-trajectory conditions by a rule-based system built on a knowledge-base, while utilizing an object-relational database to respond to semantic (keyword, event/activity and category-based) and low-level (color, shape and texture) video queries. Research results obtained from this thesis work have been realized by a prototype video database management system, which we call *BilVideo*. Its tools, *Fact-Extractor* and *Video-Annotator*, its Web-based visual query interface and its SQL-like textual query language are presented. Moreover, the query processor of *BilVideo* and our spatio-temporal query processing strategy are also discussed.

Keywords: video databases, multimedia databases, information systems, video data modeling, content-based retrieval, spatio-temporal relations, spatio-temporal query processing, video query languages.

ÖZET

VİDEO VERİ TABANLARI İÇİN VERİ MODELLEME VE SORGULAMA

Mehmet Emin Dönderler
Bilgisayar Mühendisliği, Doktora
Tez Yöneticileri: Doç. Dr. Özgür Ulusoy ve
Yard. Doç. Dr. Uğur Güdükbay
Temmuz, 2002

Bilgi teknolojisindeki gelişmeler ile, elde edilen, üretilen ve saklanan mültimedya veri miktarı hızlı bir şekilde artmakta ve bu veriler günümüzde birçok uygulamada kullanılmaktadır. Bu nedenle, bu verilerin düzenlenmesi ve bu verilere büyük miktarlarda bilgi bulunduran saklama alanlarından erişim gereksinimi, hem ticari hem de akademik olarak, bir tetikleyici etken oluşturmuştur. Kaçınılmaz olan bu eğilime bağlı olarak, ilk olarak resim ve özellikle daha sonra da video veri tabanı yönetim sistemleri, geleneksel veri tabanı sistemlerinin mültimedya için uygun olmaması nedeniyle, büyük bir ilgi çekmiştir.

Bu tezde, yeni bir video veri tabanı sistem mimarisi önerilmektedir. Bu mimarinin özelliği, yerleşimsel, zamansal, nesne görünüm, harici önerme, hareket izdüşüm ve benzerlik tabanlı nesne hareket koşullarının herhangi bir kombinasyonunu içeren yerleşim-zamansal sorgulara bir bilgi tabanı üzerine kurulu kural tabanlı bir sistem ile, anlamsal (anahtar kelime, olay/aktivite ve kategori tabanlı) ve alt seviyedeki (renk, şekil ve desen) video sorgularına da nesneye yönelik ve ilişkisel bir veri tabanı kullanılarak tam bir desteğin sağlanmasıdır. Bu tez kapsamında elde edilen araştırma sonuçları, *BilVideo* olarak isimlendirdiğimiz bir video veri tabanı yönetim sistemi prototipinin gerçekleştirilmesinde kullanılmıştır. *BilVideo* sisteminin parçaları olan *Gerçek Çıkartıcı*, *Video Anlamsal İlişkilendirici*, Web tabanlı görsel sorgu arayüzü ve SQL benzeri metne dayalı sorgu dili de tanıtılmaktadır. Ayrıca, *BilVideo* sisteminin sorgu işlemcisi ve yerleşim-zamansal sorgu işleme yöntemimiz de tartışılmaktadır.

Anahtar sözcükler: video veri tabanları, mültimedya veri tabanları, bilgi sistemleri, video veri modelleme, içerik-tabanlı veri alma, yerleşim-zamansal ilişkiler, yerleşim-zamansal sorgu işleme, video sorgu dilleri.

Acknowledgement

I would like to express my sincere gratitude to my supervisors Assoc. Prof. Dr. Özgür Ulusoy and Asst. Prof. Dr. Uğur Güdükbay for their instructive comments, suggestions, support and encouragement during this thesis work.

I am also very much thankful to Prof. Dr. Mehmet B. Baray for showing a keen interest in finding me a place to stay on campus during the last two years of my study, which accelerated the pace of my research considerably.

Finally, I am grateful to Asst. Prof. Dr. Attila Gürsoy, Asst. Prof. Dr. Uğur Doğrusöz and Prof. Dr. Adnan Yazıcı for reading and reviewing this thesis.

To My Family,

Contents

List of Figures

List of Tables

Chapter 1

Introduction

There is an increasing demand toward multimedia technology in recent years with the rapid growth in the amount of multimedia data available in digital format, much of which can be accessed through the Internet. As a consequence of this inevitable trend, first image and later video database management systems have attracted a great deal of attention both commercially and academically because traditional database systems are not suitable to be used for multimedia data. The following are two possible approaches in developing a multimedia system [39]:

a) metadata along with its associated multimedia data may be stored in a single database system, or

b) multimedia data is stored in a separate file system whereas the corresponding metadata is stored in a database system.

The first approach implies that databases should be redesigned to handle multimedia data together with conventional data. Since the user of the system may not need a full-fledged multimedia system and some modifications to existing databases are required, the first approach is not considered in practice. The second approach allows users to base their multimedia systems on their existing database systems with an additional multimedia storage server where the actual multimedia data is stored. Users only need to integrate their existing database

systems with the multimedia storage system, and even though this approach may complicate the implementation of some of the database functionalities such as data consistency, it is preferred over the first approach.

Major challenges in designing a multimedia system are [54]:

a) the storage and retrieval requirements of multimedia data,

b) finding an expressible and extensible data model with a rich set of modeling constructs, and

c) user interface design, query language and processing.

In this thesis, *BilVideo*, a Web-based prototype Video Database Management System (VDBMS), is introduced [7, 8]. The architecture of *BilVideo* is original in that it provides full support for spatio-temporal queries that contain any combination of spatial, temporal, object-appearance, external-predicate, trajectory-projection and similarity-based object-trajectory conditions by a rule-based system built on a knowledge-base, while utilizing an object-relational database to respond to semantic (keyword, event/activity and category-based) and low-level (color, shape and texture) video queries. The knowledge-base of *BilVideo* contains a fact-base and a comprehensive set of rules implemented in Prolog. The rules in the knowledge-base significantly reduce the number of facts that need to be stored for spatio-temporal querying of video data; our storage space savings was about 40% for some real video data we experimented on. Moreover, the system's response time for different types of spatio-temporal queries posed on the same data was at interactive rates [10]. Query processor interacts with both of the knowledge-base and object-relational database to respond to user queries that contain a combination of spatio-temporal, semantic and low-level video queries. Intermediate query results returned from these two system components are integrated seamlessly by the query processor and final results are sent to Web clients. *BilVideo* has a simple, yet very powerful SQL-like textual query language for spatio-temporal queries on video data [9]. For novice users, there is also a visual query language [6]. Both languages are currently being extended to

support semantic and low-level video queries. Contributions made by this thesis work can shortly be stated as follows:

Rule-based approach *BilVideo* uses a *rule-based* approach for modeling and querying spatio-temporal relations. Spatio-temporal relations are represented as Prolog facts partially stored in the knowledge-base and those relations that are not stored explicitly can be derived by our inference engine, Prolog, using the rules in the knowledge-base. *BilVideo* has a comprehensive set of rules, which reduces the storage space needed for spatio-temporal relations considerably as proven by our performance tests conducted using both synthetic and real video data.

Spatio-temporal video segmentation: A novel approach is proposed for the segmentation of video clips based on the spatial relationships between salient objects in video data. Video clips are segmented into shots whenever the current set of relations between salient objects changes, thereby helping us to determine parts of videos where the spatial relationships do not change at all.

Directional relations To determine which directional relation holds between two objects, center points of the objects' Minimum Bounding Rectangles (MBRs) are used. Thus, directional relations may also be defined for overlapping objects provided that the center points of their MBRs are different, as opposed to other works that are based on Allen's temporal interval algebra [2, 28, 46, 47].

Third-Dimension (3D) Relations Some additional relations were also defined on the third-dimension (z-axis of the three dimensional space) and rules were implemented for them. 3D relations defined in the system are *infrontof*, *behind*, *strictlyinfrontof*, *strictlybehind*, *touchfrombehind*, *touchedfrombehind* and *samelevel*.

Query types: *BilVideo* system architecture has been designed to support spatio-temporal (directional, topological, 3D-relation, external-predicate, object-appearance, trajectory-projection and similarity-based object-trajectory),

semantic (keyword, event/activity and category-based) and low-level (color, shape and texture) video queries in an integrated manner.

Query language: An SQL-like textual query language, based on our data model, is proposed for spatio-temporal querying of video data. This language is very easy to use even by novice users, who are a bit familiar with SQL. In fact, it is relatively easier to use compared with other proposed query languages for video databases, such as *CVQL*, *MOQL* and *VideoSQL* [21, 29, 38].

Retrieval Granularity: Users may wish to see only the parts of a video, where the conditions given in a query are satisfied, rather than the scenes that contain these segments. To the best of our knowledge, all the systems proposed in the literature associate video features with scenes that are defined to be the smallest logical units of video clips. Nevertheless, our spatio-temporal data model supports a finer granularity for query processing that is independent of semantic segmentation of videos (events/activities): it allows users to retrieve any segment of a video clip, in addition to semantic video units, as a result of a query. Thereby, *BilVideo* query language can return precise answers for spatio-temporal queries in terms of frame intervals.

Predicate-like conditions: Users specify the conditions in *where* clause of the *BilVideo* query language as is the same in SQL. However, spatial and external-predicate conditions are specified as Prolog-type predicates, which makes it much easier to shape complex query conditions, especially when combined with temporal operators, such as *before*, *during*, etc. Intermediate result sets computed for each subquery contain a list of interval sequences and/or a list of variable-value sequences. Output of all interval operators is of the same type, as well. Hence, temporal operators may follow one another in *where* clause, and the output of a temporal operator may become an input argument of the next one. This feature of the language results in a more intuitive, easy-to-write and easy-to-understand query declaration. It also provides more flexibility for users in forming complex spatio-temporal queries.

Aggregate Functions: *BilVideo* query language provides three aggregate functions, *average*, *sum* and *count*, which may be very attractive for some applications to collect statistical data on spatio-temporal events.

Application Independency: *BilVideo* is application-independent, and thus, it can be used for any application that requires spatio-temporal, semantic and low-level query processing capabilities on video data.

Extensibility: *BilVideo* can easily be tailored for specific requirements of any application through the definition of *external* predicates. *BilVideo* query language has a condition type *external* defined for application-dependent predicates. This condition type is generic, and hence, a user query may contain any application-dependent predicate in *where* clause of the language having a name different from a predefined predicate and language construct, and with at least one argument that might be either a variable or a constant (atom). Such predicates are processed just like spatial predicates as part of the Prolog subqueries. If an external predicate is to be used to query video data, facts and/or rules related to the predicate should be added to the knowledge-base priori, which is the only requirement posed.

1.1 Organization of the Thesis

The rest of this thesis is organized as follows:

Chapter 2 gives a review of the research in the literature that is related to our work.

Chapter 3 explains the overall architecture of *BilVideo* and gives some example spatio-temporal queries based on an imaginary soccer game fragment through which our rule-based approach is demonstrated.

Chapter 4 presents the tools developed for *BilVideo*, namely *Fact-Extractor* and *Video-Annotator*. The Fact-Extractor tool was developed to populate the knowledge-base of the system with facts for spatio-temporal querying of video data. The tool also extracts color and shape histograms of objects

and stores them in the feature database for low-level video queries. The Video-Annotator tool is used to annotate video clips for semantic content and to populate the system's feature database.

Chapter 5 presents the Web-based visual query interface of *BilVideo*.

Chapter 6 presents the system's SQL-like textual query language for spatio-temporal querying of video data.

Chapter 7 provides a discussion on the query processor of *BilVideo*.

Chapter 8 elaborates on our spatio-temporal query processing strategy.

Chapter 9 provides the results of our performance tests for spatio-temporal queries regarding the efficiency of the proposed system in terms of space and time criteria, and its scalability with respect to the number of salient objects per frame and the total number of frames in video.

Chapter 10 makes a discussion on the system's flexibility to support a broad range of applications and gives an example application of *BilVideo*, news archives search system, with some spatio-temporal queries.

Chapter 11 states the conclusions and future work.

Appendix A gives a list of our inference rules.

Appendix B presents the grammar of *BilVideo* query language.

Appendix C provides some of our spatio-temporal query processing functions in the form of simplified pseudo-codes.

Chapter 2

Related Work

There are numerous Content-Based Retrieval (CBR) systems, both commercial and academic, developed in recent years. However, most of these systems support only image retrieval. In this chapter, we restrict our discussion to the research in the literature mostly related to video modeling, indexing and querying. A comprehensive review on the CBR systems in general can be found in [52, 55].

One point worth noting at the outset is that *BilVideo*, to the best of our knowledge, is unique in its support for retrieving any segment of a video clip, where the given query conditions are satisfied, regardless of how video data is semantically partitioned. None of the systems discussed in this chapter can return a subinterval of a scene as part of a query result because video features are associated with scenes defined to be the smallest semantic units of video data. In our approach, object trajectories, object-appearance relations and spatio-temporal relations between video objects are represented as Prolog facts in a knowledge-base and they are not explicitly related to semantic units of videos. Thus, *BilVideo* can return precise answers for spatio-temporal queries in terms of frame intervals. Moreover, our assessment for the directional relations between two video objects is also novel in that overlapping objects may have directional relations defined for them provided that the center points of their MBRs are different. It is because Allen's temporal interval algebra, [2], is not used as a basis for the directional relation definition in our approach [10]: in order to determine which directional

relation holds between two objects, center points of the objects' MBRs are used. Furthermore, *BilVideo* query language provides three aggregate functions, *average*, *sum* and *count*, which may be very attractive for such applications as sports statistical analysis systems to collect statistical data on spatio-temporal events.

2.1 Spatio-Temporal Video Modeling

As mentioned in [48], there is a very limited number of proposals in the literature that take into account both spatial and temporal properties of video salient objects in an integrated manner. Some of the proposed index structures are *MR-tree*s and *RT-tree*s [53], *3D R-tree*s [49] and *HR-tree*s [37]. These structures are some adaptations of the well-known R-tree family. There are also quadtree-based indexing structures, such as *Overlapping Linear Quadtree*s [50], proposed for spatio-temporal indexing.

*3D R-tree*s consider time as an extra dimension to the original two-dimensional space. Thus, objects represented by two-dimensional MBRs are now captured by three-dimensional Minimum Bounding Boxes (MBBs). However, if this approach were to be used for moving objects, a lot of empty space would be introduced within objects' MBBs since the movement of an object is captured by using only one MBB. Thus, it is not a proper representation mechanism for video data, where objects frequently change their positions in time.

*RT-tree*s are proposed to solve this dead space problem by incorporating the time information by means of time intervals inside the R-tree structure. Nevertheless, whenever an object changes its position, a new entry with temporal information must be inserted to the structure. This causes the generation of many entries that makes the *RT-tree* grow considerably. Furthermore, time information stored with nodes plays a complementary role and *RT-tree*s are not able to answer temporal queries such as "*find all objects that exist in the database within a given interval*".

*MR-tree*s and *HR-tree*s use the concept of overlapping B-trees [32]. They have separate index structures for each time point where a change occurs in an object position within the video data. It is space-efficient if the number of objects changing their locations is low because index structures may have some common paths for those objects that have not moved. Nonetheless, if the number of moving objects is large, they become inefficient. Detailed discussion of all these index structures can be found in [48].

All these approaches incorporate the MBR representation of spatial information within index structures. Thus, to answer spatio-temporal queries, spatial relations should be computed and checked for query satisfaction, which is a costly operation when performed during query processing. Our rule-based approach to model spatio-temporal relations in video data eliminates the need for the computation of relations at the time of query processing, thereby cutting down the query response time considerably. In our approach, a keyframe represents some consecutive frames in a video with no change in the set of spatial relations between video objects in the frames. Computed spatial relations for each keyframe are stored to model and query video data for spatio-temporal relations.

Li et al. describe an effort somewhat similar to our approach, where some spatial relations are computed by associated methods of objects while others may be derived using a set of inference rules [28]. Nonetheless, the system introduced in [24, 25, 28] does not explicitly store a set of spatio-temporal relations from which a complete set of relations between all pairs of objects can be derived by rules, and consequently, the relations which cannot be derived by rules are computed during query processing. Our approach of pre-computing and storing a set of relations that cannot be derived by the set of inference rules a priori to querying reduces the computational cost of queries considerably since there is no need at all to compute any spatio-temporal relation using any coordinate information at the time of query processing. All the relations that are not stored explicitly in the fact-base can be easily derived by the inference rules.

A video model, called Common Video Object Tree Model (CVOT), is described in [24]. In this model, there is no restriction on how videos are segmented. After the segmentation, shots are grouped in a hierarchy on the basis of the common video objects they contain, developing an index structure, called CVOT. However, employed as a common practice by all the systems proposed in the literature to the best of our knowledge, video features are associated with scenes that are defined to be the smallest logical units of videos. In our approach, spatio-temporal relations between video objects, object-appearance relations and object-trajectories are represented as facts in a knowledge-base and they are not explicitly related to semantic units of videos. It is because users may also wish to see only the parts of a video, where the conditions given in a query are satisfied, rather than the scenes that contain these segments. Thus, *BilVideo* returns precise answers for spatio-temporal queries in terms of frame intervals whereas this functionality is not implemented in CVOT.

Sistla et al. propose a graph and automata based approach to find the minimal set of spatial relations between objects in a picture given a set of relations that is a superset of the minimal set [46, 47]. They provide algorithms to find the minimal set from a superset as well as to deduce all the relations possible from the minimal set itself for a picture. However, the authors restrict the directional relations to be defined only for disjoint objects as opposed to our approach, where overlapping objects may also have directional relations. Moreover, the set of inference rules considered in their implementation is rather small compared to ours. They do not incorporate any 3D relation, either. Furthermore, our fact-extraction algorithm is simpler and it extracts spatio-temporal, appearance and trajectory properties of objects from a video even though we do not claim that it produces the minimal set of spatial relations in a video frame as they do for a picture.

2.2 Semantic Video Modeling

A video database system design for automatic semantic extraction, semantic-based video annotation and retrieval with textual tags is proposed in [31]. Low-level image features, such as color, shape, texture and motion, and object extraction/recognition techniques are used in extracting some semantic content from video clips. To reveal the temporal information, the authors use temporal diagrams for videos and scenes in videos. Components of a temporal diagram constructed for a video are the temporal diagrams for scenes, and the arcs between two such components (scenes) present the relationships between the scenes in one cluster. A temporal diagram created for a scene contains the shots in the scene, and the components in the diagram represent the objects in the shots. Video semantic content is automatically extracted using low-level image features (color, shape, texture and motion) and the temporal diagrams constructed for videos and scenes. As a result of this process, shots/scenes are added some textual descriptions (tags), which are used for semantic queries. However, automatic extraction of semantic content and tagging shots/scenes with some textual descriptions with respect to the extracted information are limited to simple events/activities.

Hacid et al. propose a video data model that is based on logical video segment layering, video annotations and associations between them [35]. The model supports user queries and retrieval of the video data based on its semantic content. The authors also give a rule-based constraint query language for querying both semantic and video image features, such as color, shape and texture. Color, shape and texture query conditions are sent to IBM's QBIC system whereas semantic video query conditions are processed by FLORID, a deductive object-oriented database management system. A database in their model can essentially be thought of as a graph and a query in their query language can be viewed as specifying constrained paths in the graph. *BilVideo* does not use a rule-based approach for semantic queries on video data. In this regard, our semantic video model diverts from the one proposed by Hacid et al.

There is also some research in the literature that takes into account audio and closed caption text stored together with video data for extracting semantic

content from videos and indexing video clips based on this extracted semantic information. In [4], a method of event-based video indexing by means of inter-model collaboration, a strategy of collaborative processing considering the semantic dependency between synchronized multimodal information streams, such as auditory and textual streams, is proposed. The proposed method aims to detect interesting events automatically from broadcasted sports videos and to give textual indexes correlating the events to shots. In [16], a digital video library prototype, called VISION, is presented. VISION is being developed at the Information and Telecommunication Technologies Laboratory of the University of Kansas. In VISION, videos are automatically partitioned into short scenes using audio and closed caption information. The resulting scenes are indexed based on their captions and stored in a multimedia system. Informedia's news-on-demand system described in [17] also uses the same information (audio and closed caption) for automatic segmentation and indexing to provide efficient access to news videos. Satoh et al. propose a method of face detection and indexing by analyzing closed caption and visual streams [43]. However, all these systems and others that take into account audio and closed caption information stored with videos for automatic segmentation and indexing are application-dependent whilst *BilVideo* is not.

2.3 Systems and Languages

2.3.1 QBIC

QBIC is a system primarily designed to query large online image databases [14]. In addition to text-based searches, QBIC also allows users to pose queries using sketches, layout or structural descriptions, color, shape, texture, sample images (Query by Example) and other iconic and graphical information. As a basis for content-based search, it supports color, shape, texture and layout. For example, it is possible to give a query such as "*Return the images that have blue at the top and red at the bottom*", which is a color-based search with layout specification.

QBIC provides some support for video data, as well [15]; however, this support is limited to the features used for image queries: video is represented as an ordered set of representative frames (still images) and the content-based query operators used for images are applicable to video data through representative frames. Consequently, spatio-temporal relations between salient objects and semantic content of video data are not taken into account for video querying.

2.3.2 OVID and VideoSQL

A paper by Oomoto and Tanaka [38] describes the design and implementation of a prototype video object database system, named OVID. Main components of the OVID system are VideoChart, VideoSQL and Video Object Definition Tool. Each video object consists of a unique identifier, a pair of starting and ending video frame numbers for the object, annotations associated with the object as a set of attribute/value pairs and some methods such as *play*, *inspect*, *disaggregate*, *merge* and *overlap*. Users may define different video objects for the same frame sequences and each video object is represented as a bar chart on the OVID user interface VideoChart. VideoChart is a visual interface to browse the video database and manipulate/inspect the video objects within the database. The query language of the system, VideoSQL, is an SQL-like query language used for retrieving video objects. The result of a VideoSQL query is a set of video objects, which satisfy given conditions. Before examining the conditions of a query for each video object, target video objects are evaluated according to the interval inclusion inheritance mechanism. A VideoSQL query consists of the basic *select*, *from* and *where* clauses. However, the *select* clause in VideoSQL is considerably different from the ordinary SQL *select* clause in that it only specifies the category of the resultant video objects with `Continuous`, `Incontinuous` and `anyObject`. `Continuous` retrieves video objects with a single continuous video frame sequence while `Incontinuous` retrieves those objects with more than one single continuous video frame sequence. `anyObject` is used to retrieve all types of video objects regardless of whether they are contiguous or not. The *from* clause is used to specify the name of the object database, and *where* clause is used to state the

conditions for a query. Conditions may contain attribute/value pairs and comparison operators. Video numbers may also be used in specifying conditions. In addition, VideoSQL has a facility to merge the video objects retrieved by multiple queries. Nevertheless, the language does not contain any expression to specify spatial and temporal conditions on video objects. Hence, VideoSQL does not support spatio-temporal queries, which is a major weakness of the language.

2.3.3 MOQL and MTQL

In [30], multimedia extensions to the Object Query Language (OQL) and TIGUKAT Query Language (TQL) are proposed. The extended languages are called Multimedia Object Query Language (MOQL) and Multimedia TIGUKAT Query Language (MTQL), respectively. The extensions made are spatial, temporal and presentation features for multimedia data. MOQL has been used both in the Spatial Attributes Retrieval System for Images and Videos (STARS) [27] and an object-oriented SGML/HyTime compliant multimedia database system [40] developed at the University of Alberta.

Most of the extensions that are introduced with MOQL are in *where* clause in the form of three new predicate expressions: `spatial-expression`, `temporal-expression` and `contains-predicate`. A `spatial-expression` may include spatial objects (points, lines, circles, etc.), spatial functions (length, area, intersection, union, etc.) and spatial predicates (cover, disjoint, left, right, etc.). A `temporal-expression` may contain temporal objects, temporal functions (union, intersection, difference, etc.) and temporal predicates (equal, before, meet, etc.). Moreover, `contains-predicate` is used to determine if a particular media object contains a given salient object. A media object may be either an image object or a video object. Besides, a new clause `present` is introduced to deal with multimedia presentation. With this clause, the layout of the presentation is specified.

MTQL has the same extensions as those made for MOQL, namely spatial,

temporal and presentation properties. Hence, both languages support content-based spatial and temporal queries as well as query presentation. MOQL and MTQL include support for 3D-relation queries, as we call them, by *front*, *back* and their combinations with other directional relations, such as *front_left*, *front_right*, etc. *BilVideo* query language has a different set of third-dimension (3D) relations, though. 3D relations supported by *BilVideo* query language are *infrontof*, *behind*, *strictlyinfrontof*, *strictlybehind*, *touchfrombehind*, *touchedfrombehind* and *samelevel*. The moving object model integrated in MOQL and MTQL, [26], is also different from our model. *BilVideo* query language does not support similarity-based retrieval on spatial conditions as MOQL and MTQL do. Nonetheless, it does allow users to specify separate weights for the directional and displacement components of trajectory conditions in queries, which both languages lack.

Nabil et al. propose a symbolic formalism for modeling and retrieving video data by means of moving objects in video frames [36]. A *scene* is represented as a connected digraph whose nodes are the objects of interest in the scene while edges are labeled by a sequence of spatio-temporal relations between two objects corresponding to the nodes. Trajectories are also associated with object nodes in the scene graph. A graph is precomputed for each scene in video data and stored before query processing. For each user query, a query scene graph is constructed to match the query with the stored scene graphs. However, 3D relations are not addressed in [36]. The concepts used in the model are similar to those adopted in [26]; therefore, the same arguments we made for MOQL and MTQL also hold for the model proposed in [36].

2.3.4 AVIS

In [34], a unified framework for characterizing multimedia information systems, which is built on top of the implementations of individual media, is proposed. Some of user queries may not be answered efficiently using these data structures; therefore, for each media-instance, some feature constraints are stored as a logic program. Nonetheless, temporal aspects and relations are not taken into account in the model. Moreover, complex queries involving aggregate operations as well

as uncertainty in queries require further work to be done. In addition, although the framework incorporates some feature constraints as facts to extend its query range, it does not provide a complete deductive system as we do.

The authors extend their work defining feature-subfeature relationships in [33]. When a query cannot be answered, it is relaxed by substituting a subfeature for a feature. This relaxation technique provides some support for reasoning with uncertainty. In [1], a special kind of segment tree called *frame segment tree* and a set of arrays to represent objects, events, activities and their associations are introduced. The proposed model is based on the generic multimedia model described in [34]. Additional concepts introduced in the model are activities, events and their associations with objects, thereby relating them to frame sequences. The proposed data model and algorithms for handling different types of semantic queries were implemented within a prototype, called Advanced Video Information System (AVIS). However, objects have no attributes other than the roles defined for the events. In [19], an SQL-like video query language, based on the data model developed by Adalı et al. [1], is proposed. Nevertheless, the proposed query language does not provide any support for temporal queries on events. Nor does it have any language construct for spatio-temporal querying of video clips since it was designed for semantic queries on video data. In our query model, temporal operators, such as *before*, *during*, etc., may also be used to specify order in time between events just as they are used for spatio-temporal queries.

2.3.5 VideoQ

An object-oriented content-based video search engine, called VideoQ, is presented in [5]. VideoQ provides two methods for users to search for video clips. The first one is to use *keywords* since each video shot is annotated. Moreover, video clips are also catalogued into a subject taxonomy and users may navigate through the catalogue easily. The other method is a visual one, which extends the capabilities of the textual search. A video object is a collection of regions that are grouped together under some criteria across several frames. A region is defined as a

set of pixels in a frame, which are homogeneous in the features of interest to the user. For each region, VideoQ automatically extracts the low-level features, *color*, *shape*, *texture* and *motion*. These regions are further grouped into higher semantic classes known as video objects. The regions of a video object may exhibit consistency in some of the features, but not all. For example, an object representing a person walking may have several regions, which show consistency only on the motion attribute of the video object, but not the others. Motion is the key attribute in VideoQ and the motion trajectory interface allows users to specify a motion trajectory for an object of interest. Users may also specify the duration of the object motion in an absolute (in seconds) or intuitive (long, medium and short) way. Video queries are formulated by animated sketches. That is, users draw objects with a particular shape, paint color, add texture and specify motion to pose a query. Objects in the sketch are then matched against those in the database and a ranked list of video shots complying with the requirements is returned. The total similarity measure is the weighted sum of the normalized distances and these weights can be specified by users while drawing the sketches of various features. When a query involves multiple video objects, the results of each individual video object query are merged. The final query result is simply the logical intersection of all individual video object query results. However, when a multiple object query is submitted, VideoQ does not use the video objects' relative ordering in space and in time. Therefore, VideoQ does not support spatio-temporal queries on video data.

2.3.6 VideoSTAR

VideoSTAR proposes a generic data model that makes it possible sharing and reusing of video data [18]. Thematic indexes and structural components might implicitly be related to one another since frame sequences may overlap and may be reused. Therefore, considerable processing is needed to explicitly determine the relations, making the system complex. Moreover, the model does not support spatio-temporal relations between video objects.

2.3.7 CVQL

A content-based logic video query language, CVQL, is proposed in [22]. Users retrieve video data specifying some spatial and temporal relationships for salient objects. An elimination-based preprocessing for filtering unqualified videos and a behavior-based approach for video function evaluation are also introduced. For video evaluation, an index structure, called *M-index*, is proposed. Using this index structure, frame sequences satisfying a query predicate can be efficiently retrieved. Nonetheless, topological relations between salient objects are not supported since an object is represented by a point in two-dimensional (2D) space. Consequently, the language does not allow users to specify topological queries. Nor does it support similarity-based object-trajectory queries. *BilVideo* query language provides full support for spatio-temporal querying of video data.

Chapter 3

BilVideo VDBMS

3.1 BilVideo System Architecture

BilVideo is built over a client-server architecture as illustrated in Figure 3.1. The system is accessed on the Internet through its visual query interface developed as a Java Applet. Users may query the system with sketches and a visual query is formed by a collection of objects with some conditions, such as object trajectories with similarity measures, spatio-temporal orderings of objects, annotations and events. Object motion is specified as an arbitrary trajectory for each salient object of interest and annotations may be used for keyword-based video search. 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 experienced users.

Web clients communicate user queries to the query processor. If queries are specified visually, they are first transformed into SQL-like textual query language expressions before being sent to the query server. The query processor is responsible for retrieving and responding to user queries. It first separates the semantic and low-level (color, shape and texture) query conditions in a query from those that could be answered by the knowledge-base. The former type of conditions is organized and sent as regular SQL queries to an object-relational database

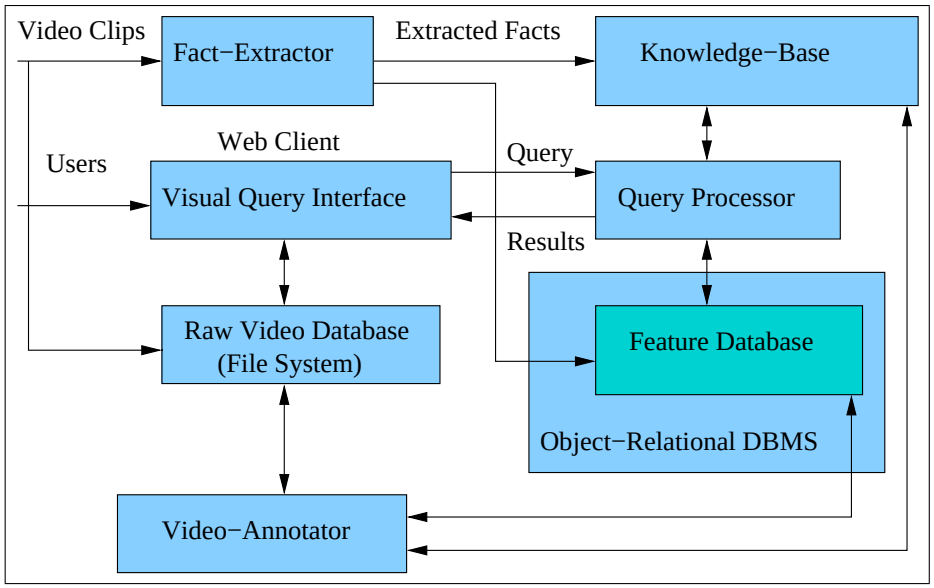


Figure 3.1: BilVideo System Architecture

whereas the latter part is reconstructed as Prolog queries. Intermediate results returned by these two system components are integrated by the query processor and the final results are sent to Web clients.

Raw video data and video data features are stored separately. The feature database contains semantic and low-level properties for videos. Video semantic features are generated and maintained by the Video-Annotator tool developed as a Java application. The knowledge-base is used to respond to spatio-temporal queries on video data and the facts-base is populated by the Fact-Extractor tool, which is a Java application as well. The Fact-Extractor tool also extracts color and shape histograms of objects of interest in video keyframes to be stored in the feature database [45].

3.2 Knowledge-Base Structure

Rules have been extensively used in knowledge representation and reasoning. The reason of why we employed a rule-based approach to model and query spatio-temporal relations between salient objects is that it is very space efficient: only does a relatively small number of facts need to be stored in the knowledge-base and the rest can be derived by the inference rules, which yields a substantial improvement in storage space. Besides, our rule-based approach provides an easy-to-process and easy-to-understand structure for a video database system.

In the knowledge-base, each fact[1] has a single frame number, which is of a keyframe. This representation scheme allows Prolog, our inference engine, to process spatio-temporal queries faster and easier than it would with frame intervals attached to the facts because the frame interval processing to form the final query results can be carried out efficiently by some optimized code, written in C++, outside the Prolog environment. Therefore, the rules used for querying video data, which we call *query rules*, have frame-number variables as a component. A second set of rules that we call *extraction rules* was also created to work with frame intervals in order to extract spatio-temporal relations from video clips. Extracted spatio-temporal relations are converted to be stored as facts with frame numbers of the keyframes attached in the knowledge-base and these facts are used by the query rules for query processing in the system. In short, spatio-temporal relations in video clips are stored as Prolog facts in the knowledge-base in a keyframe basis and the extraction rules are only used to extract the spatio-temporal relations from video data.

The reason of using a second set of rules with frame intervals to extract spatio-temporal relations is that it is much easier and more convenient to create the facts-base by first populating an initial facts-base with frame intervals and then converting this facts-base to the one with frame numbers of the keyframes

[1]Except for *appear* and *object-trajectory* facts, which have frame intervals as a component instead of frame numbers because of storage space, ease of processing and processing cost considerations.

in comparison to directly creating the final facts-base in the process of fact-extraction. The main difficulty, if a second set of rules with frame intervals had not been used while extracting spatio-temporal relations, would be detecting the keyframes of a video clip when processing it frame by frame at the same time. It is not a problem so far as the coding is concerned, but since the program creating the facts-base would perform this keyframe detection operation for each frame, it would take whole a lot of time to process a video clip compared to our method.

In the knowledge-base, only are the basic facts stored, but not those that can be derived by rules according to our fact-extraction algorithm. Nonetheless, using a frame number instead of a frame interval introduces some space overhead because the number of facts increases due to the repetitions of some relations for each keyframe over a frame interval. Nevertheless, it also greatly reduces the complexity of the rules and improves the overall query response time.

The algorithm developed for converting an initial facts-base of a video clip to the one incorporated into the knowledge-base is very simple. It makes use of a keyframe vector, also stored as a fact in the facts-base, which stores frame numbers of the keyframes of a video clip in ascending order. Using this vector, each fact with a frame interval is converted into a group of facts with frame numbers of the keyframes. For example, if *west*(A, B, [1, 100]) is a fact in the initial facts-base and 1, 10 and 50 are the keyframes that fall into the frame interval range of [1, 100], then, this fact is converted to the following facts in the knowledge-base: *west*(A, B, 1), *west*(A, B, 10) and *west*(A, B, 50). Keyframe detection and fact-base conversion are automatically performed by the Fact-Extractor tool for each video clip processed.

In the system, facts are stored in terms of four directional relations, *west*, *south*, *south-west* and *north-west*, six topological relations, *cover*, *equal*, *inside*, *disjoint*, *touch* and *overlap*, and four 3D relations defined on z-axis of the three dimensional space, *infrontof*, *strictlyinfrontof*, *touchfrombehind* and *samelevel*, because the query rules are designed to work on these types of explicitly stored facts. However, there are also rules for *east*, *north*, *north-east*, *south-east*, *right*, *left*, *below*, *above*, *behind*, *strictlybehind*, *touchedfrombehind*, *contains* and *covered-by*.

These rules do not work directly with the stored facts, but rather they are used to invoke related rules. For example, let's suppose that there is a relation stored as a fact for the pair of objects σ(A, B), such as *west*(A, B, 1), where A and B are object identifiers and 1 is the frame number of the relation. When a query "*east*(B, A, F)" is posed to the system, the rule *east* is used to call the rule *west* with the order of objects switched. That is, it is checked to see if *west*(A, B, F) can be satisfied. Since there is a fact *west*(A, B, 1) stored in the facts-base, the system returns 1 for F as the result of the query. This argument also holds for the extraction rules only this time for extracting relations from a video clip rather than working on stored facts. Therefore, the organization of the extraction rules is the same as that of the query rules.

Four types of inference rules, *strict directional*, *strict topological*, *heterogeneous directional and topological*, and *3D rules*, were defined with respect to the types of the relations in the rule body. For example, *directional rules* have only directional relations in their body whilst *heterogeneous rules* incorporate both directional and topological components. The complete listing of our inference rules is given in Appendix A.

In addition, some other facts, such as *object-trajectory* and *appear* facts, are also stored in the knowledge-base. These facts have frame intervals rather than frame numbers attached as a component. *Appear* facts are used to derive some trivial facts, *equal*(A, A), *overlap*(A, A) and *samelevel*(A, A), as well as to answer object-appearance queries in video clips by rules. *Object-trajectory* facts are used for processing trajectory-projection and similarity-based object-trajectory query conditions.

Table 3.1 presents semantic meanings of our 3D relations based on Allen's temporal interval algebra. The relations *behind*, *strictlybehind* and *touchedfrombehind* are inverses of *infrontof*, *strictlyinfrontof* and *touchfrombehind*, respectively. Moreover, the relation *strictlyinfrontof* is transitive whilst *samelevel* is reflexive and symmetric. While the relations *strictlyinfrontof* and *strictlybehind* impose that objects be disjoint on z-axis of the three dimensional space, *infrontof* and

Relation	Inverse	Meaning
A infrontof B	B behind A	AAA BBB or AAABBB or AAA BBB
A strictlyinfrontof B	B strictlybehind A	AAA BBB or AAABBB
A samelevel B	B samelevel A	AAA BBBBBB or AAA BBBBBB or AAA BBBBBB or AAA BBB
A touchfrombehind B	B touchedfrombehind A	BBBAAA

Table 3.1: Definitions of 3D relations on z-axis of three-dimensional space

behind do not enforce this condition. Hence, if object o_1 *strictlyinfrontof* (*strictlybehind*) object o_2, then o_1 *infrontof* (*behind*) o_2. Object o_1 *touchfrombehind* object o_2 iff o_1 *strictlybehind* o_2 and o_1 touches o_2 on the z-axis. If object o_1 *samelevel* object o_2, then, $o_1(o_2)$ is inside, covered-by or equal to $o_2(o_1)$ on z-axis of the three dimensional space. Further information on directional and topological relations can be found in [13, 41].

3.3 Fact-Extraction Algorithm

The algorithm for deciding what relations to store as facts in the knowledge-base is illustrated as a pseudo-code in Figure 3.2. In this algorithm, objects at each frame, κ, are ordered with respect to the center-point x-axis values of objects' MBRs. Index values of the objects are used as object labels after this sorting process. Then, objects are paired with respect to their labels starting with the object whose label is 0. The directional and topological relations are computed for each possible object pair whose first object's label is smaller than that of the

second object and whose label distance is one. The *label distance* of an object pair is defined as the absolute numerical difference between the object labels. After exhausting all the pairs with the label distance one, the same operation is carried out for the pairs of objects whose label distance is two. This process is continued in the same manner and terminated when the distance reaches the number of objects in the frame.

Initially, the set of relations, η, is empty. All directional and topological relations are computed for each object pair as described above for the current frame being processed and the computed relations are put in the array λ in order. Then, for each relation in λ, starting with the first one indexed as 0, it is checked to see if it is possible to derive the computed relation from the relations in η by the extraction rules. For example, for the first frame, if a relation cannot be derived from η using the rules, this relation is added to η with the frame interval [0, 0]. Otherwise, it is ignored since it can be derived. For the consecutive frames, if a computed relation cannot be derived, an additional check is made to see whether there is such a relation in η that holds for a frame interval up to the current frame processed. If so, the frame interval of that relation is extended with the current frame by increasing the last component of its interval by one. Otherwise, the computed relation is added to η with the frame interval [*current frame*, *current frame*]. The set of relations obtained at the end contains the relations that must be stored as facts in the knowledge-base after conversion. The rest of the relations may be derived from these facts by rules.

For 3D relations, computation cannot be done automatically since 3D coordinates of the objects are unknown and cannot be extracted from video frames. Hence, these relations are entered manually for each object-pair of interest and those that can be derived by rules are eliminated automatically by the Fact-Extraction tool. The tool can perform an interactive conflict check for 3D relations and has some facilities to keep the existing set of 3D relations intact for the consecutive frames as well as to edit this set with error-and-conflict check on the current set for the purpose of easy generation of 3D relations. Generation of 3D relations is carried out for each frame of a video clip at the same time while the rest of the spatio-temporal relations is extracted. These 3D relations are then

```
1.  Start with an empty set of facts, η.
2.  Set m to the number of frames in video
3.  For (currentFrame = 0; currentFrame < m; currentFrame++)
4.  Begin
5.      Set κ to be the object array of the current frame
6.      Sort κ in ascending order on x-axis coordinates of object MBR center points
        (* Use object index values in the sorted array as object labels *)
7.      Set n = |κ| (Number of objects in the frame)
8.      For (i = 0; i < n; i++)
9.      Begin
10.         If (there exist an object-appearance fact for κ[i] in η)
11.             Update this fact accordingly for currentFrame
12.         Else
13.             Put an object-appearance fact for κ[i] in η
14.         If (κ[i] has changed its position in [currentFrame - 1, currentFrame])
15.             If (there exist an object-trajectory fact for κ[i] in η)
16.                 Update this fact accordingly for currentFrame
17.             Else
18.                 Put an object-trajectory fact for κ[i] in η
19.     EndFor
20.     Set λ to be an empty array
21.     Set index to 0
22.     For (labelDistance = 1; labelDistance < n; labelDistance++)
23.     Begin
24.         For (index1 = 1; index1 < n − labelDistance; index1++)
25.         Begin
26.             index2 = index1 + labelDistance
27.             Find dirRelation(κ[index1], κ[index2]) and put it in λ[index]
28.             Increment index by 1
29.             Find topRelation(κ[index1], κ[index2]) and put it in λ[index]
30.             Increment index by 1
31.         EndFor
32.     EndFor
33.     Put 3D relations in λ incrementing index by 1 at each step
34.     Reorder λ with respect to the dependency criteria among relations as follows:
            * A relation with a smaller index value is placed before a relation of the same
              type with a bigger index value
            * The order of placement is (a), (b), (c), (d), (e), (f), (g) and (h)
            a) {equal}, b) directional relations, c) {cover, touch}
            d) {inside}, e) {overlap, disjoint}, f) {samelevel, touchfrombehind}
            g) {strictlyinfrontof}, h) {infrontof}
35.     Update η as follows: (* Facts-base population *)
36.         For (i = 0; i < index; i++)
37.         Begin
38.             If (λ[i] can be derived by extraction rules using the relations in η)
39.                 Skip and ignore the relation
40.             Else
41.                 If (∃β ∈ η such that β is the same as λ[i] except for its frame
                        interval whose ending frame is currentFrame − 1)
42.                     Extend the frame interval of β by 1 to currentFrame
43.                 Else
44.                     Put λ[i] in η with the frame interval [currentFrame, currentFrame]
45.         EndFor
46. EndFor
```

Figure 3.2: Fact-Extraction Algorithm

put in λ and they, along with the rest of the relations, are also used for keyframe detection.

The initial fact-base, η, is also populated with the *appear* and *object-trajectory* facts. For each object, an *appear* fact is kept where it appears in video represented with a list of frame intervals. Furthermore, for each object, an *object-trajectory* fact is added for the entire video. These facts are copied to the final facts-base without any conversion. *Appear* facts are also used to detect keyframes if an object appears when there is no object in the previous frame or if an object disappears while it is the only object in the previous frame.

Our approach greatly reduces the number of relations to be stored as facts in the knowledge-base, which also depends on some other factors as well, such as the number of salient objects, the frequency of change in spatial relations, and the relative spatial locations of the objects with respect to each other. Nevertheless, it is not claimed that the set of relations stored in the knowledge-base is the minimal set of facts that must be stored because the number of facts to be stored depends on the labeling order of objects in our method and we use the x-axis ordering to reduce this number. Our heuristic in this approach is that if it is started with the pairs of objects whose label distance is smaller, most of the relations may not need to be stored as facts for the pairs of objects with a bigger label distance. The reason is that these relations might be derived from those already considered to be stored in the knowledge-base. In addition, since the spatial relations are ordered according to the dependency criteria given in Table 3.2 before deciding which relations to store in the facts-base, no dependent relation is stored just because a relation of different type it depends on has not been processed yet, except for the relations *strictlyinfrontof* and *infrontof*. The relations *strictlyinfrontof* and *infrontof* depend on each other; however, the precedence is given to *strictlyinfrontof* since it implies *infrontof*.

The fact-extraction process is semi-automatic: objects' MBRs are specified manually and 3D relations are entered by the user through graphical components. Users do not have to draw each MBR for consecutive frames because

Order	Relation	Dependencies
1	equal	-
2	Directional Relations	equal
3	cover, touch	equal
4	inside	equal, cover
5	overlap, disjoint	equal, cover, touch, inside
6	samelevel, touchfrombehind	-
7	strictlyinfrontof	touchfrombehind, infrontof
8	infrontof	touchfrombehind, strictlyinfrontof

Table 3.2: Dependencies Among Rules

MBR resizing, moving and deletion facilities are provided for convenience. Moreover, the tool performs 3D-relation conflict check and eliminates the derivable 3D relations from the set as they are entered by the user. The set for 3D relations is also kept intact for subsequent frames so that the user can update it without having to reenter any relation that already exists in the set. Nevertheless, with this user intervention involved, it is not possible to make a complete complexity analysis of the algorithm. During our experience with the tool, it has been observed that the time to populate a facts-base for a given video is dominated by the time spent interacting with the tool. However, since the fact extraction process is carried out offline, it does not have any influence on the system's performance. When the user intervention part is ignored, the complexity of our algorithm can be roughly stated as $O(mn^2)$, where m is the number of frames processed and n is the average number of objects per frame. It is a rough estimation because the facts-base is populated as frames are processed and it is not possible to guess the size of the facts-base or the number of relations put in the set by type at any time during the fact-extraction process.

3.4 Directional Relation Computation

According to our definition, overlapping objects can also have directional relations associated with them except for the pairs of objects whose MBRs' center points are the same, as opposed to the case where Allen's temporal interval algebra is

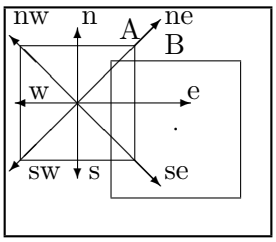
used to define the directional relations.

In order to determine which directional relation holds between two objects, the center points of the objects' MBRs are used. Obviously, if the center points of the objects' MBRs are the same, then there is no directional relation between the two objects. Otherwise, the most intuitive directional relation is chosen with respect to the closeness of the line segment between the center points of the objects' MBRs to the eight directional line segments. For that, the origin of the directional system is placed at the center of the MBR of the object for which the relation is defined. In the example given in Figure 3.3, object A is to the *west* of object B because the center of object B's MBR is closer to the directional line segment *east* than the one for *south-east*. Moreover, these two objects overlap with each other, but a directional relation can still be defined for them. As a special case, if the center points of objects' MBRs fall exactly onto the middle of two directional segments, which one to be considered is decided as follows: the absolute distance of the objects' MBRs is computed on x and y axes with respect to the farmost vertex coordinates on the region where the two directional line segments in question reside. If the distance in x-axis is greater, then the line segment that is closer to the x-axis is selected. Otherwise, the other one is chosen. Here, the objects' relative sizes and positions in 2D coordinate system implicitly play an important role in making the decision. Our approach to find the directional relations between two salient objects can be formally expressed as in Definitions 1 and 2.

Definition 1 *The directional relation $\beta(A,B)$ is defined to be in the opposite direction to the directional line segment that originates from the center of object A's MBR and is the closest to the center of object B's MBR.*

Definition 2 *The inverse of a directional relation $\beta(A,B)$, $\beta^{-1}(B,A)$, is the directional relation defined in the opposite direction.*

According to Definition 1, given two objects A and B, if the center of object B's MBR is closer to the directional line segment *east* in comparison to the others



west(A,B), east(B,A)

Figure 3.3: Directional Relation Computation

when the directional system's origin is at the center of object A's MBR, then the directional relation between objects A and B is *west*(A, B), where object A is the one for which the relation is defined. Thus, object A is to the *west* of object B. Using Description 2, it can also be concluded that object B is to the *east* of object A. The rest of the directional relations can be determined in the same way.

3.5 Query Examples

This section provides some spatio-temporal query examples based on an imaginary soccer game fragment between England's two teams *Arsenal* and *Liverpool*. These queries do not have any 3D-relation condition. Nor do they contain any temporal, trajectory-projection or similarity-based object-trajectory conditions because algorithms to process such conditions were still under development at the time of testing the system. In the examples, the word "player(s)" is used for the member(s) of a soccer team except for the goalkeeper. Prolog query predicates and query results are only provided for the first example.

Example 1 "Give the number of passes for each player of *Arsenal*".

Query: `pass X Y arsenal`, where X and Y are variables that stand for the players of *Arsenal* who give and take the passes, respectively.

Query Predicates:

```
pass(X, Y, T) :- fmember(X, T), fmember(Y, T), X \= Y,
       p_touch(X, ball, F1), p_inside(ball, field, F1),
```

```
        noother(X, ball, F1), p_touch(Y, ball, F2), F2 > F1,
        p_inside(ball, field, F2), noother(Y, ball, F2),
        fkframe(L, F1, F2), checklist(p_inside(ball, field), L),
        checklist(notouch(ball), L).

fmember(X, T) :- (getmembers(L, T), member(X, L),
        not(goalkeeper(X, T))).

noother(X, Y, F) :- findall(Z, p_touch(Z, Y, F), L),
        forall(member(Z, L), Z = X).

fkframe(L, F1, F2) :- keyframes(K),
        findall(X, kframes(X, K, F1, F2), L).

keyframes([1, 10, 21, 25, 31, 35, 55, 61, 80, 91, 95, 101, 105,
        111, 115, 121, 125, 131, 135, 141, 150, 161, 165, 171,
        172, 175, 181]).

kframes(X, L, F1, F2) :- member(X, L), X > F1, X < F2.

notouch(X, F) :- not(p_touch(Z, X, F)).

goalkeeper(X, T) :- getmembers(Y, T), last(X, Y).

getmembers(X, T) :- (T = arsenal, X = [dixon, keown, adams,
        winterburn, ljunberg, petit, vieira, overmars, kanu,
        bergkamp, seaman]); (T = liverpool, X = [staunton,
        henchoz, hyypia, heggem, carragher, redknapp, hamann,
        smicer, owen, camara, westerveld]).
```

It is assumed that if a player touches the ball alone, it is in his control. Consequently, if a player of *Arsenal* touches the ball for some time and then transfers the control of it to another player of his team, this event is considered as a pass from this player to another one in his team. Moreover, the ball should not be played (touched) by anyone else and it should also stay inside the field during this event.

The result of this query is:

```
Player:keown Passes(given):1
Player:adams Passes(given):2
Player:kanu Passes(given):1
Player:bergkamp Passes(given):1

Team:arsenal Total Passes:5
```

Example 2 "Give the number of shots to the goalkeeper of the opponent team for each player of *Arsenal*".

Query: `shoot X arsenal`, where X is a variable that stands for the players of *Arsenal* who shoot.

In this query, we are interested in finding the number of shots to the goalkeeper of *Liverpool* by each player of *Arsenal*. In order to answer this query, the facts of *touch* to the ball are found for each player of *Arsenal*. For each fact found, it is also checked if there is a fact of *touch* to the ball for the opponent team's goalkeeper, whose frame number is bigger. Then, a check is made to see if there is no other *touch* to the ball between these two events and also if the ball is inside the field during the entire period. If all above conditions are satisfied, this is considered a shot to the goalkeeper. Then, all such occasions are counted to find the number of shots to the goalkeeper by each player of *Arsenal*.

Example 3 "Give the average ball control (play) time in frames for each player of *Arsenal*".

Query: `hold X arsenal`, where X is a variable that stands for the players of *Arsenal* who play with the ball.

As it is assumed that when a player touches the ball alone, it is in his control, the ball control time for a player is computed with respect to the frame intervals during which he is in touch with the ball. Therefore, the following operation is performed for each player of *Arsenal* so as to answer this query: frame intervals

during which a player touches the ball are found and the number of frames in the intervals are summed up. Divided by the number of frame intervals found, this gives for the player the average ball control time in terms of the number of frames. Since in a soccer game, a player may touch the ball outside the field as well, only are the frame intervals when the ball is inside the field considered. It is also possible to give the time information in seconds provided that the frame rate of the video is known.

Example 4 "Give the number of ball losses to the opponent team's players for *Adams* of *Arsenal*".

Query: `loss adams arsenal`.

If *Adams* of *Arsenal* touches the ball for some time and then the control of the ball goes to a player of the opponent team, this event is considered as a ball loss from *Adams* to an opponent player. Furthermore, the ball should not be played (touched) by anyone else and it should stay inside the field during this event.

Example 5 "Give the number of kicks to outside field for *Adams* of *Arsenal*".

Query: `outside adams arsenal`.

First, the keyframes when *Adams* of *Arsenal* is in touch with the ball while the ball is inside the field are found. Then, for each keyframe found, a fact with a bigger frame number, representing the ball being outside the field, is searched. If there is no *touch* to the ball between these two events, then this is a kick outside the field. All such occasions are counted to find the number of kicks outside the field by *Adams*.

Chapter 4

Tools For BilVideo

4.1 Fact-Extractor Tool

Fact-Extractor is used to populate the facts-base of *BilVideo* and to extract color and shape histograms of salient objects in video keyframes. Spatio-temporal relations between salient objects, object-appearance relations and object trajectories are extracted semi-automatically. This information is stored in the facts-base as a set of facts representing the relations and trajectories, and it is used to query video data for spatio-temporal query conditions. Sets of facts are kept in separate facts-files for each video clip processed, along with some other video specific data, such as video length, video rate, keyframes list, etc., extracted automatically by the tool. Extracted color and shape histograms of salient objects are stored in the feature database to be used for color and shape video queries.

The fact-extraction process is semi-automatic: objects are manually specified in video frames by MBRs. Using the object MBRs, a set of spatio-temporal relations (directional and topological) is automatically computed. The rules in the knowledge-base are used to eliminate redundant relations; therefore, the set contains only the relations that cannot be derived by the rules. For 3D-relations, extraction cannot be done automatically because 3D-coordinates of the objects

cannot be obtained from video frames. Hence, these relations are entered manually for each object-pair of interest and the relations that can be derived by the rules are eliminated automatically. The tool performs an interactive conflict-check for 3D-relations and carries the set of 3D-relations of a frame to the next frame so that the user may apply any changes in 3D-relations by editing this set in the next frame. Object trajectories and object-appearance relations are also extracted automatically for each object once the objects are identified by their MBRs. Moreover, object MBRs need not be redrawn for each frame since MBR resizing, moving and deletion facilities are available. When exiting the tool after saving the facts, some configuration data is also stored in the knowledge-base if the video is not entirely processed yet so that the user may continue processing the same video clip later on from where it was left off. Since object MBRs are drawn manually by users, there is a space for erroneous MBR specification although in many cases small errors do not affect the set of relations computed. To automate this process, an *Object-Extractor* utility module has been developed [44]. We plan to embed this module into *Fact-Extractor* to help users specify object MBRs with a few mouse clicks on objects instead of drawing them manually.

Fact-Extractor populates the facts-base with facts that have a single frame number, which is of a keyframe, except for the object-appearance and object trajectory facts that have frame intervals rather than frame numbers because of storage space, ease of processing and processing cost considerations. Thus, the tool segments video clips into shots, each represented by a single keyframe, during the process of fact-extraction. This segmentation is based on spatial relationships between objects in video frames: video clips are segmented into shots whenever the current set of relations between objects changes and the video frames, where these changes occur, are chosen as keyframes. The relations stored in the facts-base are those that are present in such keyframes in a video clip because the set of relations in a frame does not change from frame to frame in the same shot. Hence, *BilVideo* can support much finer granularity for spatio-temporal query processing, which is independent of the semantic segmentation of video clips employed by all other video database systems in the literature to the best of our knowledge: it allows users to retrieve any part of a video clip, where the

Figure 4.1: Fact-Extractor Tool

relations do not change at all, in addition to semantic video units, as a result of a query.

Fact-Extractor uses a heuristic algorithm to decide which spatio-temporal relations to store as facts in the knowledge-base as explained in Chapter 3. Figure 4.1 gives a snapshot of the *Fact-Extractor* tool.

4.2 Video-Annotator Tool

Video-Annotator is a tool developed for annotating video clips for semantic content and populating the system's feature database with this data to be used for semantic video queries. The tool also provides facilities for viewing, updating and deleting semantic data that has already been obtained from video clips and stored in the feature database. A snapshot of the tool is given in Figure 4.2.

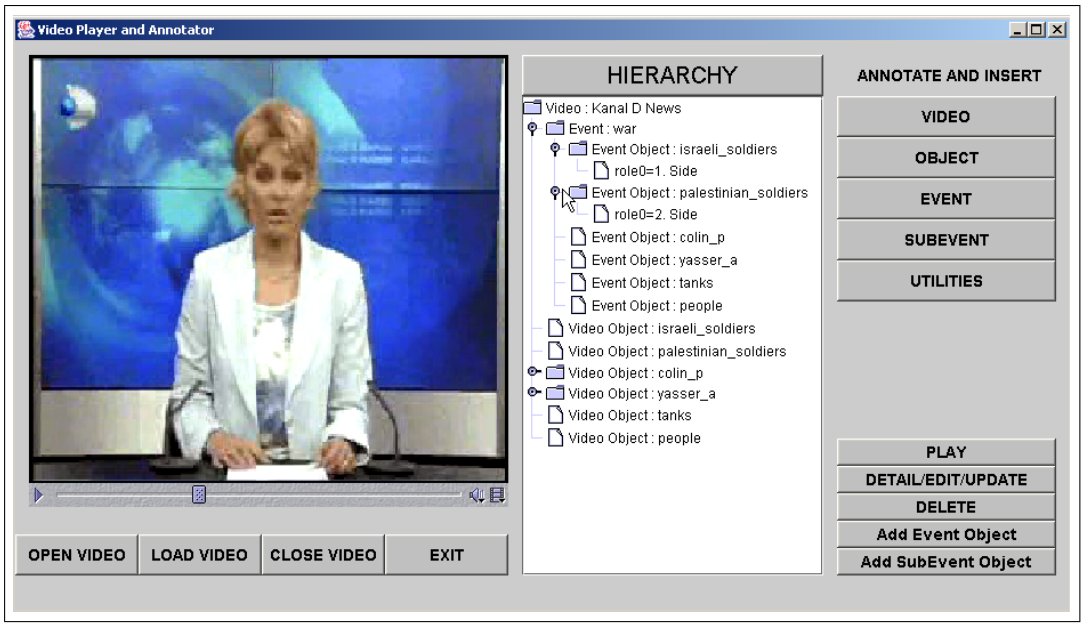


Figure 4.2: Video-Annotator Tool

Our semantic video hierarchy contains three levels: *video*, *sequence* and *scene*. *Videos* consist of *sequences* and *sequences* contain *scenes* that need not be consecutive in time. With this semantic data model, we plan to answer three types of queries: *video*, *event/activity* and *object*. *Video* queries can be used for retrieving videos based on descriptional data (annotations) of video clips. Conditions may include title, length, producer, production year, category and director information about a video clip. *Event/activity* queries are the most common queries among all and they can be used to retrieve videos by specifying events that occur at the *sequence* layer because events are associated with sequences. However, a particular scene or scenes of an event can also be returned as an answer to a semantic query when requested because events may have subevents associated with scenes. *Object* queries are used to retrieve videos by specifying semantic object features. As videos are annotated, video salient objects are also associated with some descriptional meta data. With respect to our semantic video model, a relational database schema, which consists of fifteen database tables, has been designed to store semantic contents of videos, such as bibliographic information about videos, utility data (audiences, video types, activity types, roles for activity

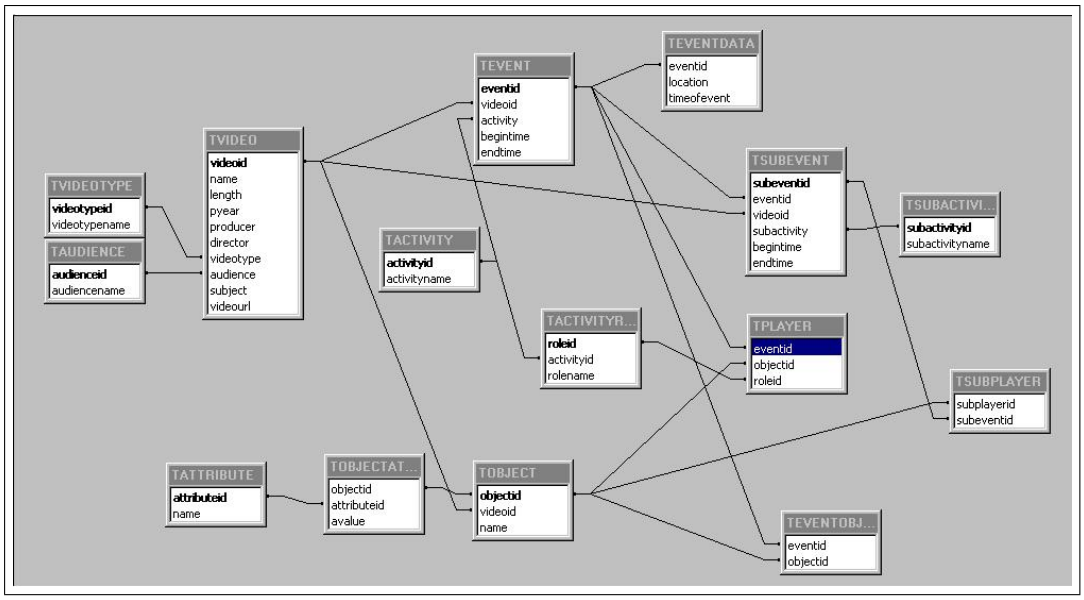


Figure 4.3: Database Schema for Our Video Semantic Model

types, subactivity types, object attributes, etc.) and data about objects of interests, events and subevents. The conceptual design of the database is presented in Figure 4.3.

Video consists of events, and activities are the abstractions of events. For example, wedding is an activity, but the wedding of Richard Gere and Julia Roberts in a movie is considered as an event, a specialization of activity wedding. Hence, activities can be thought of as classes while events constitute some instances (specializations) of these classes in videos. In our semantic model, a number of roles are defined for each activity. For example, activity murder is defined with two roles, *murderer* and *victim*. If the murder of Richard Gere by Julia Roberts is an event in a movie, then Richard Gere and Julia Roberts have the roles *victim* and *murderer*, respectively. Events may also have subevents defined for them, and these subevents are used to detail events and model the relationships between objects of interest. For example, a party event in a video may have a number of subevents, such as drinking, eating, dancing and talking, as some people participating in this event may be drinking, eating, dancing or talking. Moreover, the objects of interest in the party event may assume the roles *host*

and *guest*. Objects are defined and assigned roles for an event; however, they are also associated with subevents defined for an event because actions represented by subevents, such as dancing and talking in the example given, are performed by those objects. Furthermore, subevents may overlap in time as is the case for events. In our semantic video model, a video is segmented into sequences, which are in turn divided into scenes. This task is accomplished by specifying events and subevents because events and subevents are associated with sequences and scenes, respectively. The order of annotation follows our hierarchical semantic model from top to bottom. In other words, video is annotated first as a whole entity and the annotation of events with their corresponding subevents may be carried out afterwards. During this process, objects may be annotated whenever needed. Further information on the video annotation process, *Video-Annotator* and our relational database schema for storing semantic contents of videos can be found in [3].

Chapter 5

Web-based User Interface

BilVideo can handle multiple requests over the Internet through a graphical query interface developed as a Java Applet [6]. The interface is composed of query specification windows for different types of queries: *spatial* and *trajectory*. The specification and formation of these queries vary significantly, and hence, specific windows to handle them are created. Since video has a time dimension, these two types of primitive queries can be combined with temporal predicates (*before*, *during*, etc.) to query temporal contents of videos.

Since the relations that are stored in the knowledge-base (e.g. directional, topological, etc.) are computed according to the MBRs of the salient objects during database population, users draw rectangles for salient objects, which represent objects' MBRs, in query specification. Specification of queries by visual sketches is much easier for novice users and most of the relations are computed automatically based on these sketches.

5.1 Spatial Query Specification

Spatial content of a video keyframe is the relative positioning of its salient objects with respect to each other. This relative positioning consists of three separate sets of relations: *directional*, *topological* and *3D relations*. In order to query the

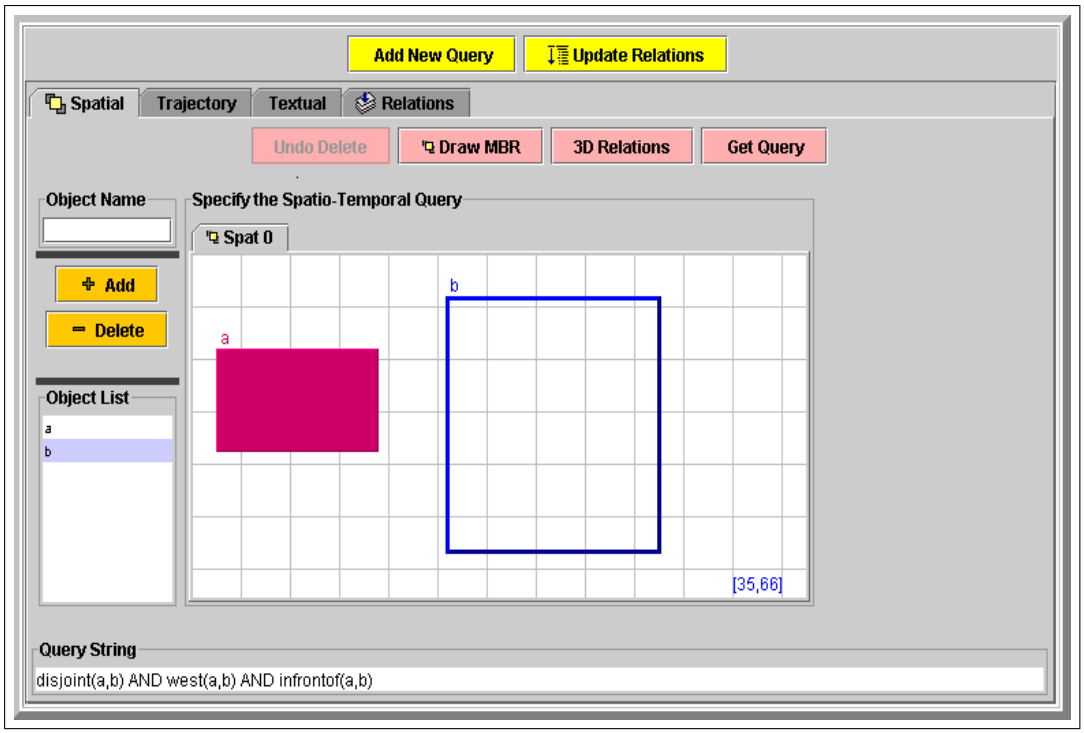


Figure 5.1: Spatial Query Specification Window

spatial content of a keyframe, these relations have to be specified in the query within a proper combination. This combination should be constructed with the logical connector *and*; thus, all the relations have to be present in the video frame(s) returned as a result.

In the spatial query specification window shown in Figure 5.1, salient objects are sketched by rectangles, which represent MBRs of the objects. Similar to the database population phase, the directional and topological relations between objects are extracted automatically from the MBRs of the salient objects in the query specification phase. Since it is impossible to extract 3D relations from 2D data, users are guided to select appropriate 3D relations for salient-object pairs. To provide flexibility, some facilities are also provided: users may change the locations, sizes and relative positions of the MBRs during the query specification. The spatial-relation extraction process takes place after the final configuration is formalized. Deleting or hiding a salient object modifies the relation set, and if this

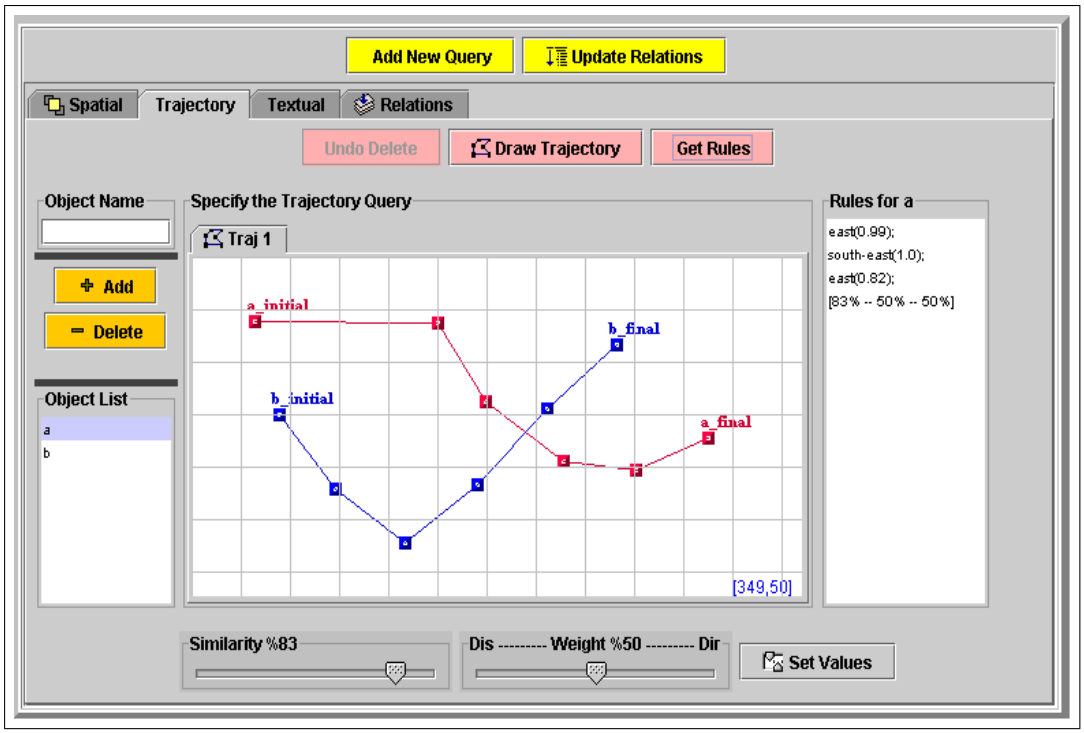


Figure 5.2: Trajectory Query Specification Window

modification occurs after the extraction process, relations relating to the deleted or hidden objects are removed accordingly from the set.

5.2 Trajectory Query Specification

Trajectory of a salient object is described as a path of vertices corresponding to the locations of the object in different video keyframes. Displacement values and directions between consecutive keyframes (vertices) are used in defining the trajectory fact of an object. In the trajectory query specification window shown in Figure 5.2, users can draw trajectories of salient objects as a sequence of vertices. The trajectories are dynamic in the sense that any vertex can be deleted from or a new vertex can be inserted to the trajectory of a salient object. Locations of the vertices can also be altered to obtain a desired trajectory.

Object-trajectory queries are similarity-based; therefore, users specify a similarity value, between 0 and 100, where the value 100 implies an exact match. Since an object trajectory contains lists of directions and displacements, weights can be assigned to each list. By default, both lists have equal weights (i.e., the weights are 0.5); however, users may modify these values that add up to 1. There are two sliders on the trajectory specification window (see lower part of Fig. 5.2): the first slider is for similarity value and the other slider is for assigning weights. If the head of the slider used for weight specification is closer to the left end, directions become more important than displacements, and vice versa.

5.3 Final Query Formulation

Spatial and trajectory queries are specified in separate windows through the user interface. Each of these specifications forms a subquery and these subqueries are combined in the final query formulation window as shown in Figure 5.3. This window contains all the specified subqueries as well as object-appearance relations for each object. Users can combine subqueries by logical operators (*and, or*) and temporal predicates (*before*, *during*, etc.). Except for the logical operator *not*, all temporal and logical operators are binary. If more than two subqueries are given as arguments to binary operators, the first two are processed first and the output is pipelined back to the operator to be processed with the next argument. After applying operators to subqueries, a new query is augmented to the list, and hierarchical combinations become possible. After the final query is formed, it can be sent to the query server. Furthermore, any subquery of the final query may also be sent to the query server at any time to obtain partial results if requested.

For an example visual query formulation, let us suppose that a and b are two salient objects and that their query trajectories are denoted by T_a and T_b, respectively. Let us also assume that S_1 denotes a spatial subquery on objects a and b, which is given as `west(`a`,`b`) AND disjoint(`a`,`b`) AND infrontof(`a`,`b`)`. The query `appear(`a`) AND appear(`b`) AND finishes(`S_1`, before(`T_a`,`T_b`))` may be formed visually as follows: Trajectory subqueries T_a and T_b are constructed

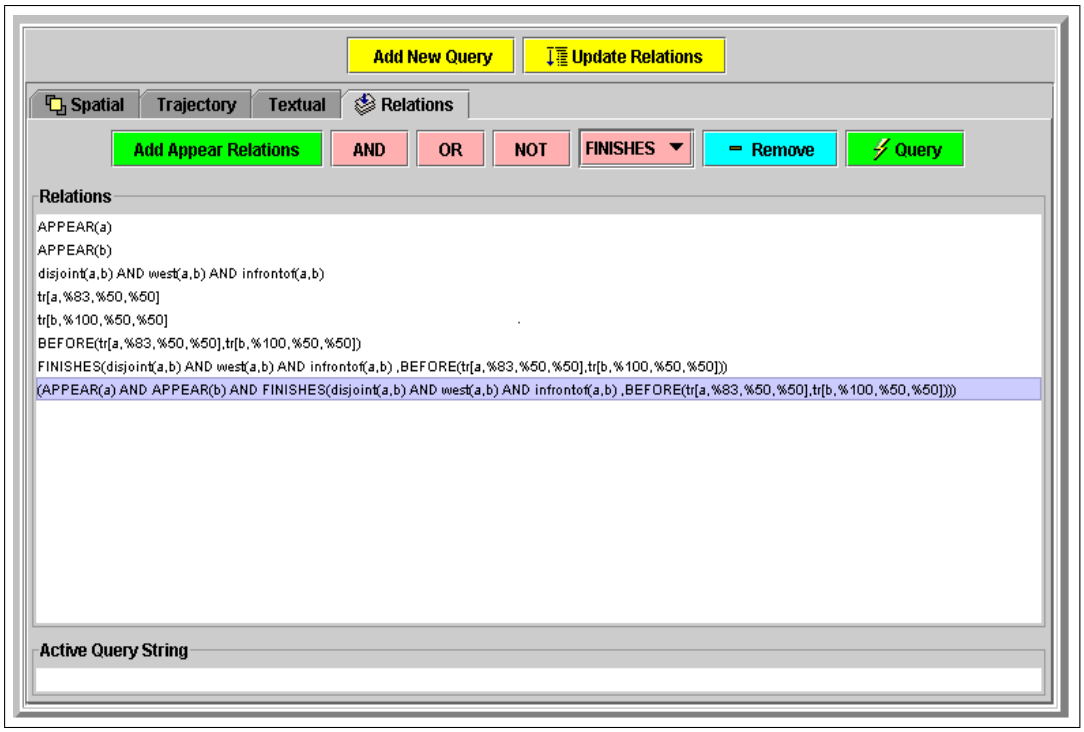


Figure 5.3: Final Query Formulation Window

in the trajectory query specification window while the spatial subquery S_1 is specified in the spatial query specification window. After the subqueries are formed, the final query can be defined visually in the final query formulation window. This window displays as a list all the subqueries constructed. The conditions `appear(`a`)` and `appear(`b`)` are added to the list in the window and are connected with the *and* operator. This new subquery is added to the list and it can be used to form new composite conditions. Then, the condition `before(`T_a`, `T_b`)` is constructed, which is also displayed in the list. After that, the condition `finishes(`S_1`, before(`T_a`, `T_b`))` can be formed with the conditions S_1 and `before(`T_a`, `T_b`)` in the list, and it is connected to the condition `appear(`a`) AND appear(`b`)` constructed before, using the operator *and*. This last composition gives the final query.

Chapter 6

BilVideo Query Language

Retrieval of video data by its content is a very important and challenging task. Users should be able to query a video database by spatio-temporal relations between video objects, object-appearance relations, object trajectories, low-level features (color, shape and texture), keywords (annotations) as well as some other semantic contents (events/activities). Query languages designed for relational, object and object-relational databases do not provide sufficient support for content-based video retrieval; either a new language that supports all these types of content-based queries on video data should be designed and implemented or an existing language should be extended with the required functionality.

In this chapter, we present a new video query language that is similar to SQL in structure. The language can currently be used for spatio-temporal queries that contain any combination of directional, topological, 3D-relation, object-appearance, external-predicate, trajectory-projection and similarity-based object-trajectory conditions. As a work in progress, the language is being extended so that it could support semantic (keyword, event/activity and category-based) and low-level (color, shape and texture) queries as well in a unified and integrated manner.

6.1 Features of the Language

BilVideo query language has four basic statements for retrieving information:

`select` *video* `from all` [`where` *condition*];

`select` *video* `from` *videolist* `where` *condition*;

`select` *segment* `from` *range* `where` *condition*;

`select` *variable* `from` *range* `where` *condition*;

Target of a query is specified in `select` clause. A query may return videos (*video*) or segments of videos (*segment*), or values of variables (*variable*) with/without segments of videos where the values are obtained. Regardless of the target type specified, video identifiers for videos in which the conditions are satisfied are always returned as part of the query answer. Aggregate functions, which operate on segments, may also be used in `select` clause. Variables might be used for object identifiers and trajectories. Moreover, if the target of a query is *video*, users may also specify the maximum number of videos to be returned as a result of a query. If the keyword `random` is used, video fact-files to process are selected randomly, thereby returning a random set of videos as a result. The range of a query is specified in `from` clause, which may be either the entire video collection or a list of specific videos. Query conditions are given in `where` clause. In *BilVideo* query language, *condition* is defined recursively, and consequently, it may contain any combination of spatio-temporal query conditions.

Supported Operators: The language supports a set of logical and temporal operators to be used in query conditions. Logical operators are *and*, *or* and *not* while temporal operators are *before*, *meets*, *overlaps*, *starts*, *during*, *finishes* and their inverse operators.

The language also has a trajectory-projection operator, *project*, which can be used to extract subtrajectories of video objects on a given spatial condition. The condition is local to *project* and it is optional. If it is not given, entire object trajectories rather than subtrajectories of objects are returned.

The language has two operators, "=" and "!=", to be used for assignment and comparison. The left argument of these operators should a variable whereas the right argument may be either a variable or a constant (atom). Operator "!=" is used for inequality comparison while operator "=" may take on different semantics depending on its arguments. If one of the arguments of operator "=" is an unbound variable, it is treated as the assignment operator. Otherwise, it is considered as the equality-comparison operator. These semantics were adopted from the Prolog language.

Operators that perform interval processing are called *interval operators*. Hence, all temporal operators are interval operators. Logical operators are also considered as interval operators if their arguments contain intervals.

In the language, the precedence values of logical, assignment and comparison operators follow their usual order. Logical operators assume the same precedence values defined for them when they are considered as interval operators, as well. Temporal operators are given a higher priority over logical operators when determining the arguments of operators and they are left associative as are logical operators.

The query language also provides a keyword, `repeat`, that can be used in conjunction with a temporal operator, such as *before*, *meets*, etc., or a trajectory condition. Video data may be queried by repetitive conditions in time using `repeat` with an optional repetition number given. If a repetition number is not given with `repeat`, then it is considered indefinite, thereby causing the processor to search for the largest intervals in a video, where the conditions given are satisfied at least once over time. The keyword `tgap` may be used for temporal operators and a trajectory condition. However, it has rather different semantics for each type. For temporal operators, `tgap` is only meaningful when `repeat` is used because it specifies the maximum time gap allowed between pairs of intervals to be processed for `repeat`. Therefore, the language requires that `tgap` be used along with `repeat` for temporal operators. For a trajectory condition, it may be used to specify the maximum time gap allowed for consecutive object movements as well as pairs of intervals to be processed for `repeat` if `repeat` is also given in

the condition. Hence, `tgap` may be used in a trajectory condition without any restriction.

Aggregate Functions: The query language has three aggregate functions, *average*, *sum* and *count*, which take a set of intervals (segments) as input. *Average* and *sum* return a time value in minutes whilst *count* returns an integer for each video clip satisfying given conditions. *Average* is used to compute the average of the time durations of all intervals found for a video clip whereas *sum* and *count* are used to calculate the total time duration for and the total number of all such intervals, respectively. These aggregate functions might be very useful to collect statistical data for some applications, such as sports event analysis systems, motion tracking systems, etc.

External Predicates: The proposed query language is generic and designed to be used for any application that requires spatio-temporal query processing capabilities. The language has a condition type *external* defined for application-dependent predicates, which we call *external predicates*. This condition type is generic; consequently, a query may contain any application-dependent predicate in `where` clause of the language with a name different from any predefined predicate and language construct, and with at least one argument that is either a variable or a constant (atom). External predicates are processed just like spatial predicates as part of Prolog subqueries. If an external predicate is to be used for querying video data, facts and/or rules related to the predicate should be added to the knowledge-base beforehand.

In our design, each video segment returned as an answer to a user query has an associated importance value ranging between 0 and 1, where 1 denotes an exact match. The results are ordered with respect to these importance values in descending order. Prolog subqueries return segments with importance value 1 because they are exact-match queries whereas the importance values for the segments returned by similarity-based object-trajectory queries are the similarity values computed. Interval operators *not* and *or* return the complements and

union of their input intervals, respectively. Interval operator *or* returns intervals without changing their importance values whilst the importance value for the intervals returned by *not* is 1. The rest of the interval operators takes the average of the importance values of their input interval pairs for the computed intervals. Users may also specify a time period in a query to view only parts of videos, either from the beginning or from the end of videos, returned as an answer. The grammar of the language is given in Appendix B.

6.2 Query Types

The architecture of *BilVideo* has been designed to support spatio-temporal, semantic and low-level (color, shape and texture) queries in an integrated manner.

6.2.1 Object Queries

This type of queries may be used to retrieve salient objects for each video queried that satisfies the conditions, along with segments if desired, where the objects appear. Some example queries of this type are given below:

Query 1: "Find all video segments from the database in which object o_1 appears."

```
select segment
from all
where appear(o1);
```

Query 2: "Find the objects that appear together with object o_1 in a given video clip and also return such segments." (Video identifier for the given video clip is assumed to be 1.)

```
select segment, X
from 1
where appear(o1, X) and X != o1;
```

6.2.2 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: directional relations that describe order in 2D space, topological relations that describe neighborhood and incidence in 2D space and 3D relations that describe object positions on the z-axis of three-dimensional space. There are eight distinct topological relations: *disjoint*, *touch*, *inside*, *contains*, *overlap*, *covers*, *covered-by* and *equal*. Fundamental directional relations are *north*, *south*, *east*, *west*, *north-east*, *north-west*, *south-east* and *south-west*, and 3D relations contain *infrontof*, *strictlyinfrontof*, *touchfrombehind*, *samelevel*, *behind*, *strictlybehind* and *touchedfrombehind*.

6.2.3 Similarity-Based Object-Trajectory Queries

In our data model, for each moving object in a video clip, a trajectory fact is stored in the facts-base. A trajectory fact is modelled as tr(ν, φ, ψ, κ), where

ν: object identifier,
φ (list of directions): [φ_1, φ_2, ..., φ_n] where $\varphi_i \in$ F[1] (1≤i≤n),
ψ (list of displacements): [ψ_1, ψ_2, ..., ψ_n] where $\psi_i \in Z^+$ (1≤i≤n), and
κ (list of intervals): [[s_1, e_1], ..., [s_n, e_n]] where s_i, $e_i \in$ N and $s_i \leq e_i$ (1≤i≤n).

A trajectory query is modeled as

`tr`(α, λ) [`sthreshold` σ [`dirweight` β | `dspweight` η]] [`tgap` γ]

[1] set of fundamental directional relations

or

`tr`(α, θ) [`sthreshold` σ] [`tgap` γ]

where

α: object identifier,
λ: trajectory list ([θ, χ]),
θ: list of directions,
χ: list of displacements,
`sthreshold` (similarity threshold): $0< \sigma <1$,
`dirweight` (directional weight): $0\leq \beta \leq 1$ and 1-$\beta = \eta$,
`dspweight` (displacement weight): $0\leq \eta \leq 1$ and 1-$\eta = \beta$, and
`tgap`: time threshold, $\gamma \in$ N, for the gap between consecutive object movements.

In a trajectory query, variables may be used for α and λ, and the number of directions is equal to the number of displacements in λ just like in trajectory facts because each element of a list is associated with an element of the other list that has the same index value. The list of directions specifies a path an object follows whilst the displacement list associates each direction in this path with a displacement value. However, it is optional to specify a displacement list in a query in which case no weights are used in matching trajectories. Such queries are useful when displacements are not important to the user.

In a trajectory query, similarity and time threshold values are also optional. If a similarity threshold is not given, the query is considered as an exact-match query. A query without a `tgap` value implies a continuous motion without any stop between consecutive object movements. The time threshold value specified in a query is considered in seconds. A trajectory query may have either a directional or a displacement weight supplied because the other is computed from the one given. Moreover, for a weight to be specified, a similarity threshold value must also be provided. If a similarity value is supplied and no weight is given, then the weights of the directional and displacement components are considered equal by default. The key idea in measuring the similarity between a pair of trajectories is to find the distance between the two and this is achieved by computing the

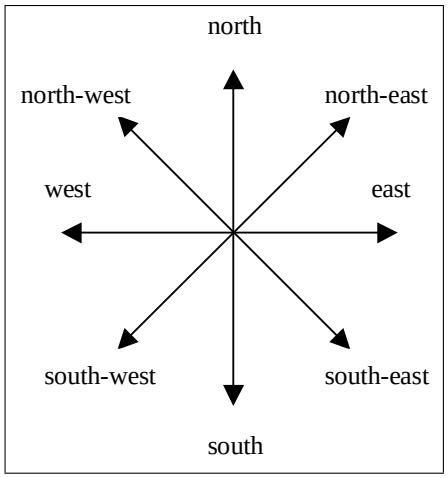


Figure 6.1: Directional Coordinate System

distances between the directional and displacement components of the trajectories when both lists are available. If a displacement list is not specified in a query, then trajectory similarity is measured by comparing the directional lists. Furthermore, when a weight value is 0, then its corresponding list is not taken into account in computing similarity between trajectories.

Directional Similarity:

Definition 3 *A directional region is an area between neighboring directional segments in the directional coordinate system depicted in Figure 6.1.*

Definition 4 *Let d_a and d_b be two directions in the directional coordinate system. The distance between d_a and d_b, denoted as $D(d_a, d_b)$, is defined to be the minimum number of directional regions between d_a and d_b.*

Definition 5 *The directional normalization factor, ω, is defined to be the number of directional regions between two opposite directions in the directional coordinate system ($w = 4$).*

Let A and B be two directional lists each having n elements such that A = $[A_1, A_2, ..., A_n]$ and B = $[B_1, B_2, ..., B_n]$. The similarity between A

and B is specified as follows according to Definitions 3-5:

$$\varsigma(A,B) = 1 - \frac{1}{w}\sqrt{\frac{1}{n}\sum_{i=1}^{n} D(A_i,B_i)^2} \tag{6.1}$$

Displacement Similarity:

Definition 6 *The displacement normalization factor of a displacement list A is defined to be the maximum displacement value in the list and it is denoted by* A_μ.

Let A and B be two displacement lists each having n elements such that A = [A_1, A_2, ..., A_n] and B = [B_1, B_2, ..., B_n]. Furthermore, let us suppose that $D_{nr}(A_i, B_i)$ denotes the normalized distance between A_i and B_i for 1≤i≤n. Then, the similarity between A and B is specified as follows:

$$\varsigma(A,B) = 1 - \sqrt{\frac{1}{n}\sum_{i=1}^{n} D_{nr}(A_i,B_i)^2} \quad , where \quad D_{nr}(A_i,B_i) = \frac{B_\mu A_i - A_\mu B_i}{A_\mu B_\mu} \tag{6.2}$$

Trajectory Matching:

Similarity-based object-trajectory queries are processed by the trajectory processor, which takes such queries as input and returns a set of intervals each associated with an importance value (similarity value), along with some other data needed by the query processor for forming the final set of answers to user queries, such as variable bindings (values) if variables are used. Here, we formally discuss how similarity-based object-trajectory queries with no variables are processed by the trajectory processor. In doing so, it is assumed without loss of generality that trajectory queries contain both the directional and displacement lists. Moreover, we restrict our discussion to such cases as those where the time gaps between consecutive object movements in trajectory facts are equal to or below the time threshold given in a query. These assumptions are made just for the sake of simplicity because our main target here is to explain the theory that provides a novel framework for our similarity-based object-trajectory matching mechanism rather than presenting our query processing algorithm in detail.

Let Q and T be a similarity-based object-trajectory query and a trajectory fact for an object that is stored in the facts-base for a video clip, respectively, such that Q = `tr`(α, λ) `sthreshold` σ `dirweight` β and T = (ν, φ, ψ, κ), where $\lambda = [\theta, \chi]$. Let us assume that there is no variable used in Q or all variables are bound, $\alpha = \nu$, $\|\varphi\|$ = n and $\|\theta\|$ = m. Let us also assume that there is no gap between any consecutive pairs of intervals in κ such that $\kappa_{e_i} = \kappa_{s_{i+1}}$ (1≤i<m).

Case 1 (n=m): The similarity between the two trajectories $Q_t = (\theta, \chi)$ and $T_t = (\varphi, \psi)$ is computed as follows:

$$\varsigma(Q_t, T_t) = \beta\,\varsigma(\theta, \varphi) + \eta\,\varsigma(\chi, \psi), \quad where \quad \beta = 1 - \eta \tag{6.3}$$

In this case, the trajectory processor returns only one interval, $\xi = [\kappa_{s_1}, \kappa_{e_n}]$, iff $\varsigma(Q_t, T_t) \geq \sigma$. Otherwise ($\varsigma(Q_t, T_t) < \sigma$), the answer set is empty because there is no similarity between Q_t and T_t with a given threshold σ.

Case 2 (n>m): In this case, the trajectory processor returns a set of intervals ϕ such that

$$\phi = \{[s_i, e_i] | 1 \leq i \leq n-m+1 \wedge s_i = \kappa_{s_i} \wedge e_i = \kappa_{e_{i+m-1}} \wedge \varsigma(Q_t, T_{t_{[i,i+m-1]}}) \geq \sigma\}$$

where

$$T_{t_{[i,i+m-1]}} = ([\varphi_i, ..., \varphi_{i+m-1}], [\psi_i, ..., \psi_{i+m-1}])$$

If there is no match found for any T_{t_i} for 1≤i≤n-m+1, where $T_{t_i} = T_{t_{[i,i+m-1]}}$, then the answer set is empty.

Case 3 (n<m): As in Case 1, the trajectory processor returns only one interval, $\xi = [\kappa_{s_1}, \kappa_{e_n}]$,

$$\text{iff} \quad \exists\, \varsigma(Q_{t_{[i,i+n-1]}}, T_t) \geq \frac{m}{n}\,\sigma \quad \text{for} \quad 1 \leq i \leq m - n + 1,$$

where

$$Q_{t_{[i,i+n-1]}} = ([\theta_i, \ldots, \theta_{i+n-1}], [\chi_i, \ldots, \chi_{i+n-1}])$$

The importance value (similarity value) associated and returned with ξ is

$$\varsigma = \frac{n}{m}\, MAX\, \{\varsigma | \varsigma(Q_{t_{[i,i+n-1]}}, T_t)\ (1 \leq i \leq m - n + 1)\}$$

If there is no match found, the answer set is empty because there is no similarity between Q_t and T_t with a given threshold σ.

Projection Operator:

BilVideo query language provides a trajectory-projection operator, *project*(α [, β]), to extract subtrajectories from trajectory facts, where α is an object identifier for which a variable might be used and β is an optional condition. If a condition is not given, then the operator returns the entire trajectory that an object follows in a video clip. Otherwise, subtrajectories of an object, where the given condition is satisfied, are returned. Hence, the output of *project* is a set $\vartheta = \{\lambda \mid \lambda = [\theta, \chi]\}$ where λ is a trajectory, and θ and χ are the directional and displacement components of λ, respectively. The condition, if it is given, is local to *project* and it is of type `<spatial-condition>` as specified in Appendix B.

6.2.4 Temporal Queries

This type of queries is used to specify the order of occurrence of conditions in time. Conditions may be of any type, but temporal operators process their arguments only if they contain intervals. *BilVideo* query language implements all temporal relations as temporal operators defined by Allen's temporal interval algebra except for *equal*: our interval operator *and* yields the same functionality as that of *equal*. Supported temporal operators, which are used as interval operators in *BilVideo* query language, are *before*, *meets*, *overlaps*, *starts*, *during*, *finishes* and their inverse operators. A user query may contain repeating temporal conditions specified by `repeat` with an optional repetition number given. If `tgap` is not provided with `repeat`, then its default value for temporal operators (equivalent to one frame when converted) is assumed.

6.2.5 Aggregate Queries

This type of queries may be used to retrieve statistical data about objects and events in video data. There are three aggregate functions, *average*, *sum* and *count* as explained in Section 6.1. These aggregate functions may be very attractive in collecting statistical data for such applications as sports event analysis systems.

6.2.6 Low-level (Color, Shape and Texture) Queries

This type of queries is used to query video data by low-level (color, shape and texture) properties. Among all, color and shape are the most frequently used properties first adopted in image databases. Since a video can be considered as a sequence of images, the techniques developed for content-based retrieval of images by color, shape and texture features can also be used in video databases to enhance their query capabilities and provide more accurate answers for user queries.

In *BilVideo*, video data is preprocessed to semi-automatically extract object trajectories, object-appearance relations and spatio-temporal relations between video objects by the Fact-Extractor tool. In our video data model, video clips are segmented into shots whenever the current set of relations between objects changes and the video frames, where these changes occur, are chosen as keyframes. We have enhanced the Fact-Extractor tool so that it could also generate color and shape histograms of video salient objects for each keyframe detected. Extracted color and shape histograms of video objects will be used together with spatio-temporal relations to enrich query capabilities of the system. Details about our work on using low-level (color, shape and texture) features for video data can be found in [6].

6.2.7 Semantic Queries

This type of queries is used to query video data by semantic features. In our system, videos are partitioned into semantic units, which form a hierarchy. This partitioning is carried out by the Video-Annotator tool. Our approach of segmenting videos into shots by detecting keyframes, where the set of spatial relations between objects changes, should not be confused by semantic partitioning of videos. The former approach is only used for extracting spatio-temporal relations between video objects and answering spatio-temporal queries on video data. Spatio-temporal relations stored in the knowledge-base are not explicitly related to any semantic video unit, and this feature makes the system more powerful than others proposed in the literature because *BilVideo* can return not only semantic units of videos but also any segment of videos, where given conditions are satisfied. *BilVideo* query language has been extended for semantic conditions on video data and semantic query processing is under development.

6.3 Example Applications

To demonstrate the capabilities of *BilVideo* query language for spatio-temporal queries, three application areas, *soccer event analysis*, *bird migration tracking* and *movie retrieval systems*, have been selected. However, *BilVideo* system architecture and query language provide a generic framework to be used for other types of applications, as well.

6.3.1 Soccer Event Analysis System

A soccer event analysis system may be used to collect statistical data on events that occur during a soccer game, such as finding the number of goals, offsides and passes, average ball control time for players, etc., and to retrieve video segments where such events take place. *BilVideo* query language can be used to answer such queries provided that some necessary facts, such as players and goalkeepers

for the teams, as well as some predicates, such as *player* to find the players of a certain team, are added to the knowledge-base.

Query 1: "Find the number of direct shots to the goalkeeper of *Liverpool* by each player of *Manchester United* in a given video clip, and return such video segments."

This query can be specified in *BilVideo* query language as follows:

```
select count(segment), segment, X
from 1
where goalkeeper(X, liverpool) and player(Y, manchester)
  and touch(Y, ball) meets not(touch(Z, ball)) meets
  touch(X, ball);
```

In this query, the external predicates are *goalkeeper* and *player*, and the video identifier for the given video clip is assumed to be 1. For each player of *Manchester United* found in the specified video clip, the number of direct shots to the goalkeeper of *Liverpool* by the player, along with the player's name and video segments found, is returned provided that such segments exist. In *BilVideo*, semantic meta data is stored in an object-relational database. Hence, video identifiers can be retrieved from this database, querying it with some descriptional data.

Query 2: "Find the average ball control (play) time for each player of *Manchester United* in a given video clip."

This query can be specified in *BilVideo* query language as follows:

```
select average(segment), X
from 1
where player(X, manchester) and touch(X, ball);
```

In answering this query, it is assumed that when a player touches the ball, it is in his control. Then, the ball control time for a player is computed with respect to the time interval during which he is in touch with the ball. Hence, the average ball control time for a player is simply the sum of all time intervals where the player is in touch with the ball divided by the

number of these time intervals. This value is computed by the aggregate function *average*. If we were interested in finding the total ball control time, then we would use the aggregate function *sum* instead of *average*.

Query 3: "Find the number of goals of *Liverpool* scored against *Manchester United* in a given video clip."

This query can be specified in *BilVideo* query language as follows:

```
select count(segment)
from 1
where samelevel(ball, net) and touch(ball, net) meets
  overlap(ball, net);
```

In this query, 3D relation *samelevel* ensures that an event, which is not a goal because the ball does not go into the net, but rather passes somewhere near the net, is not considered as a goal. The ball may first touch and then overlap with the net in 2D space while it is behind or in front of the net on the z-axis of three-dimensional space. Hence, by using 3D relation *samelevel*, such false events are discarded.

6.3.2 Bird Migration Tracking System

A bird migration tracking system is used to determine the migration paths of birds over a set of regions in certain times. In [36], an animal movement querying system is discussed, and we have chosen a specific application of such a system to show how *BilVideo* query language might be used to answer spatio-temporal, especially object-trajectory, queries on the migration paths of birds.

Query 1: "Find the migration paths of *bird* o_1 over *region* r_1 in a given video clip."

This query can be specified in *BilVideo* query language as follows:

```
select X
from 2
where X = project(o1, inside(o1, r1));
```

In this query, X is a variable used for the trajectory of *bird* o_1 over *region* r_1. Video identifier of the video clip, where the migration of *bird* o_1 is recorded, is assumed to be 2. This query returns the paths *bird* o_1 follows when it is inside *region* r_1.

Query 2: "How long does *bird* o_1 appear inside *region* r_1 in a given video clip?"

This query can be specified in *BilVideo* query language as follows:

```
select sum(segment)
from 2
where inside(o1, r1);
```

The result of this query is a time value, which is computed by the aggregate function *sum* adding up the time intervals during which *bird* o_1 is inside *region* r_1.

Query 3: "Find the video segments where *bird* o_1 enters *region* r_1 from west and leaves from north in a given video clip."

This query can be specified in *BilVideo* query language as follows:

```
select segment
from 2
where (touch(o1, r1) and west(o1, r1)) meets
  overlap(o1, r1) meets coveredby(o1, r1) meets
  inside(o1, r1) meets
  coveredby(o1, r1) meets overlap(o1, r1) meets
  (touch(o1, r1) and north(o1, r1));
```

Query 4: "Find the names of birds following a similar path to that of *bird* o_1 over *region* r_1 with a similarity threshold value 0.9 in a given video clip, and return such segments."

This query can be specified in our query language as follows:

```
select segment, X
from 2
where Y = project(o1, inside(o1, r1)) and inside(X, r1) and
  X != o1 and tr(X, Y) sthreshold 0.9;
```

Here, X and Y are variables representing the bird names and subtrajectories of *bird* o_1 over *region* r_1, respectively. Projected subtrajectories of *bird* o_1, where the given condition is to be inside *region* r_1, are used to find similar subtrajectories of other birds over the same region.

6.3.3 Movie Retrieval System

A movie retrieval system contains movies and series from different categories, such as cartoon, comedy, drama, fiction, horror, etc. Such a system may be used to retrieve videos or segments from a collection of movies with some spatio-temporal, semantic and low-level (color, shape and texture) conditions given. In this section, a specific episode of *Smurfs*, a cartoon series, titled as *Bigmouth's Friend* is used for all query examples given. The video identifier of this episode is assumed to be 3.

Query 1: "Find the segments from *Bigmouth's Friend* where *Bigmouth* is below *RobotSmurf* while *RobotSmurf* starts moving towards west and then goes to east repeating this as many times as it happens in the video clip."

```
select segment
from 3
where below(bigmouth, robotsmurf) and
  (tr(bigmouth, [west, east])) repeat;
```

Query 2: "Find the segments from *Bigmouth's Friend* where *Gargamel* is to the southwest of his *father* and *boyking* while *boyking* is to the right of *soldier1* and left of *soldier2*, and *soldier1* is behind *soldier2*."

```
select segment
from 3
where southwest(gargamel, father) and
  southwest(gargamel, boyking) and right(boyking, soldier1)
  and left(boyking, soldier2) and behind(soldier1, soldier2);
```

Query 3: "Find the segments from *Bigmouth's Friend* where *lazysmurf*, *farmersmurf*, *grouchysmurf*, *smurfette* and *handysmurf* all appear together such that *lazysmurf* is to the west of *handysmurf* and *smurfette* is to the east of *farmersmurf*."

```
select segment
from 3
where appear(grouchysmurf) and west(lazysmurf, handysmurf) and
  east(smurfette, farmersmurf);
```

In this query, there is no need to ask if *lazysmurf*, *handysmurf*, *smurfette* and *farmersmurf* appear together with *grouchysmurf* because the conditions *west* and *east* imply this condition.

Query 4: "Find the segments from *Bigmouth's Friend* where *robotsmurf* and *bigmouth* are disjoint and *robotsmurf* is to the right of *bigmouth* while there is no other object of interest that appears."

```
select segment
from 3
where disjoint(robotsmurf, bigmouth) and
  right(robotsmurf, bigmouth) and
  appear_alone(robotsmurf, bigmouth);
```

In this query, *appear_alone* is an external predicate defined in the knowledgebase as follows:

```
appear_alone(X, Y, F) :- keyframes(L1), member(F, L1),
  findall(W, p_appear(W, F), L2), length(L2, 2),
  forall(member(Z, L2), (Z = X; Z = Y)).
```

Chapter 7

Query Processor

Figure 7.1 illustrates how the query processor communicates with Web clients and the underlying system components to answer user queries. The phases of query processing for spatio-temporal queries are shown in Figure 7.2. Web clients make a connection request to the query request handler, which creates a process for each request passing a new socket for communication between the process and the Web client. Then, the clients send user queries to the processes created by the query request handler. If the queries are specified visually, they are transformed into SQL-like textual query language expressions before being sent to the server. Having received the query from the client, each process calls the query processor, compiled as a library, with a query string and waits for the query answer. When the query processor returns, the process communicates the answer to the Web client issuing the query and exits. The query processor first separates the semantic (keyword, event/activity and category-based) and low-level (color, shape and texture) query conditions in a query from the spatio-temporal query conditions that could be handled by the knowledge-base. The former type of conditions is organized and sent as regular SQL queries to an object-relational database whereas the latter part is reconstructed as Prolog-type knowledge-base queries. Intermediate results obtained are integrated by the query processor and returned to the query request handler, which communicates the final results to Web clients. Currently, the query processor can handle a wide range of spatio-temporal queries and we are working on extending it to support semantic and low-level queries, as

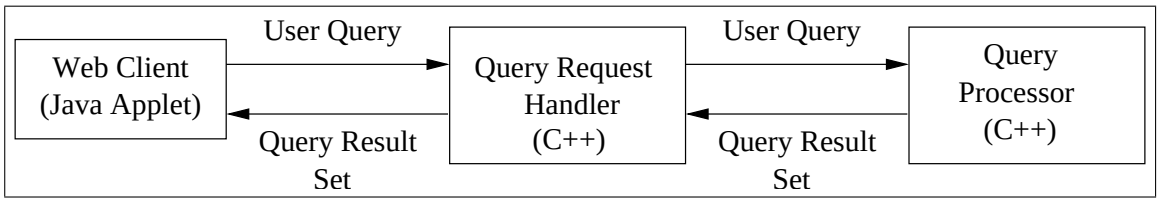


Figure 7.1: Web Client - Query Processor Interaction

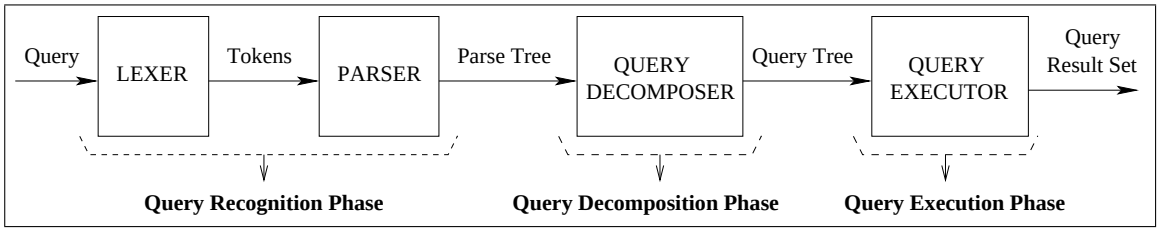


Figure 7.2: Query Processing Phases

well.

7.1 Query Recognition

The lexical analyzer and parser for the query language were implemented using Flex and Bison that work under a Linux operating system [11, 42], which are the GNU versions of the original Lex&Yacc [20, 23] compiler-compiler generator tools. 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*.

7.2 Query Decomposition

The parse tree generated after the query recognition phase is traversed in a second phase, which we call *query decomposition phase*, to construct a query tree. The query tree is constructed from the parse tree decomposing a query into three basic types of subqueries: *Prolog queries* or *maximal subqueries* that can be directly sent to the inference engine Prolog, *trajectory-projection queries* that are handled by the trajectory projector and *similarity-based object-trajectory queries* that are processed by the trajectory processor. Temporal queries are handled by interval-operator functions such as *before*, *during*, etc. Arguments of the interval operators are handled separately because they should be processed before the interval operators are applied. Since a user may give any combination of spatial, temporal, object-appearance, external-predicate, trajectory-projection and similarity-based object-trajectory conditions in any order while specifying a query, a query is decomposed in such a way that a minimum number of subqueries are formed. This is achieved by grouping the Prolog-type predicates into maximal subqueries without changing the semantic meaning of the original query.

7.3 Query Execution

The input for the *query execution phase* is a query tree. In this phase, the query tree is traversed in postorder, executing each subquery separately and performing interval processing in internal nodes so as to obtain the final set of results. Since it would be inefficient and very difficult, if not impossible, to fully handle spatio-temporal queries by Prolog alone, *query execution phase* is mainly carried out by some efficient C++ code. Thus, Prolog is utilized only to obtain intermediate answers to user queries from the facts-base. Intermediate query results returned by Prolog are further processed and final answers to user queries are formed after interval processing. Figure 7.3 illustrates the *query execution phase*.

BilVideo query language has been designed to return variable values, when requested explicitly, as part of the result as well. Therefore, the system not only

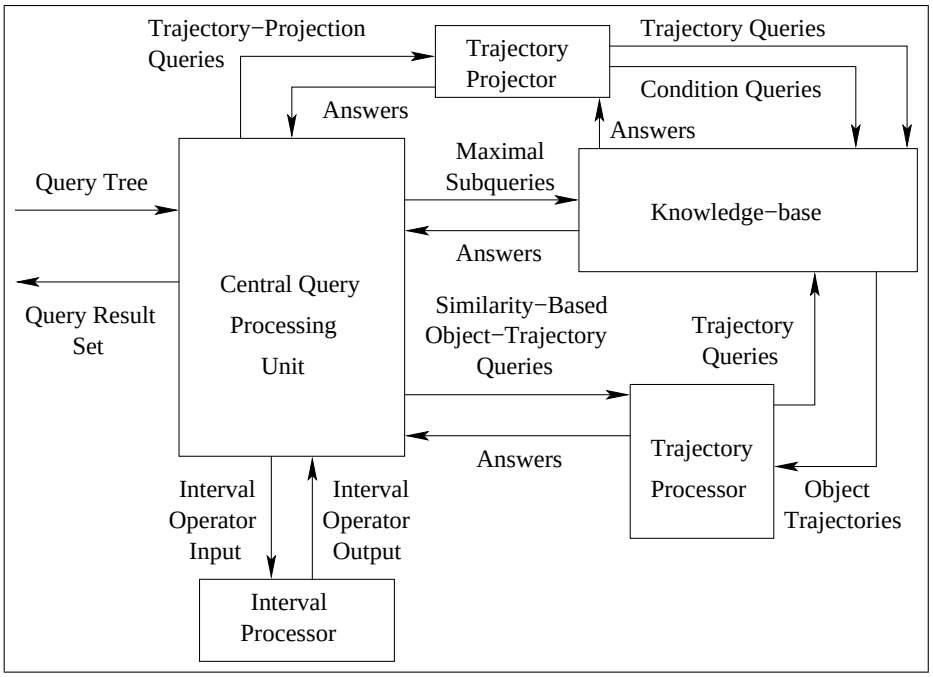


Figure 7.3: Query Execution

supports video/segment queries but also variable-value retrieval for the parts of videos satisfying given query conditions utilizing a knowledge-base. Variables may be used for object identifiers and trajectories.

One of the main challenges in query execution is to handle such user queries where the scope of a variable used extends to several subqueries after the query is decomposed. It is a challenging task because subqueries are processed separately, accumulating and processing the intermediate results along the way to form the final set of answers. Hence, the values assigned to variables for a subquery are retrieved and used for the same variables of other subqueries within the scope of these variables. Therefore, it is necessary to keep track of the scope of each variable for a query. This scope information is stored in a hash table generated for the variables. Dealing with variables makes the query processing much harder, but it also empowers the query capabilities of the system and yields much richer semantics for user queries.

7.4 Query Examples

In this section, three example spatio-temporal queries are given to demonstrate how the query processor decomposes a query into subqueries. Intermediate results obtained from these subqueries are integrated step by step to form the final answer set.

Query 1: `select segment, X, Y`
`from all`
`where west(X, Y) and west(Y, `o_1`) and west(`o_1`, `o_2`) and`
`tr(`o_2`, [[west, east], [24, 40]]) sthreshold 0.4`
`dspweight 0.3 and disjoint(X, Y) before touch(X, Y)`
`and disjoint(Y, `o_1`);`

This example query is decomposed into following subqueries:

Subquery 1: `tr(b, [[west, east], [24, 40]]) sthreshold 0.4`
`dspweight 0.3`

Subquery 2: `disjoint(X, Y)`

Subquery 3: `touch(X, Y)`

Subquery 4: `west(X, Y) and west(Y, `o_1`) and west(`o_1`, `o_2`) and`
`disjoint(Y, `o_1`)`

In this example, subqueries 2 and 3 are linked to each other by temporal operator *before*. The rest of the internal nodes in the query tree contains operator *and*. Figure 7.4 depicts the query tree constructed for this example query.

Query 2: `select segment, Y`
`from all`
`where west(X, Y) and west(Y, `o_1`) and`
`tr(`o_2`, [[west, east], [24, 40]]) sthreshold 0.4`
`dirweight 0.4 and disjoint(Y, `o_1`);`

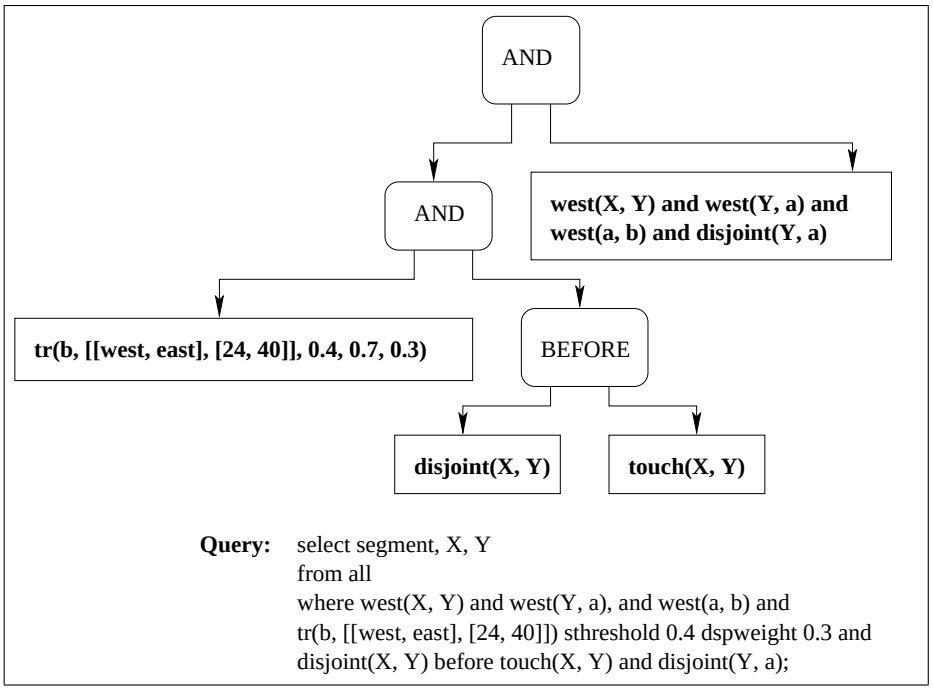


Figure 7.4: The query tree constructed for Query 1

Query 2 is decomposed into following subqueries:

Subquery 1: `tr(`o_2`, [[west, east], [24, 40]]) sthreshold 0.4 dirweight 0.4`

Subquery 2: `west(X, Y) and west(Y,` o_1`) and disjoint(Y,` o_1`)`

To answer Query 1, the query processor computes each subquery traversing the query tree in postorder performing interval processing at each internal node and taking into account the scope of each variable encountered. Here, the scope of object variables X and Y is subqueries 2, 3 and 4. Hence, for each value-pair of variables X and Y, a set of intervals is computed in subquery 2. Another reason for computing a set of intervals for each value-pair is that the values obtained for variables X and Y are also returned in pairs, along with the video segments satisfying the query conditions, as part of the query results. Hence, even if the scope of these variables were to be only subquery 2, the same type of interval

processing and care must be provided. Nonetheless, if an object variable is bound by only one subquery and its values are not to be returned as part of the query result as in the case of object variable X in Query 2, then it is possible to combine consecutive intervals, where the variable takes different values while the rest of the conditions are satisfied for the same set of value-sequences for the rest of the variables. Query 3 better explains this concept of interval processing and variable value computation:

Query 3: "Return video segments in the database where object o_1 is first disjoint from object o_2 and then touches it repeating this event 3 times while it is inside another object."

```
select segment
from all
where inside(o1, X) and
   (disjoint(o1, o2) meets touch(o1, o2)) repeat 3;
```

In this query, we do not care which object object o_1 is inside, but we are only interested in finding the video segments where object o_1 is first disjoint from object o_2 and then touches it repeating this event 3 times while it is inside another object. Thus, consecutive intervals for different objects that contain object o_1 may be combined provided that given conditions are satisfied.

Chapter 8

Spatio-Temporal Query Processing

BilVideo query language currently provides support for spatio-temporal video queries that contain *Prolog*, *similarity-based object-trajectory* and *trajectory-projection* types of subqueries. The query processor makes use of the query tree constructed from the parse tree, which is formed in the process of query parsing. Since it is relatively inefficient and difficult to handle similarity-based object-trajectory, temporal-predicate and trajectory-projection query conditions by Prolog alone, these conditions are mainly processed by some C++ code outside the Prolog environment. However, Prolog is still used for such types of conditions to obtain some necessary data from the knowledge-base, such as object trajectories, which is needed for query processing. *BilVideo* query language has been designed to return object names, which satisfy the given query conditions, as part of the result as well. This feature of the system and the language is unique because the system not only supports video/video-segment queries but also object information (name) retrieval for the parts of videos satisfying given query conditions utilizing a knowledge-base. The main challenge here is to handle such user queries when the scope of an object variable used in a query extends to several subqueries after the query is decomposed. Hence, it is necessary to keep track of the scope of each variable for a query. This task makes the query processing and especially the interval processing much harder than it would be without

object-information-retrieval support implemented, but it also broadens the set of query types that the system can answer and yields much richer semantics for user queries. Some of our spatio-temporal query processing functions are given in the form of simplified pseudo-codes in Appendix C.

8.1 Interval Processing

Intervals are categorized into two types: *non-atomic* and *atomic* intervals. If a condition holds for every frame of a part of a video clip, then the interval representing an answer for this condition is considered as a non-atomic interval. Non-atomicity implies that for every frame within an interval in question does the condition hold. Hence, the condition also holds for any subinterval of a non-atomic interval as well. This implication is not correct for atomic intervals, though. The reason is that the condition associated with an atomic interval does not hold for all its subintervals. Consequently, an atomic interval cannot be broken into its subintervals for query processing. On the other hand, subintervals of an atomic interval are populated for query processing provided that conditions are satisfied in their range. In other words, the query processor generates all possible atomic intervals for which the given conditions are satisfied. This interval population is necessary since atomic intervals cannot be broken into subintervals and all such intervals, where the conditions hold, should be generated for query processing. The intervals returned by the Prolog queries that contain topological, directional, object-appearance, external-predicate and 3D-relation conditions are non-atomic whereas those obtained by applying the temporal predicate functions to the interval sets as well as those returned by the similarity-based object-trajectory function are atomic intervals. As the logical operators *AND*, *OR* and *NOT* are considered as interval operators when their arguments contain intervals to process, they also work on intervals. The operators *AND* and *OR* may return atomic and/or non-atomic intervals depending on the types of their input intervals. The operator *AND* takes the intersection of its input intervals while the operator *OR* performs a union operation on its input intervals. The unary operator *NOT* returns the complement of its input interval

set with respect to the video clip being queried, and the intervals in the result set are of type non-atomic regardless of the types of the input intervals. Semantics of the interval intersection and union operations are given in Tables 8.1 and 8.2, respectively.

The rationale behind classifying video frame intervals into two categories as atomic and non-atomic may be best described with the following query example: "*Return the video segments in the database, where object A is to the west of object B and object A follows a similar trajectory to the one specified in the query with respect to the similarity threshold given*". Let us assume that the intervals [10, 200] and [10, 50] are returned as part of the answer set for a video for the trajectory and spatial (directional) conditions of this query, respectively. Here, the first interval is of type atomic because the trajectory of object A is only valid within the interval [10, 200], and therefore, trajectory similarity computation is not performed for any of its subintervals. However, the second interval is non-atomic since the directional condition given is satisfied for each frame in this interval. When these two intervals are processed to form the final result by the *AND* operator, no interval is returned as an answer because there is no such an interval, where both conditions are satisfied together. If there were no classification of intervals and all intervals were to be breakable into subintervals, then the final result set would include the interval [10, 50]. However, as obvious, the two conditions cannot hold together in this interval due to the fact that the trajectory of object A spans over the interval [10, 200]. As another case, let us suppose that the intervals [10, 200] and [10, 50] are returned as part of the answer set for the spatial (directional) and trajectory conditions of this query, respectively, and the intervals were to be unbreakable to sub-intervals. Then, the result set would be empty for these two intervals. This is not correct since there is an interval, [10, 50], where both conditions hold. These two cases clearly show that intervals must be classified into two groups as atomic and non-atomic for query processing. Following is a discussion with another example query that has a temporal predicate provided to make all these concepts much clearer.

Let us suppose that a user wants to find the parts of a video clip satisfying the following query:

Input Interval 1	Input Interval 2	Result Set	Result Interval Type
I_1 (Atomic)	I_2 (Atomic)	I_1 iff $I_1 \supseteq I_2$ I_2 iff $I_1 \subset I_2$ otherwise, Ø	Atomic
I_1 (Atomic)	I_2 (Non-atomic)	I_1 iff $I_2 \supseteq I_1$ otherwise, Ø	Atomic
I_1 (Non-atomic)	I_2 (Atomic)	I_2 iff $I_1 \supseteq I_2$ otherwise, Ø	Atomic
I_1 (Non-atomic)	I_2 (Non-atomic)	$[I_s, I_e]$ iff I_1 overlaps I_2 $I_s = I_{1_s}$ iff $I_{1_s} \geq I_{2_s}$ otherwise, $I_s = I_{2_s}$ $I_e = I_{1_e}$ iff $I_{1_e} \leq I_{2_e}$ otherwise, $I_e = I_{2_e}$ otherwise, Ø	Non-atomic

Table 8.1: Interval Intersection (AND)

Input Interval 1	Input Interval 2	Result Set	Result Interval Type
I_1 (Atomic)	I_2 (Atomic)	$\{I_1, I_2\}$	Atomic
I_1 (Atomic)	I_2 (Non-atomic)	$\{I_1, I_2\}$	Atomic and Non-atomic
I_1 (Non-atomic)	I_2 (Atomic)	$\{I_1, I_2\}$	Non-atomic and Atomic
I_1 (Non-atomic)	I_2 (Non-atomic)	$[I_{1_s}, I_{2_e}]$ if $I_{2_s} = I_{1_e} + 1$ $[I_{2_s}, I_{1_e}]$ if $I_{1_s} = I_{2_e} + 1$ $[I_s, I_e]$ if I_1 overlaps I_2 $I_s = I_{1_s}$ iff $I_{1_s} \geq I_{2_s}$ otherwise, $I_s = I_{2_s}$ $I_e = I_{1_e}$ iff $I_{1_e} \leq I_{2_e}$ otherwise, $I_e = I_{2_e}$ otherwise, $\{I_1, I_2\}$	Non-atomic

Table 8.2: Interval Union (OR)

Query: (A before B) and west(x, y), where A and B are Prolog subqueries, and x and y are atoms (constants).

The interval operator "before" returns a set of atomic intervals, where first A is true and B is false, and then, A is false and B is true in time. If A and B are true in the intervals [4, 10] and [20, 30], respectively, and if these two intervals are both non-atomic, then the result set will consist of [10, 20], [10, 21], [9, 20], [10, 22], [9, 21], ..., [4, 30]. Now, let's discuss two different scenarios:

Case 1: west(x, y) holds for [9, 25]. This interval is non-atomic because "west(x, y)" returns non-atomic intervals. If the operator "before" returned only the atomic interval [4, 30] as the answer for "A before B", then the answer set to the entire query would be empty. However, the user is interested in finding the parts of a video clip, where "(A before B) and west(x, y)" is true. The intervals [10, 20], [10, 21], ..., [4, 29] also satisfy "A before B"; however, they would not be included in the answer set for "before". This is wrong! All these intervals must be a part of the answer set for "before" as well. If they are included, then the answer to the entire query will be [9, 25] because [9, 25] (atomic) and [9, 25] (non-atomic) => [9, 25] (atomic). Nonetheless, make a note of that such intervals as [10, 19], [11, 25], etc. are not included in the answer set of "A before B" since they do not satisfy the condition "A before B".

Case 2: west(x, y) holds for [11, 25]. Let's suppose that "before" returned non-atomic intervals rather than atomic intervals and that the answer for "A before B" were [4, 30]. Then, the answer to the entire query would be [11, 25] for [4, 30] (non-atomic) and [11, 25] (non-atomic) => [11, 25] (non-atomic). Nevertheless, this is wrong due to the fact that "A before B" is not satisfied within this interval. Hence, "before" should return atomic intervals so that such incorrect results are not produced.

These two cases clearly show that temporal operators should return atomic intervals and that the results should also include subintervals of each largest

interval that satisfy the given conditions rather than consisting only of the set of largest intervals. It also demonstrates why such a classification for the intervals as atomic and non-atomic is necessary.

Chapter 9

Performance and Scalability Experiments

In order to show that *BilVideo* is scalable for spatio-temporal queries in terms of the number of salient objects per frame and the total number of frames in a video clip as well as to demonstrate the space savings due to our rule-based approach, some program-generated synthetic video data was used. These tests constitute the first part of the overall tests. In the second part, the system's performance was tested on some real video clip fragments with the consideration of space and time efficiency criteria to show its applicability for spatio-temporal queries in real-life applications. The real video clip fragments were extracted from *jornal.mpg*, MPEG-7 Test Data set CD-14, Port. news, and a *Smurfs* cartoon episode named *Bigmouth*'s *Friend*. Table 9.1 presents some information about these video fragments.

In order to make a judgement on how successful our fact-extraction algorithm

Original Video	Total # of Frames	Total # of Objects	Max. # of Objects in a Frame
Jornal.mpg	5254	21	4
Smurfs.avi	4185	13	6

Table 9.1: Specifications of the movie fragments

is in eliminating the redundant facts for both synthetic and real video data, two facts-bases were created. The first facts-base consists only of the basic facts extracted using the fact-extraction algorithm while the second one comprises all the facts computed again with this algorithm, but this time with its fact-reduction feature turned off. Since the two facts-bases were created using the same video data for synthetic and real video separately, the sizes of the resultant facts-bases give us an idea about how well our fact-reduction feature works as well as how efficient our approach is for space considerations.

9.1 Tests with Program-Generated Video Data

For the space efficiency tests, the number of objects per frame was selected as 8, 15 and 25 while the total number of frames was fixed to 100. To show the system's scalability in terms of the number of objects per frame, the total number of frames was chosen to be 100 and the number of objects per frame was varied from 4 to 25. For the scalability test with respect to the total number of frames, the number of objects was fixed to 8 whilst the total number of frames was varied from 100 to 1000.

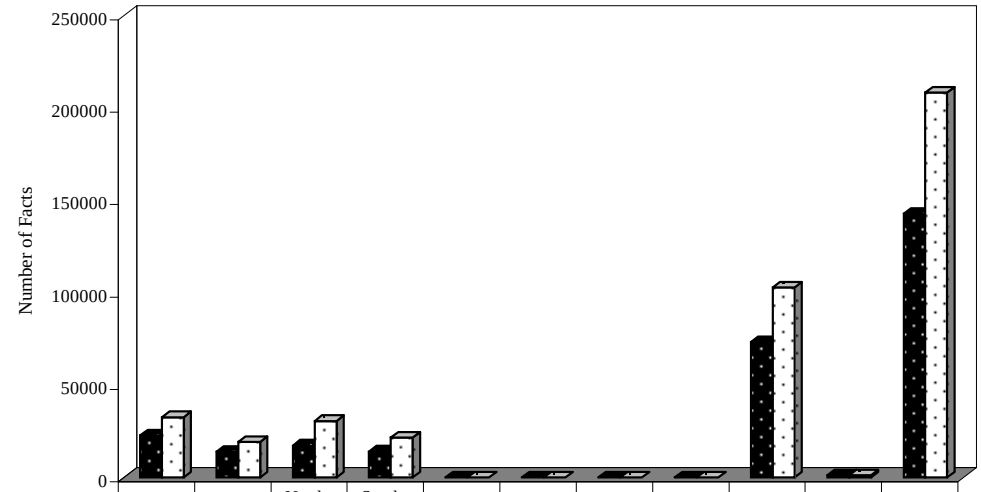
Figures 9.1-9.3 give the space efficiency test results as bar charts for 8, 15 and 25 objects per frame for a 1000-frame synthetic video data. In these figures, *Facts-base 1* is the facts-base with redundant facts eliminated whereas *Facts-base 2* is the other facts-base that contains all the relations computed by the fact-extraction algorithm with its fact-reduction feature turned off. The numbers corresponding to these fields present the number of facts stored for each relation separately and in total for all relations in respective facts-bases.

Four different types of queries were used for the scalability tests by taking four possible combinations of object-variable unifications that can be used to query the system. The queries are based on the *west* and *disjoint* relations and they are given in Table 9.2.

In the first part of the tests, where the system's scalability in terms of the

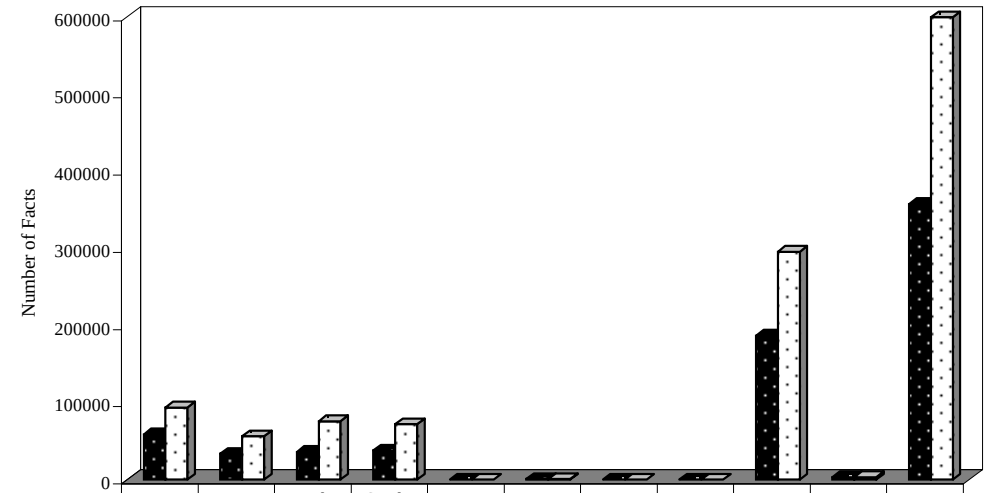
	West	South	North-west	South-west	Equal	Cover	Inside	Touch	Disjoint	Overlap	All
Facts-base 1	7240	4290	6018	4257	0	79	16	1	22068	383	44352
Facts-base 2	9141	5300	7944	5391	0	79	16	1	27101	383	55160

Figure 9.1: Space Efficiency Test Results (8 Objects and 1000 Frames)



	West	South	North-west	South-west	Equal	Cover	Inside	Touch	Disjoint	Overlap	All
Facts-base 1	22763	13910	17274	14079	0	114	50	35	73392	1129	142746
Facts-base 2	32626	19322	30560	21652	0	114	50	35	102832	1129	208320

Figure 9.2: Space Efficiency Test Results (15 Objects and 1000 Frames)



	West	South	North-west	South-west	Equal	Cover	Inside	Touch	Disjoint	Overlap	All
Facts-base 1	58765	33528	36022	37987	0	592	274	188	186538	3235	357129
Facts-base 2	93620	56442	75875	71763	0	596	274	188	295392	3250	599400

Figure 9.3: Space Efficiency Test Results (25 Objects and 1000 Frames)

X	Y	Query Format
Not Unified	Not Unified	west(X, Y, F) $\wedge$ disjoint(X, Y, F)
Unified	Not Unified	west(1, Y, F) $\wedge$ disjoint(1, Y, F)
Not Unified	Unified	west(X, 7/0, F) $\wedge$ disjoint(X, 7/0, F)
Unified	Unified	west(1, 7/0, F) $\wedge$ disjoint(1, 7/0, F)

Table 9.2: Queries for the Scalability Tests

number of objects per frame was checked, 1 and 7 were used in queries as object identifiers while for the second part, in which the system was tested for its scalability on the total number of frames, 1 and 0 were selected as object identifiers. In our test data, integer identifiers were used for each object for the sake of simplicity, but in real video, salient objects are annotated by some meaningful textual names they can be remembered with. In our tests, each query returns non-empty results. Figures 9.4-9.11 provide the graphs obtained from the tests.

9.2 Tests with Real Video Data

We present our space efficiency test results as bar charts in Figures 9.12 and 9.13 for the video fragments taken from *jornal.mpg* and *smurfs.avi*, respectively. For the time efficiency tests, four queries were used for each of the video fragments. The queries used on the news report video fragment are as follows:

Query 1: Show the fragments of the clip where *priest*, *interviewee2* and *interviewee3* appear together, and also *interviewee2* is to the left of *interviewee3*.

Query 2: Show the fragments of the clip where *reporter1* and *reporter2* appear together with *priest* who is in his *car*.

Query 3: Show the fragments of the clip where *man3* is to the west of *man4* who is to the east of *woman2*.

Query 4: Show the longest possible fragments of the clip where *man6* is first to the left of *man5*, and later he becomes to the right of *man5*.

The first query is a directional-appearance query on salient objects *priest*, *interviewee2* and *interviewee3*. Query 2 is a topological-appearance query and Query 3 is a directional query. The last query, Query 4, is a motion query based on directional relations between salient objects *man5* and *man6*.

The second query assumes that if a person is inside a car or covered-by a car, then he/she is in that car. This assumption may not be correct depending on the

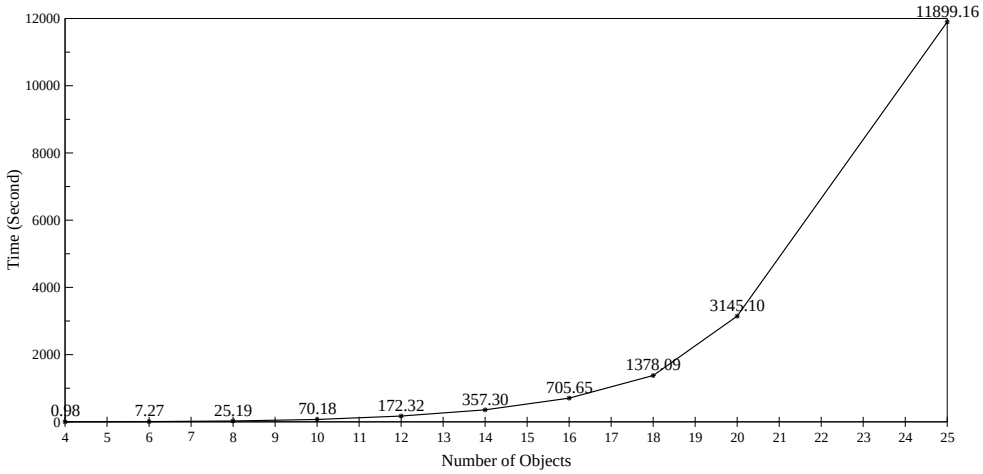


Figure 9.4: Query 1: west(X, Y, F) ∧ disjoint(X, Y, F) (100 Frames)

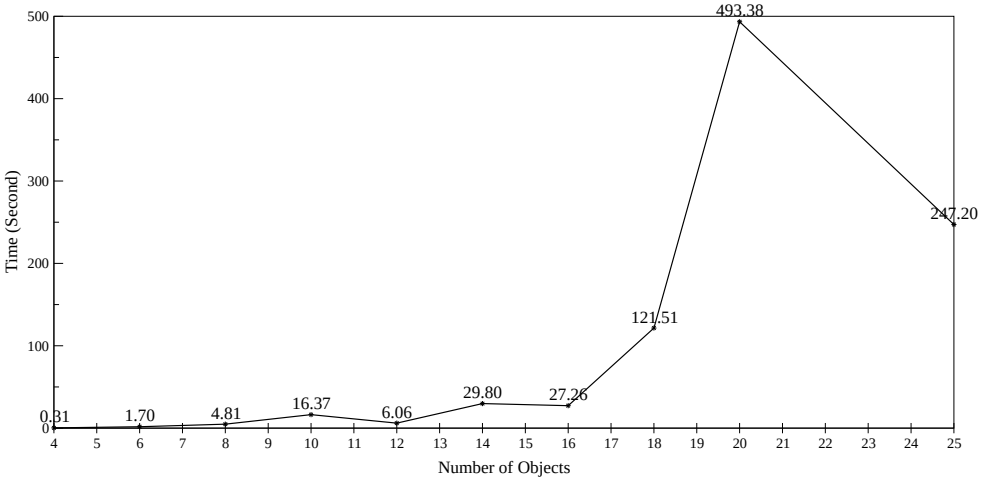


Figure 9.5: Query 2: west(1, Y, F) ∧ disjoint(1, Y, F) (100 Frames)

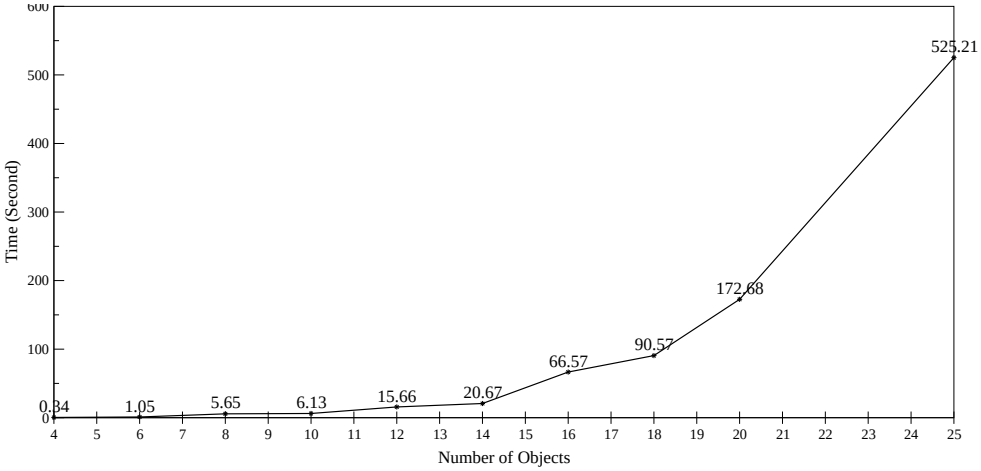


Figure 9.6: Query 3: west(X, 7, F) ∧ disjoint(X, 7, F) (100 Frames)

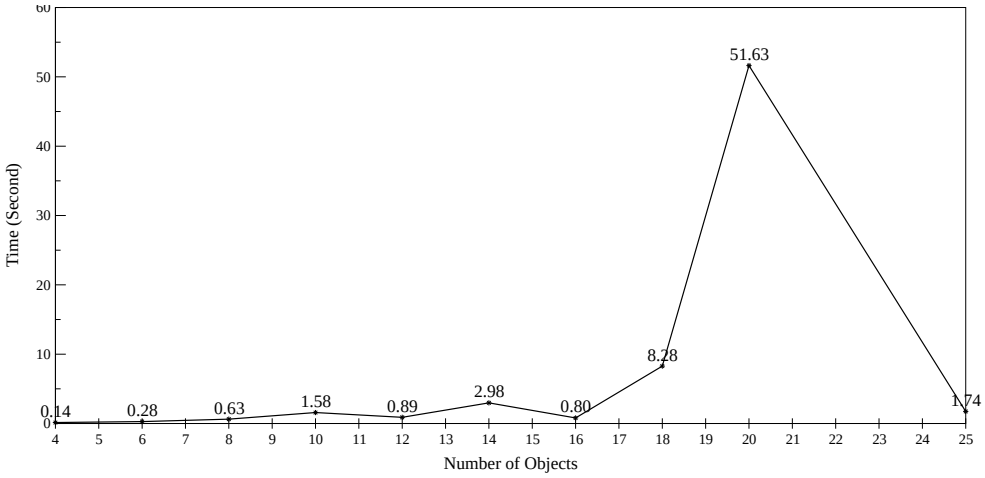


Figure 9.7: Query 4: west(1, 7, F) ∧ disjoint(1, 7, F) (100 Frames)

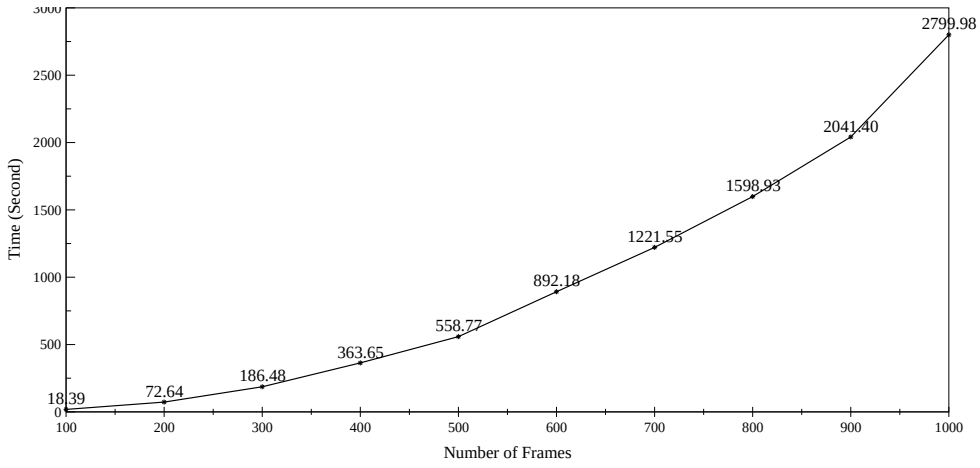


Figure 9.8: Query 5: west(X, Y, F) ∧ disjoint(X, Y, F) (8 Objects)

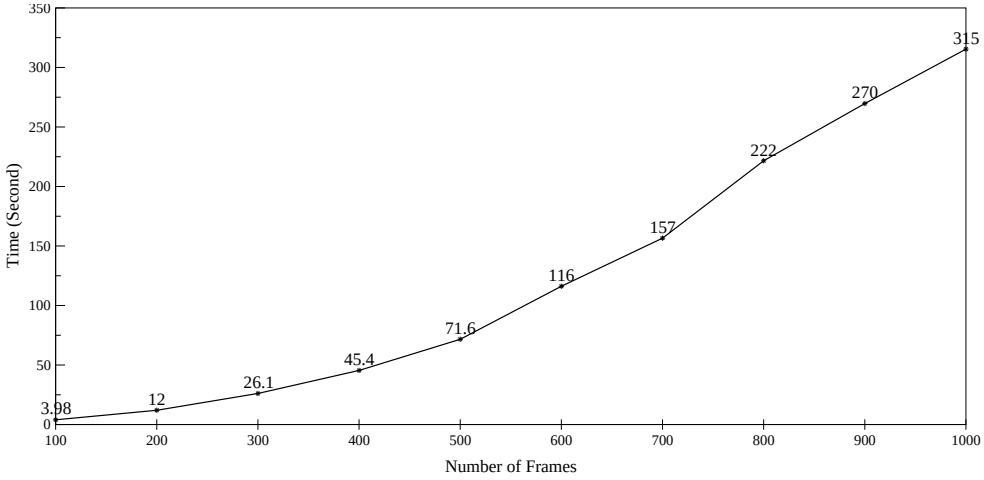


Figure 9.9: Query 6: west(1, Y, F) ∧ disjoint(1, Y, F) (8 Objects)

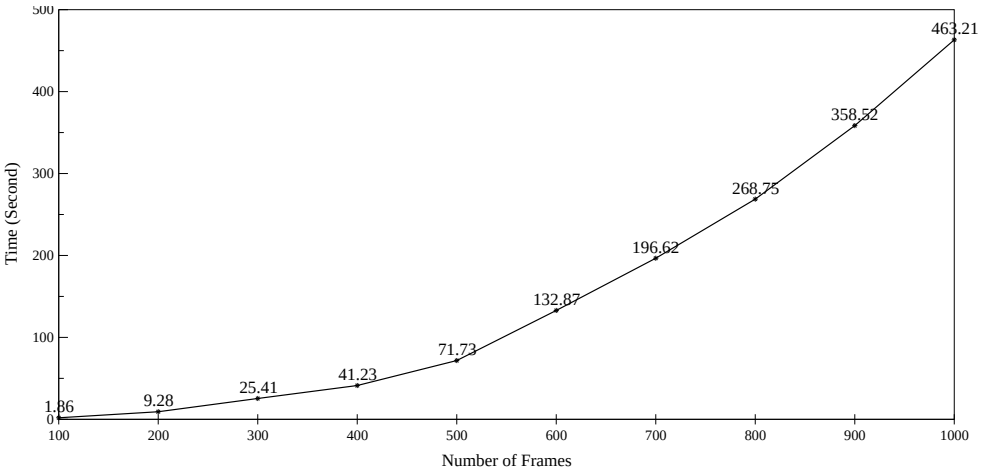


Figure 9.10: Query 7: west(X, 0, F) ∧ disjoint(X, 0, F) (8 Objects)

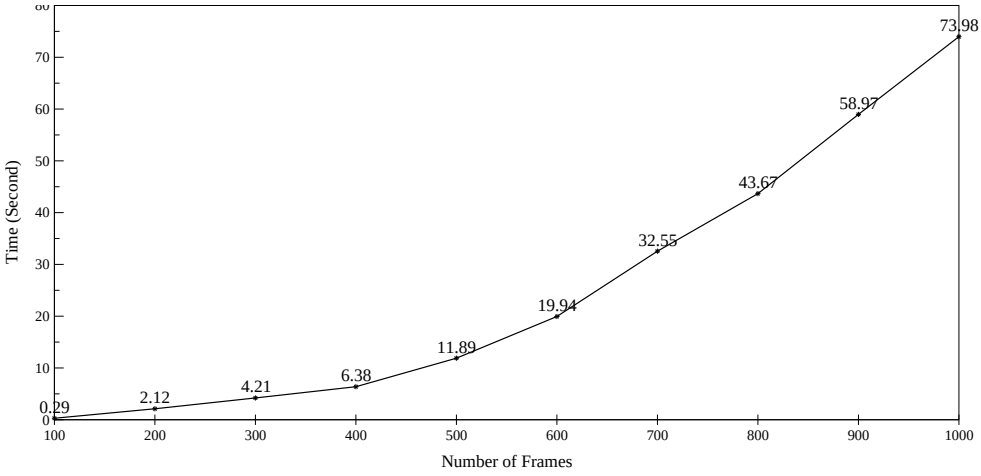
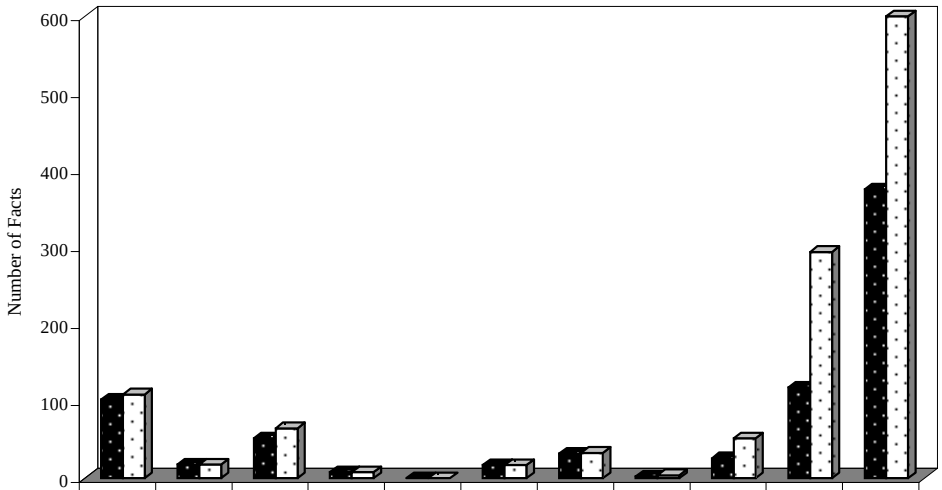
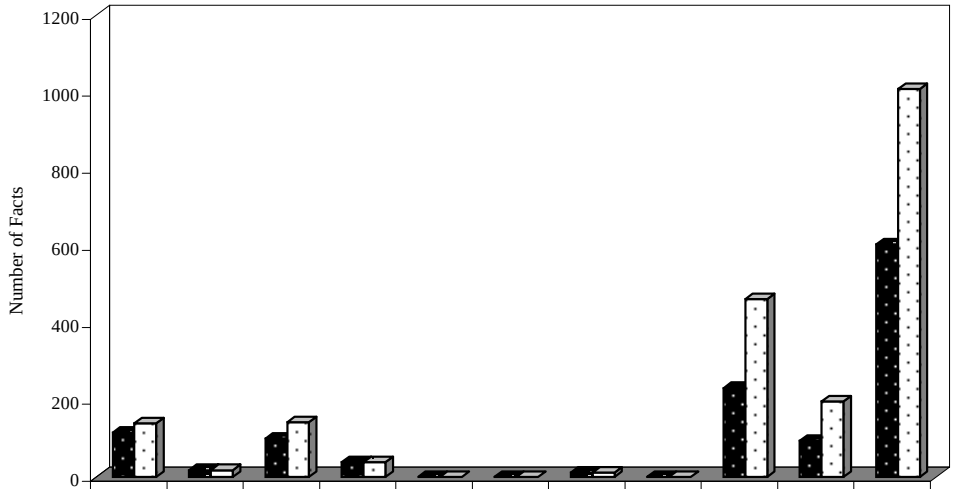


Figure 9.11: Query 8: west(1, 0, F) ∧ disjoint(1, 0, F) (8 Objects)



	West	South	North-west	South-west	Equal	Cover	Inside	Touch	Disjoint	Overlap	All
Facts-base 1	102	18	52	8	0	17	32	2	26	118	375
Facts-base 2	109	18	65	8	0	17	33	4	52	294	600

Figure 9.12: Space Efficiency Test Results for jornal.mpg



	West	South	North-west	South-west	Equal	Cover	Inside	Touch	Disjoint	Overlap	All
Facts-base 1	115	18	100	39	0	0	12	0	231	94	604
Facts-base 2	139	18	142	39	0	0	12	0	462	196	1008

Figure 9.13: Space Efficiency Test Results for smurfs.avi

camera view, but yet, it could be handled easily using the 3D relation *samelevel*. Nonetheless, since our tests are based on 2D spatial relations, no 3D relation is considered even though the system has a set of 3D inference rules as well. The results obtained for the queries are given in Table 9.3. The queries posed on the *Smurfs* video fragment are as follows:

Query 1: Give the parts of the video clip where *bigmouth* is below *robotsmurf* while *robotsmurf* starts moving from *bigmouth*'s left to his right and then goes from his right to his left repeating this as many times as it happens in the video fragment.

Query 2: Give the parts of the video clip where *Gargamel* is to the southwest of his *father* and *boyking*, who is between *soldier1* and *soldier2* (to his left and his right) and is in some distance with *Gargamel* and his *father*.

Query 3: Give the parts of the video clip where *lazysmurf*, *farmersmurf*, *grouchysmurf*, *smurfette* and *handysmurf* all appear together such that *lazysmurf* is to the west of *handysmurf* and *smurfette* is to the east of *farmersmurf*.

Query 4: Give the parts of the video clip where *robotsmurf* and *bigmouth* are close to each other (not disjoint) and *robotsmurf* is to the right of *bigmouth*, and there is no other object of interest that appears.

Query 1 is a directional-motion query while Query 2 is a directional-topological query. The third query is a directional-appearance query and the last one is a directional-topological-appearance query. In Query 1, we are interested in finding the largest sequences of frames in the fragment repeating the motion condition stated as many times as possible sequentially in the video. Thus, the final answer returned by the system is the set of video frame intervals where this motion is repeated as many times as possible sequentially in the video. For Query 2, it is concluded that two objects are in some distance and not close to each other if their MBRs are disjoint. If the objects are close to each other, then it is decided that their MBRs are not disjoint as in Query 4. These assumptions

Query #	Reduced Set (Sec.)
1	0.04
2	0.03
3	0.01
4	0.02

Table 9.3: Time Efficiency Test Results for jornal.mpg

Query #	Reduced Set (Sec.)
1	0.13
2	0.03
3	0.01
4	0.03

Table 9.4: Time Efficiency Test Results for smurfs.avi

are only our semantic definitions of being two objects close to each other or in a distance. Therefore, these queries are partially based on these semantic definitions. Table 9.4 shows the results obtained for the queries.

In the tests conducted with program-generated video data, there is a 19.59% savings from the space for the sample data of 8 objects and 1000 frames. The space savings for the sample video of 15 objects and 1000 frames is 31.47% while it is 40.42% for 25 objects and 1000 frames. With real data, for the first video fragment *jornal.mpg*, our rule-based approach provides a savings of 37.5% from the space. The space savings for the other fragment, *smurfs.avi*, is 40%.

The space savings obtained from the program-generated video data is relatively low compared to that obtained from the real video fragments. We believe that the reason behind such a behavior is due to the random simulation of the motion of objects in our synthetic test data: while creating the synthetic video data, the motion pattern of objects was simulated randomly changing objects' MBR coordinates by choosing only one object to move at each frame. However, in real video, objects generally move slower causing the set of spatial relations to change over a longer period of frames. During the tests with the synthetic video data, it is also observed that space savings do not change when the number of frames is increased while the number of objects of interest per frame is fixed. The test results obtained for the synthetic data comply with those obtained for the

real video. Some differences seen in the results are due to the fact that synthetic data was produced by a program, therefore not being able to perfectly simulate a real-life scenario.

The results plotted in Figures 9.4-9.11 show that *BilVideo* is scalable for spatio-temporal queries in terms of the number of objects and the number of frames when either of these numbers is increased while the other is fixed. The time value deviations in some graphs are due to the data sets that had to be created separately for each object set, thereby each set having possibly different facts. Furthermore, the results obtained from the time efficiency tests on real video data show that *BilVideo* has a reasonable response time for spatio-temporal queries.

Chapter 10

Application Areas

BilVideo has been designed to be a full-fledged Web-based video database management system that supports spatio-temporal, semantic and low-level (color, shape and texture) queries on video data. There are only a few video database prototypes around developed for either academic or commercial purposes; nonetheless, they do only provide support for a rather small subset of the video features in comparison to *BilVideo*, and hence, their success in returning what the user has actually in mind for his/her query is very limited. Moreover, their support for visual query specification is also not as powerful as that of *BilVideo*, which is very important because the success rate of a video database system also depends on how it acquires the query parameters from users. The visual query interface should be simple and easy-to-use, yet sophisticated and powerful enough to make use of all the capabilities of the underlying system.

BilVideo does not target a specific application area, and thus, it can be used to support any application, where vast amount of video data needs to be searched by spatio-temporal, semantic and low-level video features. Furthermore, *BilVideo* query language provides a simple way to extend the system's query capabilities through *external predicates*, which makes *BilVideo* application-independent but yet easily fine-tunable for specific needs of such applications without much effort and without any loss in performance at all. This can be achieved by adding to the knowledge-base some application-dependent rules and/or facts that will

be used for queries. Some example applications that might be supported are sports event analysis systems (soccer, basketball, etc.), object movement tracking systems (medical, biological, astrophysical, etc.) and video archive search systems (movie retrieval, digital libraries, news retrieval, etc.). Specifically, some emerging applications in such areas as digital culture, tourism, entertainment, education and e-commerce may greatly benefit from *BilVideo* using it as their underlying video database management system.

10.1 An Example Application: News Archives Search System

In this section, we present an application, news archives search system, for *BilVideo*. A news archives search system contains video clips of news broadcasts and is used to retrieve specific news fragments based on some descriptions given as query conditions. The traditional approach for accomplishing this task is to provide some keywords that would describe semantic content of the news fragments for retrieval. For this, a traditional database system would suffice since news fragments are indexed by some textual data. Nevertheless, spatio-temporal relations between objects and object trajectories are not considered. Moreover, traditional database systems also lack of support for color, shape and texture video queries. Furthermore, the traditional approach might result in retrievals of some news fragments that are irrelevant to what the user wants to see while also missing some others that are actually expected by the user. It is because keyword-based search is not powerful enough to formulate what the user has in his/her mind as a query. Therefore, some other search mechanisms are also needed. In this regard, BilVideo fills up this gap by providing support for spatio-temporal, semantic, color, shape and texture video queries. Users may also query news archives by some specific application-dependent predicates supported by the query language of BilVideo to retrieve precise answers to queries.

A fragment video clip captured from news broadcast by a national Turkish TV channel was chosen as a basis for the spatio-temporal query examples given in this

section. Facts representing the spatio-temporal relations between objects, object-appearance relations and object trajectories were extracted and inserted into the knowledge-base prior to submitting the queries to the system. The following is an example set of such relations extracted from a keyframe of this news fragment:

```
south(policevehicle, israeliflag)
overlap(policevehicle, israeliflag)
appear(israeliflag)
appear(policevehicle)
samelevel(israeliflag, policevehicle)
```

Query 1: "Retrieve the segments from the sample news clip, where Arafat and Powell appear together alone (no other object of interest is in the scene) and Powell is to the right of Arafat."

```
select segment from vid
where appear_alone(arafat, powell) and right(powell, arafat);
```

In this query, *appear_alone* is an external (application-dependent) predicate. It is used to search for video keyframes, where the only objects appearing are those specified. The predicate *right* is a directional predicate. *Vid* is a unique video identifier assigned to the sample news video clip.

Query 2: "Retrieve the segments from the sample news clip, where Turkish Prime Minister Ecevit and Turkish Foreign Affairs Minister Cem appear together close to each other and Ecevit is to the right of and in front of Cem."

```
select segment from vid
where right(ecevit, cem) and infrontof(ecevit, cem) and
  close(ecevit, cem);
```

In this query, *close* is an external predicate. It is used to search for video keyframes, where the objects specified are very close to each other. Here, the closeness is defined semantically as follows: If two objects are close, then their MBRs are not disjoint. This definition is given for this application and

may change for others. The system can easily adapt to such changes through external predicates defined in the knowledge-base according to applications' specific needs. The predicate *infrontof* is a third-dimension (3D) predicate.

Query 3: "Retrieve the segments from the sample news clip, where a police vehicle moves toward west together with an Israeli flag that is above the vehicle and overlaps with it, given a similarity threshold value of 0.8 and an allowed time gap value of 1 second."

```
select segment from vid
where (tr(policevehicle, [[west]]) sthreshold 0.8 tgap 1)
  repeat and overlap(israeliflag, policevehicle) and
  above(israeliflag, policevehicle);
```

In this query, a similarity-based trajectory condition is given, along with directional and topological conditions. The interval operator *and* implies that all conditions are satisfied in the intervals returned to the user as segments and that for all video frames in such segments, the flag is above the police vehicle and also it overlaps with the vehicle. The keywords `tgap` (time gap) and `repeat` are used for the trajectory condition to ensure that all segments in the clip that satisfy the given conditions, where the police vehicle may stop for at most 1 second at a time during its movement toward west, are returned as an answer to the query.

Chapter 11

Conclusions and Future Work

BilVideo system architecture has been designed to support spatio-temporal (directional, topological, 3D-relation, object-appearance, external-predicate, trajectory-projection and similarity-based object-trajectory conditions), semantic (keyword, event/activity and category-based conditions) and low-level (color, shape and texture conditions) queries on video data in an integrated manner. *BilVideo* responds to spatio-temporal queries using its knowledge-base, which consists of a fact-base and a comprehensive set of rules implemented in Prolog, while semantic and low-level queries are handled by an object-relational database. The query processor of *BilVideo* interacts with both of the knowledge-base and object-relational database to respond to user queries and the intermediate query results returned from these two system components are integrated seamlessly by the query processor to be sent to Web clients.

The query language of *BilVideo* currently supports a broad range of spatio-temporal video queries. In order to provide support for color and shape queries, we propose a new approach to store and compare color and shape features of the salient objects in video keyframes [45]. In our approach, three histograms, *distance*, *angle* and *color*, are used to store shape and color contents of the salient objects. Shape information is gathered from the interior and boundary pixels of an object and the mass center of the object plays an important role in this process. This technique resembles the visualization of an human eye and resolves the drawbacks of the existing methods because only are the boundaries of objects

considered in most of the other systems. Moreover, since the method gathers information from the pixels, the length and shape of an object's boundary as well as the presence of holes in the interior of an object have no effect on the method. Besides, there is no polynomial restriction on object boundaries. Hence, the method is very successful in processing even noisy objects.

Our work on semantic and low-level (color, shape and texture) modeling and querying of video data is ongoing. For semantic video queries, extensions to the *BilVideo* textual query language have been defined, but not incorporated fully into its parser yet. We will also make extensions to the language grammar for low-level video queries. Furthermore, semantic and low-level query processing algorithms are currently being implemented and will be incorporated into the *BilVideo* query processor. Another important issue we are studying is the optimization of user queries [51]. *BilVideo* query processor has currently been optimized for spatio-temporal video queries, but we are still looking for ways to improve it more.

In an ideal environment, *BilVideo* textual query language will establish the basis for a visual query interface and serve as an embedded language for users because some video queries are much easier to specify visually. Hence, we will also enhance query specification capabilities of the Web-based user interface of *BilVideo* in compliance with the features supported by its textual video query language.

We are also in close watch of the MPEG-7 work, whose goal is to provide a rich set of standardized tools to describe the multimedia content [12]. Structure-based Description Schemes in MPEG-7 describe the audio-visual (AV) content from the viewpoint of its structure. They are organized around a Segment Description Scheme that represents the spatial, temporal or spatio-temporal structure of the AV content. The Segment Description Scheme can be organized into a hierarchical structure to produce a Table of Content for accessing or Index for searching the AV content. Segments can be further described on the basis of perceptual features using MPEG-7 Descriptors for color, texture, shape, motion and audio features, and semantic information using textual annotations. We will see in future if we could make use of the outcomes of the MPEG-7 work in *BilVideo*.

Bibliography

[1] S. Adalı, K.S. Candan, S. Chen, K. Erol, and V.S. Subrahmanian. Advanced video information systems: Data structures and query processing. *ACM Multimedia Systems*, 4:172–186, 1996.

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

[3] U. Arslan. A semantic data model and query language for video databases. M.S. thesis, Department of Computer Engineering, Bilkent University, Ankara, Turkey, January 2002.

[4] N. Babaguchi, Y. Kawai, and T. Kitahashi. Event based indexing of broadcasted sports video by intermodel collaboration. *IEEE Transaction on Multimedia*, 4(1):68–75, March 2002.

[5] S. Chang, W. Chen, H.J. Meng, H. Sundaram, and D. Zhong. VideoQ: An automated content-based video search system using visual cues. In *Proc. of ACM Multimedia*, pages 313–324, Seattle, Washington, USA, 1997.

[6] E. Şaykol. Web-based user interface for query specification in a video database system. M.S. thesis, Department of Computer Engineering, Bilkent University, Ankara, Turkey, September 2001.

[7] M.E. Dönderler, E. Şaykol, U. Arslan, Ö. Ulusoy, and U. Güdükbay. BilVideo: A video database management system. *submitted journal paper*.

[8] M.E. Dönderler, E. Şaykol, Ö. Ulusoy, and U. Güdükbay. BilVideo: A video database management system. *accepted for publication in IEEE Multimedia (Multimedia at Work)*, 2002.

[9] M.E. Dönderler, Ö. Ulusoy, and U. Güdükbay. Rule-based spatio-temporal query processing for video databases. *submitted journal paper*.

[10] M.E. Dönderler, Ö. Ulusoy, and U. Güdükbay. A rule-based video database system architecture. *Information Sciences*, 143(1-4):13–45, June 2002.

[11] C. Donnelly and R. Stallman. Bison: The yacc-compatible parser generator. Online manual, `http://www.combo.org/bison/`, 1995.

[12] J. M. Martinez (Editor). Overview of the MPEG-7 standard (Version 6.0). Technical Report ISO/IEC JTC1/SC29/WG11 N4509, `http://mpeg.telecomitalialab.com/standards/mpeg-7/mpeg-7.htm`, December 2001.

[13] M. Egenhofer and R. Franzosa. Point-set spatial relations. *Int'l Journal of Geographical Information Systems*, 5(2):161–174, 1991.

[14] C. Faloutsos, , W. Equitz, M. Flickner, W. Niblack, D. Petkovic, and R. Barber. Efficient and effective querying by image content. *Special Issue on Integrating Artificial Intelligence and Database Technology, Journal of Intelligent Information Systems*, 3(3/4):231–262, July 1994.

[15] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani abd J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: The qbic system. *IEEE Computer*, 28(9):23–32, September 1995.

[16] S. Gauch, J. Gauch, and K. M. Pua. The VISION digital video library project. *to appear in the Encyclopedia of Library and Information Science.*

[17] A.G. Hauptmann and M.J. Witbrock. *Intelligent Multimedia Information Retrieval, M. T. Maybury (ed.)*, chapter Informedia: News-on-demand Multimedia Information Acquisition and Retrieval, pages 215–239. MIT Press, 1997.

[18] R. Hjelsvold and R. Midtstraum. Modelling and querying video data. In *Proc. of the 20th Int. Conference on VLDB*, pages 686–694, Santiago, Chile, 1994.

[19] E. Hwang and V.S. Subrahmanian. Querying video libraries. *Journal of Visual Communication and Image Representation*, 7(1):44–60, 1996.

[20] S.C. Johnson. Yacc: Yet another compiler compiler. Computing Science Technical Report 32, Bell Laboratories, Murray Hill, NJ, 1975.

[21] T.C.T. Kuo and A.L.P. Chen. A content-based query language for video databases. In *Proc. of IEEE Multimedia Computing and Systems*, pages 209–214, 1996.

[22] T.C.T. Kuo and A.L.P. Chen. Content-based query processing for video databases. *IEEE Transactions on Multimedia*, 2(1):1–13, 2000.

[23] M.E. Lesk. Lex - a lexical analyzer generator. Computing Science Technical Report 39, Bell Laboratories, Murray Hill, NJ.

[24] J.L. Li, I.A. Goralwalla, M.T. Özsu, and D. Szafron. Modeling video temporal relationships in an object database management system. In *International Symposium on Electronic Images: Multimedia Computing and Networking*, 1997.

[25] John Z. Li, M. Tamer Özsu, and Duane 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.

[26] J.Z. Li. Modeling and querying multimedia data. Technical Report TR-98-05, Department of Computing Science, The University of Alberta, Alberta, Canada, 1998.

[27] J.Z. Li and M.T. Özsu. Stars: A spatial attributes retrieval system for images and videos. In *Proc. of the 4th Int. Conf. on Multimedia Modeling*, pages 69–84, Singapore, 1997.

[28] J.Z. Li, M.T. Özsu, and D. Szafron. Spatial reasoning rules in multimedia management systems. In *Proc. of Int. Conf. on Multimedia Modeling*, pages 119–133, Toulouse, France, 1996.

[29] J.Z. Li, M.T. Özsu, D. Szafron, and V. Oria. MOQL: A multimedia object query language. In *Proc. of the 3rd Int. Workshop on Multimedia Information Systems*, pages 19–28, Como, Italy, 1997.

[30] J.Z. Li, M.T. Özsu, D. Szafron, and V. Oria. Multimedia extensions to database query languages. Technical Report TR-97-01, Department of Computing Science, The University of Alberta, Alberta, Canada, 1997.

[31] Y. Liu and F.Li. Semantic extraction and semantics-based annotation and retrieval for video databases. *Multimedia Tools and Applications*, 17:5–20, 2002.

[32] Y. Manolopoulos and G. Kapetanakis. Overlapping B+ trees for temporal data. In *Proceedings of the 5th Jerusalem Conference on Information Technology (JCIT)*, pages 491–498, 1990.

[33] S. Marcus and V.S. Subrahmanian. Foundations of multimedia information systems. *Journal of ACM*, 43(3):474–523, 1996.

[34] S. Markus and V.S. Subrahmanian. *Multimedia Database Systems: Issues and Research Directions (eds. V.S. Subrahmanian and S. Jajodia)*, chapter Towards a Theory of Multimedia Database Systems, pages 1–35. Springer-Verlag, 1996.

[35] M.S.Hacid, C. Decleir, and J. Kouloumdjian. A database approach for modeling and querying video data. *IEEE Transaction on Knowledge and Data Engineering*, 12(5):729–750, September/October 2000.

[36] M. Nabil, A.H. Ngu, and J.Shepherd. Modeling and retrieval of moving objects. *Multimedia Tools and Applications*, 13:35–71, 2001.

[37] M.A. Nascimento and J.R.O. Silva. Towards historical R-trees. In *Proceedings of ACM Symposium on Applied Computing (ACM-SAC)*, pages 235–240, 1998.

[38] E. Oomoto and K. Tanaka. OVID: Design and implementation of a video object database system. *IEEE Trans. on Knowledge and Data Engineering*, 5:629–643, 1993.

[39] Özden, R. Rastogi, and A. Silberschatz. Multimedia support for databases. In *PODS*, pages 1–11, 1997.

[40] M.T. Özsu, P. Iglinski, D. Szafron, S. El-Medani, and M. Junghanns. An object-oriented sqml/hytime compliant multimedia database management system. In *Proc. of ACM Multimedia*, pages 233–240, Seattle, WA, 1997.

[41] D. Papadias, Y. Theodoridis, T. Sellis, and M. Egenhofer. Topological relations in the world of minimum bounding rectangles: A study with R-trees. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 92–103, San Jose, CA, USA, 1996.

[42] V. Paxson. Flex: A fast scanner generator. Online manual, `http://www.combo.org/flex/`, 1995.

[43] S. Satoh, Y. Nakamura, and T. Kanade. Name-it: Naming and detecting faces in news videos. *IEEE Multimedia*, 6(1):22–35, January-March 1999.

[44] E. Şaykol, U. Güdükbay, and Ö. Ulusoy. A semi-automatic object extraction tool for querying in multimedia databases. In S. Adali and S. Tripathi, editors, *7th Workshop on Multimedia Information Systems MIS'01, Capri, Italy*, pages 11–20, November 2001.

[45] E. Şaykol, U. Güdükbay, and Ö. Ulusoy. A histogram-based approach for object-based query-by-shape-and-color in multimedia databases. *submitted journal paper and also Tech. Rep. BU-CE-0201, Bilkent University*, January 2002.

[46] A.P. Sistla and C. Yu. Similarity based retrieval of pictures using indices on spatial relationships. In *Proc. of the 21st VLDB Conference*, pages 619–629, Zurich, Switzerland, September 1995.

[47] A.P. Sistla and C. Yu. Reasoning about qualitative spatial relationships. *Journal of Automated Reasoning*, 25(4):291–328, November 2000.

[48] Y. Theodoridis, J.R.O. Silva, and M.A. Nascimento. On the generation of spatio-temporal datasets. In *Proceedings of the 6th Int'l Symposium on*

Large Spatial Databases (SSD), LNCS Series, Hong Kong, China, July 1999. Springer-Verlag.

[49] Y. Theodoridis, M. Vazirgiannis, and T. Sellis. Spatio-temporal indexing for large multimedia applications. In *Proceedings of the 3rd IEEE Conference on Multimedia Computing and Systems (ICMCS)*, 1996.

[50] T. Tzouramanis, M. Vassilakopoulos, and Y. Manolopoulos. Overlapping linear quadtrees: A spatio-temporal access method. In *Proceedings of the 6th Int'l ACM Workshop on Geographical Information Systems (ACM-GIS)*, pages 1–7, 1998.

[51] G. Ünel. An efficient query optimization strategy for spatio-temporal queries in video databases. M.S. thesis, Department of Computer Engineering, Bilkent University, Ankara, Turkey, July 2002.

[52] R.C. Veltkamp and M. Tanase. Content-based retrival system: A survey. Technical Report UU-CS-2000-34, Utrect University, The Netherlands, October 2000.

[53] X. Xu, J. Han, and W. Lu. RT-tree: An improved R-tree index structure for spatio-temporal databases. In *Proceedings of the 4th International Symposium on Spatial Data Handling (SDH)*, pages 1040–1049, 1990.

[54] Y. Yesha and M. Singhal. Multimedia database systems: Challenges and opportunities. Technical Report OSU-CISRC-1/96-TR07, Department of Computer and Information Science, Ohio State University, 1996.

[55] A. Yoshitaka and T. Ichikawa. A survey of content-based retrieval for multimedia databases. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):81–93, 1999.

Appendix A

List of Inference Rules

In this appendix, we give our set of *strict directional*, *strict topological*, *heterogeneous directional and topological* and *third-dimension* rules. In defining the rules, the following terminology has been adopted: if the relation r_1 implies the relation r_2, $r_1 \Longrightarrow r_2$ is used. Moreover, if $r_1 \Longrightarrow r_2$ and $r_2 \Longrightarrow r_1$, it is denoted by $r_1 \Longleftrightarrow r_2$. In addition, there is also a rule set for *appear*, which is used to derive trivial facts, *equal*(A,A), *overlap*(A,A) and *samelevel*(A,A) , as well as to answer object-appearance queries in video clips. This set is given below:

appear(A) $\Longleftrightarrow$ *equal*(A,A)
appear(A) $\Longleftrightarrow$ *overlap*(A,A)
appear(A) $\Longleftrightarrow$ *samelevel*(A,A)

A.1 Strict Directional Rules

Rule Set 1.1 (Inverse Property) The relations *west*, *north*, *north-west*, *north-east*, *right* and *above* are inverses of *east*, *south*, *south-east*, *south-west*, *left* and *below*, respectively.

a) *west*(A,B) $\Longleftrightarrow$ *east*(B,A)
b) *north*(A,B) $\Longleftrightarrow$ *south*(B,A)
c) *north-west*(A,B) $\Longleftrightarrow$ *south-east*(B,A)
d) *north-east*(A,B) $\Longleftrightarrow$ *south-west*(B,A)
e) *right*(A,B) $\Longleftrightarrow$ *left*(B,A)
f) *above*(A,B) $\Longleftrightarrow$ *below*(B,A)

Rule Set 1.2 (Transitivity) If $\beta \in$ S, where S is the set of directional relations, then

β(A,B) $\wedge$ β(B,C) $\Longrightarrow$ β(A,C).

Rule Set 1.3 The relations *right*, *left*, *above* and *below* can be expressed by other directional relations.

a) *east*(A,B) $\vee$ *north-east*(A,B) $\vee$ *south-east*(A,B) $\Longleftrightarrow$ *right*(A,B)
b) *west*(A,B) $\vee$ *north-west*(A,B) $\vee$ *south-west*(A,B) $\Longleftrightarrow$ *left*(A,B)
c) *north*(A,B) $\vee$ *north-east*(A,B) $\vee$ *north-west*(A,B) $\Longleftrightarrow$ *above*(A,B)
d) *south*(A,B) $\vee$ *south-east*(A,B) $\vee$ *south-west*(A,B) $\Longleftrightarrow$ *below*(A,B)

A.2 Strict Topological Rules

Rule Set 2.1 (Inverse Property) The relations *inside* and *cover* are inverses of *contains* and *covered-by*, respectively.

a) *inside*(A,B) $\Longleftrightarrow$ *contains*(B,A)
b) *cover*(A,B) $\Longleftrightarrow$ *covered-by*(B,A)

Rule Set 2.2 (Reflexivity) The relations *equal* and *overlap* are reflexive.

a) *equal*(A,A)
b) *overlap*(A,A)

Rule Set 2.3 (Symmetry) The relations *equal*, *overlap*, *disjoint* and *touch* are symmetric.

a) *equal*(A,B) $\Longleftrightarrow$ *equal*(B,A)
b) *overlap*(A,B) $\Longleftrightarrow$ *overlap*(B,A)
c) *disjoint*(A,B) $\Longleftrightarrow$ *disjoint*(B,A)
d) *touch*(A,B) $\Longleftrightarrow$ *touch*(B,A)

Rule Set 2.4 (Transitivity) The relations *inside* and *equal* are transitive.

a) *inside*(A,B) $\wedge$ *inside*(B,C) $\Longrightarrow$ *inside*(A,C)
b) *equal*(A,B) $\wedge$ *equal*(B,C) $\Longrightarrow$ *equal*(A,C)

Rule Set 2.5 The relations *inside*, *equal* and *cover* imply the relation *overlap*.

a) *inside*(A,B) $\Longrightarrow$ *overlap*(A,B)
b) *equal*(A,B) $\Longrightarrow$ *overlap*(A,B)
c) *cover*(A,B) $\Longrightarrow$ *overlap*(A,B)

Rule Set 2.6 The relationships between *equal* and {*cover*, *inside*, *disjoint*, *touch*, *overlap*} are as follows:

a) *equal*(A,B) $\wedge$ *cover*(B,C) $\Longrightarrow$ *cover*(A,C)
b) *equal*(A,B) $\wedge$ *cover*(C,B) $\Longrightarrow$ *cover*(C,A)
c) *cover*(A,B) $\wedge$ *equal*(A,C) $\Longrightarrow$ *cover*(C,B)
d) *cover*(A,B) $\wedge$ *equal*(B,C) $\Longrightarrow$ *cover*(A,C)
e) *equal*(A,B) $\wedge$ *inside*(B,C) $\Longrightarrow$ *inside*(A,C)
f) *equal*(A,B) $\wedge$ *inside*(C,B) $\Longrightarrow$ *inside*(C,A)
g) *inside*(A,B) $\wedge$ *equal*(A,C) $\Longrightarrow$ *inside*(C,B)
h) *inside*(A,B) $\wedge$ *equal*(B,C) $\Longrightarrow$ *inside*(A,C)
i) *equal*(A,B) $\wedge$ *disjoint*(B,C) $\Longrightarrow$ *disjoint*(A,C)
j) *disjoint*(A,B) $\wedge$ *equal*(B,C) $\Longrightarrow$ *disjoint*(A,C)
k) *equal*(A,B) $\wedge$ *overlap*(B,C) $\Longrightarrow$ *overlap*(A,C)
l) *overlap*(A,B) $\wedge$ *equal*(B,C) $\Longrightarrow$ *overlap*(A,C)
m) *equal*(A,B) $\wedge$ *touch*(B,C) $\Longrightarrow$ *touch*(A,C)
n) *touch*(A,B) $\wedge$ *equal*(B,C) $\Longrightarrow$ *touch*(A,C)

Rule Set 2.7 The relationships between *disjoint* and {*inside*, *touch*, *cover*} are as follows:

a) *inside*(A,B) $\wedge$ *disjoint*(B,C) $\Longrightarrow$ *disjoint*(A,C)
b) *disjoint*(A,B) $\wedge$ *inside*(C,B) $\Longrightarrow$ *disjoint*(A,C)
c) *inside*(A,B) $\wedge$ *touch*(B,C) $\Longrightarrow$ *disjoint*(A,C)
d) *touch*(A,B) $\wedge$ *inside*(C,B) $\Longrightarrow$ *disjoint*(A,C)
e) *cover*(A,B) $\wedge$ *disjoint*(C,A) $\Longrightarrow$ *disjoint*(C,B)
f) *disjoint*(A,B) $\wedge$ *cover*(A,C) $\Longrightarrow$ *disjoint*(C,B)
g) *inside*(A,B) $\wedge$ *touch*(B,C) $\wedge$ *cover*(C,D) $\Longrightarrow$ *disjoint*(A,D)

Rule Set 2.8 The relationships between *overlap* and {*inside*, *cover*} are as follows (excluding those given by Rule Set 2.5):

a) *inside*(A,B) $\wedge$ *overlap*(C,A) $\Longrightarrow$ *overlap*(B,C)
b) *overlap*(A,B) $\wedge$ *inside*(B,C) $\Longrightarrow$ *overlap*(A,C)
c) *cover*(A,B) $\wedge$ *overlap*(B,C) $\Longrightarrow$ *overlap*(A,C)
d) *overlap*(A,B) $\wedge$ *cover*(C,B) $\Longrightarrow$ *overlap*(A,C)
e) *cover*(A,B) $\wedge$ *inside*(C,B) $\Longrightarrow$ *overlap*(A,C)

Rule Set 2.9 The relationships between *inside* and *cover* are as follows:

a) *inside*(A,B) $\wedge$ *cover*(C,B) $\Longrightarrow$ *inside*(A,C)
b) *inside*(A,C) $\wedge$ *cover*(A,B) $\Longrightarrow$ *inside*(B,C)
c) *cover*(A,B) $\wedge$ *cover*(B,C) $\wedge$ not(*cover*(A,C)) $\Longrightarrow$ *inside*(C,A)

A.3 Heterogeneous Directional and Topological Rules

Rule Set 3.1 If $\beta \in$ S, where S is the set of directional relations, then

a) *equal*(A,B) $\wedge$ β(B,C) $\Longrightarrow$ β(A,C)
b) β(A,B) $\wedge$ *equal*(B,C) $\Longrightarrow$ β(A,C)

A.4 Third-Dimension Rules

Rule Set 4.1 (Reflexivity) The relation *samelevel* is reflexive.

samelevel(A,A)

Rule Set 4.2 (Symmetry) The relation *samelevel* is symmetric.

samelevel(A,B) $\Longrightarrow$ *samelevel*(B,A)

Rule Set 4.3 (Inverse Property) The relations *infrontof*, *strictlyinfrontof* and *touchfrombehind* are inverses of *behind*, *strictlybehind* and *touchedfrombehind*, respectively.

a) *infrontof*(A,B) $\Longleftrightarrow$ *behind*(B,A)
b) *strictlyinfrontof*(A,B) $\Longleftrightarrow$ *strictlybehind*(B,A)
c) *touchfrombehind*(A,B) $\Longleftrightarrow$ *touchedfrombehind*(B,A)

Rule Set 4.4 (Transitivity) The relations *infrontof* and *strictlyinfrontof* are transitive.

a) *infrontof*(A,B) $\wedge$ *infrontof*(B,C) $\Longrightarrow$ *infrontof*(A,C)
b) *strictlyinfrontof*(A,B) $\wedge$ *strictlyinfrontof*(B,C) $\Longrightarrow$ *strictlyinfrontof*(A,C)

Rule Set 4.5 The relation *touchfrombehind* implies the relation *strictlyinfrontof*.

touchfrombehind(A,B) $\Longrightarrow$ *strictlyinfrontof*(B,A)

Rule Set 4.6 The relationships between *strictlyinfrontof* and *infrontof* are as follows:

a) *strictlyinfrontof*(A,B) $\Longrightarrow$ *infrontof*(A,B)
b) *strictlyinfrontof*(A,B) $\wedge$ *infrontof*(B,C) $\Longrightarrow$ *strictlyinfrontof*(A,C)
c) *infrontof*(A,B) $\wedge$ *strictlyinfrontof*(B,C) $\Longrightarrow$ *strictlyinfrontof*(A,C)

Appendix B

Query Language Grammar Specification

```
/* Select-from-where */
<query> := select <target> from all [where <condition>] ‘;’
        | select <target> from <videolist> where <condition> ‘;’
        | select segment [‘,’ <variablelist>] from <range>
          where <condition> ‘;’
        | select <variablelist> from <range> where <condition> ‘;’
        | select <aggregate> ‘(’ segment ‘)’ [‘,’ segment]
          [‘,’ <variablelist>] from <range> where <condition> ‘;’

<target> := <video> [‘:’  (<number> | random ‘(’ <number> ‘)’)]

/* Aggregate Functions */
<aggregate> := average | sum | count

<range> := all | <videolist>

<video> := video [[last] <time> [seconds]]

<videolist> := [<videolist> ‘,’] <vid>
```

```
/* Condition specification */
<condition> := ‘(’ <condition> ‘)’ | not ‘(’ condition ‘)’
        | <condition> and <condition> | <condition> or <condition>
        | <condtype1> | <condtype2> | <condtype3> | <condtype4>

<condtype1> := <appearance> | <directional> | <topological> |
        <tdimension> | <external-predicate>

<condtype2> := <variable> <cop> (<atom> | <variable>)
        | <variable> ‘=’ <tprojection>

<condtype3> := <condition> <tmpred> <condition>
        | ‘(’ <condition> <tmpred> <condition> [<timegap>] ‘)’
        <trepeat>

<condtype4> := <trajectory-query> | ‘(’ <trajectory-query> ‘)’
        <trepeat>

<appearance> := appear ‘(’ <objectlist> ‘)’

<directional> := <direction> ‘(’ <object> ‘,’ <object> ‘)’

<topological> := <tpred> ‘(’ <object> ‘,’ <object> ‘)’

<tdimension> := <tdpred> ‘(’ <object> ‘,’ <object> ‘)’

<external-predicate> := <predicate-name> ‘(’ <objectlist> ‘)’

<tprojection> := project ‘(’ <object>
        [‘,’ <spatial-condition>] ‘)’
```

```
/* Trajectory condition */
<trajectory-query> := tr ‘(’ <object> ‘,’
        (<trajectory1> ‘)’ [<similarity>]
        | <trajectory2> ‘)’ [<simthreshold>]) [<timegap>]

<trajectory1> := <variable> | ‘[’ <dircomponent> ‘,’
        <dispcomponent> ‘]’

<trajectory2> := ‘[’ <dircomponent> ‘]’

<dircomponent> := ‘[’ <dirlist> ‘]’

<dispcomponent> := ‘[’ <displist> ‘]’

<similarity> := <simthreshold> [dirweight <dirweight>
        | dspweight <dspweight>]

<simthreshold> := sthreshold <threshold>

<timegap> := tgap <time>

<displist> := [<displist> ‘,’] <dspvalue>

<dirlist> := [<dirlist> ‘,’] <fdirection>

<trepeat> := repeat [<number>]

/* Local condition for trajectory projection */
<spatial-condition> := ‘(’ <spatial-condition> ‘)’
        | not ‘(’ <spatial-condition> ‘)’
        | <spatial-condition> and <spatial-condition>
        | <spatial-condition> or <spatial-condition>
        | <appearance> | <directional>
        | <topological> | <tdimension>
        | <variable> <cop> <object> | <external-predicate>
```

```
<direction> := left | right | above | below | <fdirection>

<fdirection> := west | east | north | south | northeast
        | southeast | northwest | southwest

<tpred> := equal | contains | inside | cover | coveredby
        | disjoint | overlap | touch

<tdpred> := infrontof | behind | sinfrontof | sbehind | tfbehind
        | tdfbehind | samelevel

<tmpred> := before | meets | overlaps | starts | during
        | finishes | ibefore | imeets | ioverlaps | istarts
        | iduring | ifinishes

<object> := <variable> | <atom>

<objectlist> := [<objectlist> ‘,’] <object>

<variablelist> := [<variablelist> ‘,’] <variable>

<vid> := (1-9)(0-9)*

<number> := (1-9)(0-9)*

<time> := (1-9)(0-9)*

<variable> := (A-Z)(A-Za-z0-9)*

<atom> := (a-z)(A-Za-z0-9)*
```

<predicate-name>[1]:= (a-z)(A-Za-z0-9_)*

<cop> := ‘=’ | ‘‘!=’’

<threshold> := 0 ‘.’ (0* (1-9) 0*)+

<dspweight> := 0 [‘.’ (0-9)*] | 1

<dirweight> := 0 [‘.’ (0-9)*] | 1

<dspvalue> := (1-9)(0-9)*

[1]Lexer recognizes such a character sequence as an external predicate name iff it is different from any predefined predicate and construct in the language.

Appendix C

Query Processing Functions

In this chapter, pseudo-codes of some of the functions used for spatio-temporal query processing on video data are given. These functions are very much simplified in that many implementation details have been omitted for the sake of simplicity and understanding. Hence, they should be considered as a very coarse and general presentation of our spatio-temporal query processing strategy.

C.1 Prolog Subqueries

```
int plQuery(CQueryNode* qnode)
Begin-CODE
  fetchSubquery(querynode:qnode, subquery:α)
  fetchVariables(querynode:qnode, variables:β)
  constructPrologQuery(subquery:α, variables:β, query:σ)
  If(β is not empty AND variables in β have previously computed values
    to use)
  Begin-IF
    For each value list ς for the variables in β do
    Begin-FOR
      CallProlog(query:σ, valuelist:ς, results:γ)
      If(γ is not empty)
        addResults(results:γ, resultset:ρ)
    End-FOR
  End-IF
  Else
```

```
  Begin-ELSE
    callProlog(query:σ, results:γ)
    If(γ is not empty)
      addResults(results:γ, resultset:ρ)
  End-ELSE
  pushResultSettoStack(resultset:ρ)
  If(ρ is empty) return NO_ANSWER
  Else return ANSWER
End-CODE
```

C.2 Similarity-Based Object-Trajectory Subqueries

```
int trQuery(CQueryNode* qnode)
Begin-CODE
  fetchSubquery(querynode:qnode, subquery:α)
  fetchVariables(querynode:qnode, variables:β)
  fetchRepeat(querynode:qnode, repeat:τ)
  callTrajectoryProcessor(subquery:α, variables:β, repeat:τ,
    resultset:ρ)
  pushResultSettoStack(resultset:ρ)
  If(ρ is empty) return NO_ANSWER
  Else return ANSWER
End-CODE
```

C.3 Trajectory-Projection Subqueries

```
int prQuery(CQueryNode* qnode)
Begin-CODE
  fetchSubquery(querynode:qnode, subquery:α)
  fetchVariables(querynode:qnode, variables:β)
  callTrajectoryProjector(subquery:α, variables:β, resultset:ρ)
  pushResultSettoStack(resultset:ρ)
  If(ρ is empty) return NO_ANSWER
  Else return ANSWER
End-CODE
```

C.4 Operator AND

```
CResultSet* opAnd(CResultSet* s1, CResultSet* s2, CQueryNode* qnode)
Begin-CODE
  If(s1->empty() OR s2->empty()) return an empty result set (no answer)
  callIntervalProcessor(operator:AND, resultset:s1, resultset:s2,
    resultset:ρ)
  return ρ
End-CODE
```

C.5 Operator OR

```
CResultSet* opOr(CResultSet* s1, CResultSet* s2, CQueryNode* qnode)
Begin-CODE
  If(s1->empty() AND s2->empty()) return an empty result set (no answer)
  If(s1->empty()) return s2
  If(s2->empty()) return s1
  callIntervalProcessor(operator:OR, resultset:s1, resultset:s2,
    resultset:ρ)
  return ρ
End-CODE
```

C.6 Operator NOT

```
CResultSet* opNot(CResultSet* s, int vlength[1], CQueryNode* qnode)
Begin-CODE
  If(s->empty()) return a result set with an empty object table
    (equivalent to TRUE)
  callIntervalProcessor(operator:NOT, resultset:s, videolength:vlength,
    resultset:ρ)
  return ρ
End-CODE
```

[1] video size in frames

C.7 Temporal Operators

```
CResultSet* before(CResultSet* s1, CResultSet* s2, CQueryNode* qnode,
  int fgap[2])
Begin-CODE
  return process(resultset:s1, resultset:s2, operator:BEFORE,
    querynode:qnode, framegap:fgap)
End-CODE
```

```
CResultSet* meets(CResultSet* s1, CResultSet* s2, CQueryNode* qnode,
  int fgap)
Begin-CODE
  return process(resultset:s1, resultset:s2, operator:MEETS,
    querynode:qnode, framegap:fgap)
End-CODE
```

```
CResultSet* overlaps(CResultSet* s1, CResultSet* s2, CQueryNode* qnode,
  int fgap)
Begin-CODE
  return process(resultset:s1, resultset:s2, operator:OVERLAPS,
    querynode:qnode, framegap:fgap)
End-CODE
```

```
CResultSet* during(CResultSet* s1, CResultSet* s2, CQueryNode* qnode,
  int fgap)
Begin-CODE
  return process(resultset:s1, resultset:s2, operator:DURING,
    querynode:qnode, framegap:fgap)
End-CODE
```

```
CResultSet* starts(CResultSet* s1, CResultSet* s2, CQueryNode* qnode,
  int fgap)
Begin-CODE
  return process(resultset:s1, resultset:s2, operator:STARTS,
    querynode:qnode, framegap:fgap)
End-CODE
```

[2]frame gap for the repeating condition computed from time gap (tgap) and video rate if tgap is specified (its default value is 1 frame) -used for repeat-

```
CResultSet* finishes(CResultSet* s1, CResultSet* s2, CQueryNode* qnode,
  int fgap)
Begin-CODE
  return process(resultset:s1, resultset:s2, operator:FINISHES,
    querynode:qnode, framegap:fgap)
End-CODE
```

```
CResultSet* process(CResultSet* s1, CResultSet* s2, int fcode,
  CQueryNode* qnode, int fgap)
Begin-CODE
  If(s1->empty() OR s2->empty()) return an empty result set (no answer)
  fetchRepeat(querynode:qnode, repeat:τ)
  callIntervalProcessor(operator:fcode, resultset:s1, resultset:s2,
    repeat:τ, framegap:fgap, resultset:ρ)
  return ρ
End-CODE
```