

QUERY PROCESSING FOR AN MPEG-7 COMPLIANT VIDEO DATABASE

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By

Hayati Çam

September, 2008

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

Prof. Dr. 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 thesis for the degree of Master of Science.

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

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

Assoc. Prof. Dr. Ahmet Coşar

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

Asst. Prof. Dr. İbrahim Körpeođlu

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

Dr. Cengiz Çelik

Approved for the Institute of Engineering and Science:

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

ABSTRACT

QUERY PROCESSING FOR AN MPEG-7 COMPLIANT VIDEO DATABASE

Hayati Çam

M.S. in Computer Engineering

Supervisors: Prof. Dr. Özgür Ulusoy and

Assoc. Prof. Dr. Uğur Güdükbay

September, 2008

Based on the recent advancements in multimedia, communication, and storage technologies, the amount of audio-visual content stored is increased dramatically. The need to organize and access the growing multimedia content led researchers to develop multimedia database management systems. However, each system has its own way of describing the multimedia content that disables interoperability among other systems. To overcome this problem and to be able to standardize the description of audio-visual content stored in those databases, MPEG-7 standard has been developed by MPEG (Moving Picture Experts Group).

In this thesis, a query language and a query processor for an MPEG-7 compliant video database system is proposed. The query processor consists of three main modules: *query parsing module*, *query execution module*, and *result fusion module*. The query parsing module parses the XML based query and divides it into sub-queries. Each sub-query is then executed with related query execution module and the final result is obtained by fusing the results of the sub-queries according to user defined weights. The prototype video database system *BilVideo v2.0*, which is formed as a result of this thesis work, supports spatio-temporal and low level feature queries that contain any weighted combination of keyword, temporal, spatial, trajectory, and low level visual feature (color, shape and texture) queries. Compatibility with MPEG-7, low-level visual query support, and weighted result fusion feature are the major factors that highly differentiate between *BilVideo v2.0* and its predecessor, *BilVideo*.

Keywords: MPEG-7, XML databases, video databases, multimedia databases, query processing, content-based retrieval, video query languages.

ÖZET

MPEG-7 UYUMLU VIDEO VERİTABANINDA SORGU İŞLEME

Hayati Çam

Bilgisayar Mühendisliği, Yüksek Lisans

Tez Yöneticileri: Prof. Dr. Özgür Ulusoy ve

Doç. Dr. Uğur Güdükbay

Eylül, 2008

Mültimedya, iletişim ve depolama teknolojilerindeki gelişmeler sayesinde depolanan işitsel-görsel içerik miktarı önemli ölçüde artmıştır. Bu büyüyen mültimedya içeriğini düzenlemek ve içeriği erişilebilir hale getirmek ihtiyacı araştırmacıları mültimedya veri tabanı sistemleri geliştirmeye yöneltmiştir. Fakat, geliştirilen sistemlerin mültimedya içeriğini tanımlama yollarının farklı olması sistemlerin birlikte işlerliğine olanak vermiyordu. Bu sorunu çözmek ve işitsel-görsel içeriğin tanımlanmasını standartlaştırmak için MPEG grubu tarafından MPEG-7 standardı geliştirilmiştir.

Bu tezde, MPEG-7 uyumlu bir video veri tabanı için sorgu dili ve sorgu işlemcisi önerilmektedir. Sorgu işlemcisi üç ana parçadan oluşmaktadır: *sorgu ayrıştırıcı*, *sorgu yürütücü*, ve *sonuç birleştirici*. Sorgu ayrıştırıcı, XML tabanlı sorguyu ayrıştırır ve alt sorgulara böler. Her bir alt sorgu ilgili sorgu yürütücü tarafından çalıştırılır ve kullanıcı tarafından belirlenen ağırlıklara göre alt sorgu cevapları birleştirilerek asıl sorgu sonucu oluşturulur. Bu tez çalışması sonucunda oluşan BilVideo v2.0 video veritabanı sistemi anahtar kelime tabanlı, yerleşimsel, zamansal, hareket izdüşüm ve alt seviyedeki (renk, şekil ve desen) video sorgularına olanak tanımaktadır. *BilVideo v2.0* MPEG-7 uyumluluğu, alt seviyedeki sorguları desteklemesi ve ağırlıklı sorgu birleştirmesiyle atası olan *BilVideo* sisteminden ayrılmaktadır.

Anahtar sözcükler: MPEG-7, XML veritabanları, video veri tabanları, Mültimedya veri tabanları, sorgu işleme, içerik-tabanlı veri alma, video sorgu dilleri.

Acknowledgement

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

I am grateful to Assoc. Prof. Dr. Ahmet Coşar, Asst. Prof. Dr. İbrahim Körpeoğlu and Dr. Cengiz Çelik for reading and reviewing this thesis. Additionally, I cannot forget the invaluable support of Muhammet Baştan in all phases of this thesis work.

I would like to thank my friends Sevgi Keskin and Ülkü Tuğba Terzi for their caring friendship and motivation. I also would like to thank Tuğçe Çınar for her friendship and profound assistance. I am grateful to all of my friends who filled my life with joy.

Above all, I owe everything to my family, who supported me in each and every way, believed in me permanently and inspired me in all dimensions of life. Without their everlasting love, this thesis would never be completed.

Contents

- 1 Introduction** **1**
 - 1.1 Motivation 2
 - 1.2 System Architecture 3
 - 1.3 Features of the System 5
 - 1.4 Organization of the Thesis 6

- 2 Background** **8**
 - 2.1 MPEG-7 8
 - 2.2 XML Databases 11
 - 2.2.1 Native versus XML Enabled Databases 12
 - 2.2.2 Tamino 13
 - 2.3 Multimedia Database Systems 14

- 3 The Query Language** **16**
 - 3.1 Query Statements and Types 17
 - 3.1.1 Keyword Query 18

3.1.2	Temporal Query	18
3.1.3	Spatial Query	19
3.1.4	Trajectory Query	20
3.1.5	Low Level Query	22
3.1.6	Composite Query	28
3.2	Query Samples	30
4	Query Processing	35
4.1	Software Architecture	35
4.2	Query Parsing	37
4.3	Query Execution	39
4.3.1	Keyword Query Execution	40
4.3.2	Temporal Query Execution	41
4.3.3	Spatial Query Execution	42
4.3.4	Trajectory Query Execution	43
4.3.5	Low Level Feature Query Execution	44
4.4	Result Fusion	45
5	User Interface and System Performance	47
5.1	Visual Query Interface	47
5.1.1	General View	49
5.1.2	Keyword Query Interface	50

5.1.3	Temporal Query Interface	50
5.1.4	Spatial Query Interface	50
5.1.5	Trajectory Query Interface	53
5.1.6	Low Level Query Interface	55
5.1.7	Composite Query Interface	56
5.2	Sample Queries	57
5.3	Performance	61
6	Conclusion and Future Work	64
	Bibliography	66

List of Figures

1.1	The system architecture of <i>BilVideo v2.0</i>	4
2.1	MPEG-7 descriptors and description schemes used in <i>BilVideo v2.0</i>	10
3.1	The spatial west relation	20
4.1	The software architecture of the query processor	36
5.1	The architecture of visual query interface	48
5.2	The top level window of the visual query interface	49
5.3	The video table of contents	51
5.4	The options menu: the user selects the output type of the query .	52
5.5	The keyword query interface	52
5.6	The temporal query interface	52
5.7	The spatial query interface	53
5.8	The trajectory query interface	54
5.9	The low level query interface	55

5.10 The result of the segmentation process	56
5.11 The composite query interface	56

List of Tables

3.1	The temporal query relation types	19
3.2	The spatial query relation types	20
4.1	The maximum distances for the keyframes in TRECVID 2008 Development Data	45
5.1	The query execution times	62

Chapter 1

Introduction

With the technological improvements achieved in the last two decades, recording, playing and storing Audio-Visual (AV) data become an essential part of the daily life. Thousands of broadcasting companies tend to produce 24 hour content everyday, the film industry produces hundreds of movies every year, and even regular people record AV content by using their mobile phones. All of these mean the amount of AV content made available increases in every second. To be able to organize this tremendous valuable content and to make this available information accessible, researchers from academic and industrial world proposed the notions of Multimedia Database Management Systems and Content Based Visual Information Retrieval Systems. Some database vendors extended their traditional database management systems, which are not suitable to store multimedia content, in order to support AV content storage whereas some other companies developed special database management systems for multimedia. Unfortunately, the descriptions of audio and visual content stored in these databases were completely different. To bring this chaotic situation to an end, MPEG (Moving Picture Experts Group), which is a working group of ISO/IEC responsible for development of standards for digital audio and video, released MPEG-7 standard.

In this thesis, query processing for an MPEG-7 compliant multimedia database management system, namely *BilVideo v2.0*, is introduced. *BilVideo*

v2.0 is designed to be extensible to offer a full-fledged multimedia querying experience. Currently, it is focused on querying videos, but its design enables it to be extendible to support audio queries and image queries as well. The query processor of *BilVideo v2.0* allows user to query the videos using keyword, temporal and spatial queries, which are conceptually based on the database system aspect. The related results are definite. Additionally, *BilVideo v2.0* supports trajectory and low level feature (color, texture, shape) queries, where results are indefinite, but ranked according to the distance to query. And this is the content based retrieval aspect of *BilVideo v2.0*.

The query processor of *BilVideo v2.0* contains three major components. The first component is query parsing module that is used for parsing the XML based query and dividing it into sub-queries. It is worth mentioning that *BilVideo* is unique for using a simple XML based query language, which is introduced with the influence of very complex query languages in current multimedia video databases. After parsing, each sub-query is sent to the related query execution module (e.g., temporal query is sent to temporal query execution module), which forms the second major component of the query processor. The query execution module performs XQuery on MPEG-7 compliant XML files and used for generating the results. The results from each sub-query is finally combined in third component named result fusion module. The query processor works in a client-server architecture, and is a distributable component of *BilVideo v2.0* video database.

1.1 Motivation

To keep pace with advancements in multimedia databases, Multimedia Database Group (MDG) of Bilkent University developed a video database management system which is introduced in [6, 7, 8], named *BilVideo*. *BilVideo* supports spatio-temporal queries along with trajectory and semantic queries. It introduces a SQL-like query language to support these types of queries. Additionally, it has its own way of storing descriptions about the videos. The description of each

video is stored in a relational database management system and the inter-object relations of objects in the videos are stored in a knowledge-base. The query results are combinations of SQL query results from the relational database and rule based query results from the knowledge-base.

In fact, with its impressive results *BilVideo* is an advanced system. However, it needs to be improved for the following reasons. Firstly, its way of storing description of video is not compatible with any standard. The knowledge-base and relational database table structure are specific to *BilVideo*. Moreover, the implementation, and thus, the maintenance of the system, is complicated because of the use of several programming languages such as C++, Java, Prolog.

To overcome the problems explained above, *BilVideo v2.0* is proposed. It is designed to use standard MPEG-7 schemes to store data in order to solve the interoperability problem that its predecessor had faced. To store the description data which are MPEG-7 schema compliant XML files, it makes use of an industrial XML database management system, called Tamino, which is used for storing and querying large XML collections. This also saves overhead of maintaining two different storage environments: Relational database management system and fact-base. Low level query (i.e., color, shape, texture) support is introduced to make it a full-fledged video database. In addition, the programming environment is unified by using only C++.

1.2 System Architecture

BilVideo v2.0 has a distributable, extensible and scalable architecture that theoretically allows it to be used either by end-users in personal computers or by broadcasting companies or even as a database for video search engines in the Internet.

The main building blocks of the system are:

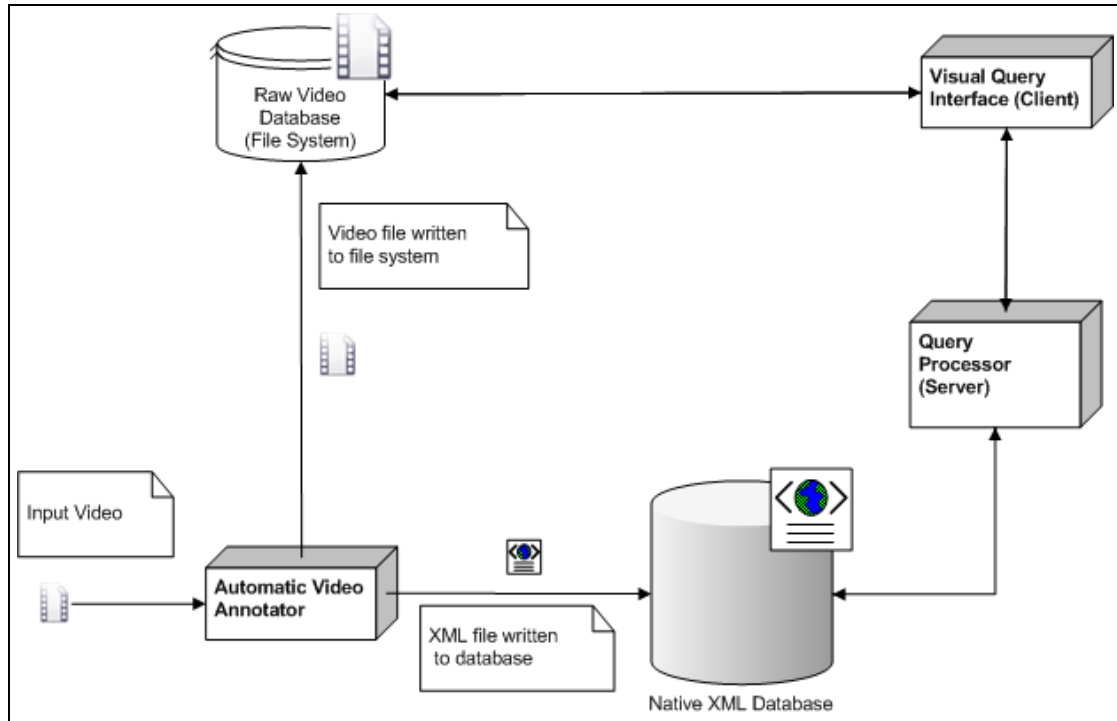


Figure 1.1: The system architecture of *BilVideo v2.0*

Automatic Video Annotator: Parses the input video and extracts low level features. Additionally, segments the image and helps the user to easily label the objects in the video. The output of this component is an MPEG-7 compliant XML file that is to be stored in the native XML database. The processed video is also sent to the raw video database.

Visual Query Interface: It is the query interface prepared to query the video database. It enables the user to perform

- keyword queries,
- temporal queries by keywords,
- temporal queries by sketch,
- spatial queries by keywords,
- spatial queries by sketch,
- trajectory queries,
- low level queries, and

- composite queries (which is any combination of above mentioned queries).

Query Processor: Query processor parses the query from visual query interface or from any source that is compatible with BilVideo query language. It processes the query and sends the result back to the requesting client.

Native XML Database: Native XML database is the logical storage unit where the XML files are stored. In current version, it is Software AG Tamino XML Database. However, it could be replaced by any native XML database that supports standard XQUERY language.

Raw Video Database: It is the physical file system location where the video files are stored. In distributed deployment scenario, raw video database location should be accessible by visual query interface by either network mapping or by web server installation.

1.3 Features of the System

BilVideo v2.0 is planned to be a full-fledged multimedia database management system. The visual part of the videos are supported in the current version. The features of the current version are as follows:

- compatibility with MPEG-7,
- spatio-temporal and object appearance query support,
- motion trajectory query support,
- the low level query support,
- video table of contents, and
- result fusion by user-specified weights.

The low level queries support the following feature descriptors:

1. Color features:
 - dominant color(s),
 - scalable color,
 - color layout,
 - color-structure, and
 - GoF/GoP color.
2. Texture features:
 - homogeneous texture,
 - texture browsing, and
 - edge histogram.
3. Shape features:
 - region shape and
 - contour shape.

1.4 Organization of the Thesis

The rest of this thesis is organized as follows:

- Chapter 2 gives an overview of MPEG-7 and explains the aspects of MPEG-7 scheme used in this work in detail. Then, it continues with a brief overview of XML databases and ends with the review of the research in the literature that is related to our work.
- Chapter 3 explains the query language used in *BilVideo v2.0* and provides example queries of each type of query supported by the system.
- Chapter 4 presents the query processor developed for *BilVideo v2.0*. The architecture of query processor and its components are detailed in this chapter.

- Chapter 5 presents the visual query interface of *BilVideo v2.0* and provides some performance results about *BilVideo v2.0*
- Chapter 6 states the conclusions of the thesis and gives a general perspective of the possible future work.

Chapter 2

Background

2.1 MPEG-7

MPEG-7 (Multimedia Content Description Interface) is an ISO/IEC standard developed by MPEG (Moving Picture Experts Group) for describing the multimedia content data. MPEG-7 does not target any specific application, rather the aim of this standardization is to support as many application areas as possible.

The scope of MPEG-7 is the description of multimedia content data, which differs it from other standards that are developed by MPEG, such as MPEG-1, MPEG-2, and MPEG-4. MPEG-7 standard consists of the following elements [15]:

Description Tools:

1. *Descriptors (D)*: The element that defines the features supported by the standard.
2. *Descriptor Schemes (DS)*: The element that defines the relationship among features and other descriptor schemes.

Description Definition Language (DDL): The syntax that is used to define existing descriptors and descriptor schemes. Additionally, DDL allows generation of new descriptors and descriptor schemes.

System Tools: The tools that defines the optimized ways of storing and transmitting MPEG-7 content.

To supply a better understanding of the elements of the standard, the following example can be given. The color layout is a descriptor in MPEG-7. On the other hand, the moving region is a descriptor scheme that may contain color layout descriptor and a still region descriptor scheme, which may contain additional descriptors. DDL can be used to define a new descriptor that can be contained in still region descriptor scheme. Finally, if you want to transfer the MPEG-7 content to another computer after creation, you can use the system tools that define how to binarize the data to improve transfer performance.

MPEG-7 allows the description of various aspects of the multimedia content. For instance, creation date, director, encoding, colors, textures, melodies, objects, events, interactions of objects, regions, motion tracking information and a lot more data about a video can be kept in a single content description data file.

BilVideo v2.0 currently deals with only the visual part of MPEG-7, which contains color, texture, shape information, objects and their spatio-temporal relations, and trajectories of objects in a video segment. To store and query these kind of information, *BilVideo v2.0* makes use of the descriptors and description schemes which are detailed in the following.

In Figure 2.1, it can be observed that *BilVideo v2.0* deals with only the video part of the standard. The figure also depicts the recursive structure allowed in MPEG-7. According to the figure, the videos are divided into shorter video segments, named shots. A shot is a single piece of video without cuts. After obtaining shots, each shot is again divided into key-segments and moving regions. A key-segment is a piece of shot, whose boundary is determined by the introduction of a new object to the frame or by the exit of an existing object from the frame.

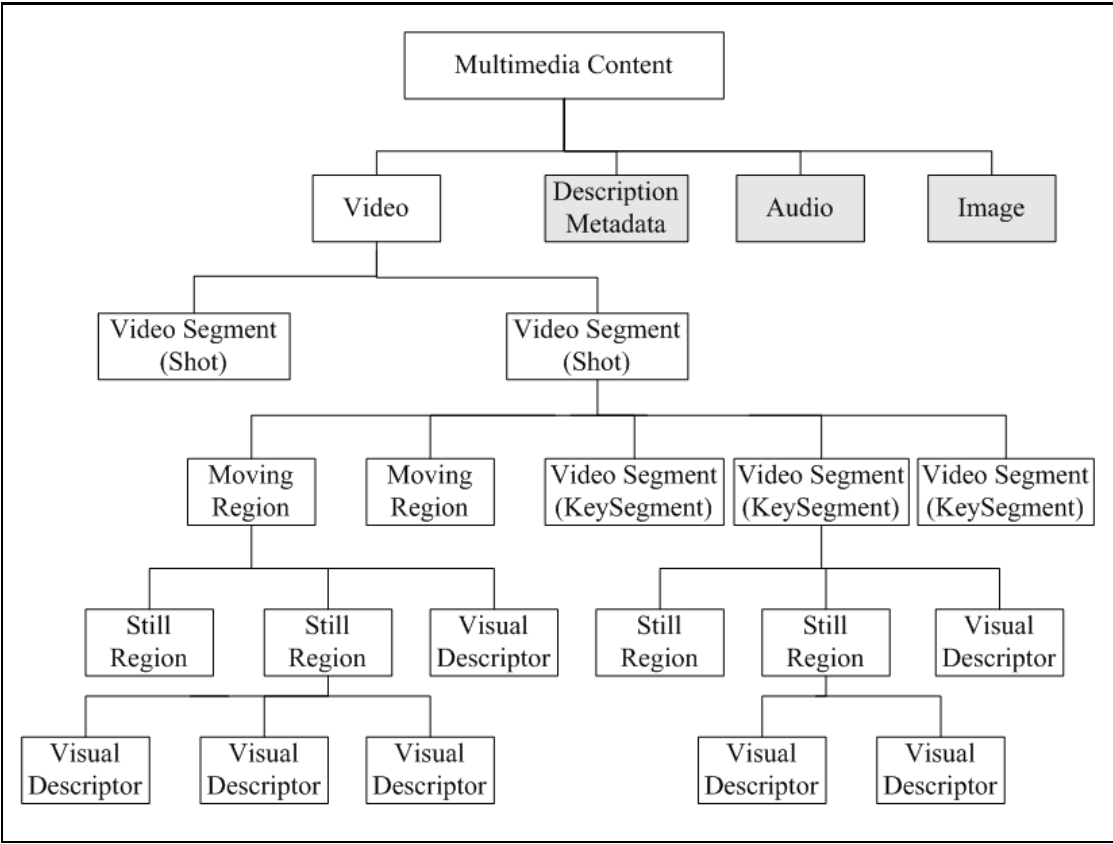


Figure 2.1: MPEG-7 descriptors and description schemes used in *BilVideo v2.0*

There are visual descriptors that are used in moving regions, still regions or key-segments. The visual descriptors used in MPEG-7 are as follows:

1. *Color descriptors:*

- dominant color (supported),
- scalable color (supported),
- color layout (supported),
- color-structure (supported), and
- GoF/GoP color (not supported).

2. *Texture descriptors:*

- homogeneous texture (supported),
- texture browsing (not supported), and
- edge histogram (supported).

3. *Shape descriptors:*

- region shape (supported),
- contour shape (supported), and
- shape 3D (not supported).

2.2 XML Databases

XML (eXtensible Markup Language) is a markup language, which is a W3C recommendation, to generate custom markup languages [24]. It does nothing for the presentation of the data; the main goal of its design is to carry data. XML simplifies data sharing and data transportation. Sometimes, it is required to store XML-formatted data in a repository, which allows easy retrieval and manipulation of data. For this purpose, XML databases are proposed. XML databases are repositories or databases that support the storage of XML files

in a data-centric or document-centric way. The databases that store XML data in data-centric way are called *XML-enabled databases*. On the other hand, the databases that store the XML data in a document-centric way are called *Native XML Databases (NXD)*.

2.2.1 Native versus XML Enabled Databases

Conceptually, there are two different ways to store XML documents in a database. The first way is to map the data model of XML Document to a database model and convert XML data according to this mapping. The other way is to map XML model into a fixed set of persistent structures, which can store any XML document. Databases that support the former method are called XML-enabled databases whereas databases that support the latter method are called native XML databases. XML-enabled databases map instances of the XML data model to instances of their own data model (relational, hierarchical, object-oriented, etc.). On the contrary, native XML databases use the XML data model directly [16].

The advantages of the native XML databases in comparison with the XML-enabled databases are as follows:

- Native XML databases do not require user to know XML document schema in advance. A native XML data store is able to store and retrieve any well-formed XML document, even if schema information of the document is not available. An XML-enabled database, however, needs schema definitions for each table, so a document with an unknown tag would require a change request for a new schema definition, to be built and approved before it can be put into production.
- Native XML databases support data models that do not fit into other models (relational, hierarchical, object-oriented). Relational database management systems may appear to be a possible choice to facilitate the exchange of XML objects. However, the table-based data model of the XML enabled

database does not suit the hierarchical and interconnected nature of XML objects.

- Native XML databases provide extensibility. On the other hand, XML-enabled relational databases and more advanced databases, such as multi-dimensional relational databases or object-oriented databases, cannot handle data with dynamic structure, which is the key to the extensibility of XML.
- Native XML databases do not suffer from the cost of mapping, which is a problem faced in XML-enabled databases. Storing XML data natively has an enormous advantage in comparison to the XML-enabled relational databases, because no extra data conversion layer is required as, for example, needed for XML-enabled relational databases and the document structure is kept intact.
- Native XML databases process queries faster. An XML-enabled relational database would need to break an XML document down into a multitude of interrelated tables. A query against this database would result in many relational retrieval and join operations, requiring high processing power to overcome a considerable degradation of performance.

2.2.2 Tamino

For the reasons listed above, a native XML database, named Tamino, is used to store MPEG-7 XML descriptions of video content in *BilVideo v2.0* system. Tamino XML Server is able to work on both Linux and Windows environments [20]. It is a high performance data management platform based on XML standards, which is developed by Software AG and is built to:

- efficiently store XML documents natively, that is in their original format,
- store non-XML documents such as Microsoft Office documents or PDF files,

- expose information residing in various external XML or non-XML sources (legacy data) or applications to the outside world in XML format, and
- to search effectively on the information Tamino has access to.

Tamino supports XQuery [25] as well as X-Query [20] which is a language specific to Tamino database and easier to write for simple queries. However, we used only XQuery to be compatible with other XML databases.

2.3 Multimedia Database Systems

Multimedia database systems and content based retrieval systems are developed in a variety of researches. In this section, the works on multimedia database systems and MPEG-7 compliant multimedia database systems are mentioned.

One of the most important works in the area is the *Informedia Digital Video Understanding Research Project* [3, 4, 9]. The project combines several aspects of multimedia such as speech recognition, image understanding and natural language processing to segment and index videos. However, the output is not compatible with MPEG-7. The *Informedia II* project, which is under development, is planned to support MPEG-7.

Another notable system, *VideoQ* [2], uses the animated sketch paradigm to query its database. It allows the users to draw sketches and query the database according to the given sketch. The system allows to query along the database according to the following attributes: motion, spatio-temporal ordering, shape, color and texture. The results of the query are presented using a web-interface, which is implemented using the Java applet technology. The drawback about the system is that it has separate feature databases for each feature that is supported. Like Informedia, it does not make use of MPEG-7 standard to keep the features.

MARS (Multimedia Analysis and Retrieval System) [10] project brings together the research efforts in computer vision, compression, information management and database management to develop a multimedia database management

system. The system supported keyword queries, color, color layout, shape and texture queries. The system makes use of POSTGRES database [21], which is an object relational database.

BilVideo is also a video database system that has no MPEG7 support [6, 7, 8]. *BilVideo* supports spatio-temporal queries along with trajectory and semantic queries. The query can be sent to the system using a SQL-like query language. In *BilVideo*, the description of each video is stored in a relational database management system and the spatio-temporal relations between objects in the videos are stored in a knowledge-base. The query results are combinations of SQL query results from the relational database and rule based query results from the knowledge-base.

After the release of MPEG-7, while some researchers stick to their video descriptions, some others have started to migrate their system to be compatible with the new standard. *VIRS* [13] is one of the projects, that makes use of the new standard. The goal of the project was to automatically extract visual descriptors of MPEG-7 without human interaction, and allow the users to query the videos using Query-By-Example or Query-By-Drawing. The database does not contain any annotation, therefore it does not have the ability to allow temporal, spatio-temporal and keyword queries.

MPEG-7 MMDB [5] is another multimedia database that supports MPEG-7. The system makes use of Oracle Database with XML extensions. It has mapping rules to map the XML schema of MPEG-7 to the Oracle database tables. The system only supports image retrieval and audio querying. It does not have any support for videos.

When compared to the existing multimedia databases, the innovative parts of *BilVideo v2.0* are (i) the description model relying on XML-based MPEG-7 standard, (ii) a new query language, (iii) the query processor for MPEG-7, and (iv) supporting composite queries, which are any combination of keyword, temporal, spatial, trajectory, or low level feature queries.

Chapter 3

The Query Language

Unlike relational databases, multimedia databases store multimedia content along with textual data. Hence, they require different data models and query structures. Generally, the multimedia databases store multimedia content in the file system and store the description of these files in one of the following ways:

- in a relational database,
- in a relational database that has XML extension,
- in the file system as text files,
- in the file system as XML files, or
- in a native XML database.

BilVideo v2.0 makes use of a native XML database, while its predecessor uses a relational approach. No matter what strategy is chosen, each multimedia database has its own query language. Because specifying SQL queries for an MM-DBMS that makes use of a relational database, or writing XQuery for an MM-DBMS that makes use of an XML database is very difficult, and requires advanced knowledge of the database design and the query language. For instance, *BilVideo* introduced its own SQL-like query language. We have not used the same

query language because low level queries are difficult to describe in an SQL-like fashion. Since we are using XML files as the description data format, we decided to use XML-structured queries. In this way, we developed human and machine readable simple XML constructs and used them as query statements.

The following is a sample query statement to query videos where “Blair appears before Clinton”:

```
<BilVideoQuery outputType = "Video">
  <TemporalQuery type="before">
    <Object1>Blair</Object1>
    <Object2>Clinton</Object2>
  </TemporalQuery>
</BilVideoQuery>
```

3.1 Query Statements and Types

All *BilVideo v2.0* query statements start with `<BilVideoQuery>` tag. As an attribute, you can choose the output type of the query when possible by specifying a value for `outputType` attribute. The possible values for the output type are:

- Video
- Shot
- Key-segment

By specifying the output type, you mean “Return the videos [shots, key-segments] where the condition is satisfied”.

Then, one or more query type elements are followed by element name according to the query types. The possible query types embedded inside `<BilVideoQuery>` are:

- <KeywordQuery>
- <TemporalQuery>
- <SpatialQuery>
- <TrajectoryQuery>
- <LowLevelQuery>

In the following sub-sections, each query type, its related syntax, and some sample queries along with their meanings are described.

3.1.1 Keyword Query

Keyword query is used for searching object appearance in videos, shots or key-segments. It allows users to query for an object and a group of objects by using grouping words (and, or). Its syntax is as follows:

```
<BilVideoQuery outputType = "[output_type]">
  <KeywordQuery>
    <FreeText>[keyword_search_text]</FreeText>
  </KeywordQuery>
</BilVideoQuery>
```

The keyword search text is composed of any keyword, grouping symbols (i.e., “(”, “)”) and search reserved words (“and”, “or”).

3.1.2 Temporal Query

Temporal query is used to search objects according to temporal properties of object appearances in videos, shots or key-segments. It allows user to specify two objects and their temporal relation. Temporal relation types that are supported

by *BilVideo v2.0* are given in Table 3.1. The syntax of the temporal query is as follows:

```
<BilVideoQuery outputType = "[output_type]">
  <TemporalQuery type="[temporal_relation_type]">
    <Object1>[name_of_first_object]</Object1>
    <Object2>[name_of_second_object]</Object2>
  </TemporalQuery>
</BilVideoQuery>
```

Type	Meaning
Before	The first object appears before the second object
After	The first object appears after the second object
Equal	The first object and the second object appear at the same time and disappear at the same time
NotEqual	The first object and the second object do not appear at the same time
During	The first object appears while the second object is appearing
Contains	The second object appears while the first object is appearing
Overlaps	The first object appears, the second object appears, the first object disappears a little bit later
OverlappedBy	The second object appears, the first object appears, the second object disappears a little bit later
Meets	The second object appears. Then, the first object appears,
MetBy	The first object appears. Then, the second object appears
Starts	The first object and the second object appear at the same time
Finishes	The first object and the second object disappear at the same time

Table 3.1: The temporal query relation types

3.1.3 Spatial Query

Spatial query is used to search objects according to spatial locations of object in frames. It allows user to specify two objects and their spatial relation. In Figure 3.1, the spatial west relation is shown. The light gray object is assumed to be on the west of dark gray object according to our algorithms. Spatial relation types that are supported by *BilVideo v2.0* are given in Table 3.2. The syntax of the spatial query is as follows:

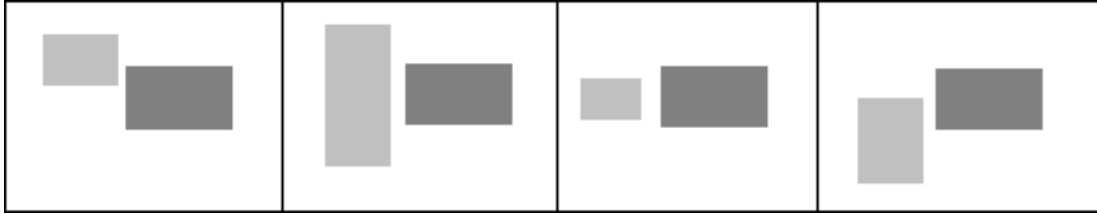


Figure 3.1: The spatial west relation

```

<BilVideoQuery outputType = "[output_type]">
  <SpatialQuery type="[spatial_relation_type]">
    <Object1>[name_of_first_object]</Object1>
    <Object2>[name_of_second_object]</Object2>
  </SpatialQuery>
</BilVideoQuery>

```

Type	Meaning
West	The first object is on the west of the second
East	The first object is on the east of the second
North	The first object is on the north of the second
South	The first object is on the the south of the second
NorthWest	The first object is on the north west of the second
NorthEast	The first object is on the north east of the second
SouthWest	The first object is on the south west of the second
SouthEast	The first object is on the south east of the second
Left	The first object is on the left of the second
Right	The first object is on the right of the second
Below	The first object is below the second
Above	The first object is above the second

Table 3.2: The spatial query relation types

3.1.4 Trajectory Query

Trajectory query is used to search object trajectories. It allows user to specify trajectory and to retrieve videos, shots or key-segments where an object moves in a similar trajectory. The syntax of the trajectory query is as follows:


```
<BilVideoQuery outputType="[output_type]">
  <TrajectoryQuery>
    <Params>
      <KeyTimePoint>
        <MediaRelIncrTimePoint>[t0]</MediaRelIncrTimePoint>
        <MediaRelIncrTimePoint>[t1]</MediaRelIncrTimePoint>
        <MediaRelIncrTimePoint>[t2]</MediaRelIncrTimePoint>
        <MediaRelIncrTimePoint>[t3]</MediaRelIncrTimePoint>
      </KeyTimePoint>
      <InterpolationFunctions>
        <KeyValue>[x0]</KeyValue>
        <KeyValue>[y0]</KeyValue>
        <KeyValue>[x1]</KeyValue>
        <KeyValue>[y1]</KeyValue>
        <KeyValue>[x2]</KeyValue>
        <KeyValue>[y2]</KeyValue>
        <KeyValue>[x3]</KeyValue>
        <KeyValue>[y3]</KeyValue>
      </InterpolationFunctions>
    </Params>
  </TrajectoryQuery>
</BilVideoQuery>
```

Trajectory query specification is more complicated because it uses an XML structure similar to the MPEG-7 schema. In this way, the parsing of the query and the parsing of the values stored in the database are unified. Here, for each `<MediaRelIncrTimePoint>`, we have two `<KeyValue>`s. `[x0]` and `[y0]` give the position of the object at `[t0]`.

3.1.5 Low Level Query

Low level query is used to search frames or objects in frames according to their color, shape or texture. It allows user to specify low level feature values and to retrieve videos, shots or key-segments that are similar to query feature values. Low level queries use a similar XML structure with MPEG-7 schema, like trajectory query. The syntax of the low level query is as follows:

```
<BilVideoQuery outputType="[output_type]">
  <LowLevelQuery searchIn = "[Search_In_Type]"
    weight="[Low_Level_Feature_Query_Weight]">
    ...
  </LowLevelQuery>
</BilVideoQuery>
```

In a low level query, we encounter two new attributes. The first one, named `SearchIn`, is used for specifying where to search the queried feature. The possible values are:

- whole frame: search this feature in a frame in the database,
- still region: search this feature in a still region in the database,
- moving region: search this feature in a moving region in the database, and
- shot: search this feature in a shot in the database.

The second attribute, named `weight`, is the weight of this low level feature among other low level features. The “...” part is different for each low level feature descriptor.

Since the low level feature queries use the low level feature descriptors, they are far from being human-readable. These queries are generated by Visual Query

Interface, so the user does not need to understand how the values are obtained. The samples below are provided to give an idea about the syntax of each type.

Dominant Color Feature Query: Dominant Color Descriptor (DCD) gives the distribution of salient colors in an image or image part. DCD provides an effective, compact and intuitive representation of colors in a region of interest [14]. DCD makes use of two other MPEG-7 color descriptors: *Color Space Descriptor* for specifying the used color space and *Color Quantization Descriptor* for specifying the used color quantization. DCD stores the representative (dominant) colors, their percentages, the optional color variances for each dominant color, and the optional spatial coherency of the dominant colors [23].

```
<BilVideoQuery outputType="[output_type]">
  <LowLevelQuery searchIn = "[Search_In_Type]" >
    <VisualDescriptor xsi:type="DominantColorType">
      <ColorSpacetype="RGB"/>
      <ColorQuantization>
        <Component>R</Component>
        <NumOfBins>8</NumOfBins>
        <Component>G</Component>
        <NumOfBins>8</NumOfBins>
        <Component>B</Component>
        <NumOfBins>8</NumOfBins>
      </ColorQuantization>
      <SpatialCoherency>23</SpatialCoherency>
      <Value>
        <Percentage>18</Percentage>
        <Index>255 255 255</Index>
        <ColorVariance>1 1 1</ColorVariance>
      </Value>
    </VisualDescriptor>
  </LowLevelQuery>
</BilVideoQuery>
```

Scalable Color Feature Query: Scalable Color Descriptor (SCD) gives the distribution of colors of an image or image part in HSV (Hue-Saturation-Value) color space. SCD is basically a color histogram encoded by a Haar transform. It uses the HSV colors space uniformly quantized to 255 bins [18]. The binary representation of SCD is scalable in terms of bin numbers. SCD stores the coefficients values of the histogram.

```
<BilVideoQuery outputType="[output_type]">
  <LowLevelQuery searchIn = "[Search_In_Type]">
    <VisualDescriptor numOfBitplanesDiscarded="0" numOfCoeff="128"
      xsi:type="ScalableColorType">
      <Coeff>-58 14 -88 40 -7 -1 9 27 -15 -9 3 18 11 7 14 22 7 -6
        -3 3 15 -3 0 1 14 0 1 0 15 4 1 -4 -2 3 -1 1 1 2 2 4 0 0 -3
        3 1 2 4 4 0 -3 -3 -1 -1 1 3 0 0 -15 -1 -2 1 0 -3 -3 -1 1 0
        0 0 -1 0 1 0 0 4 1 2 2 2 2 -2 1 -3 -3 -1 -3 0 -1 2 1 0 1 3
        3 2 0 -1 3 -1 -2 -1 0 2 2 2 -7 1 1 3 3 1 0 -1 -2 0 1 0 0 1
        0 1 1 1 0 1 0 -1 1</Coeff>
    </VisualDescriptor>
  </LowLevelQuery>
</BilVideoQuery>
```

Color Layout Feature Query: Color Layout Descriptor (CLD) specifies the spatial distribution of colors of an image. CLD is obtained in four steps [12]: (i) partition of image into 64 blocks, (ii) dominant color selection, (iii) Discrete Cosine Transform (DCT), and (iv) non-linear quantization of the zig-zag scanned DCT coefficients. CLD is suitable for the comparison of images in different scales because the distribution of colors does not change after rescaling.

```
<BilVideoQuery outputType="[output_type]">
  <LowLevelQuery searchIn = "[Search_In_Type]">
    <VisualDescriptor xsi:type="ColorLayoutType">
      <YDCCoeff>23</YDCCoeff>
      <CbDCCoeff>28</CbDCCoeff>
    </VisualDescriptor>
  </LowLevelQuery>
</BilVideoQuery>
```

```

    <CrDCCoeff>32</CrDCCoeff>
    <YACCCoeff5>9 10 15 22 12</YACCCoeff5>
    <CbACCCoeff2>6 14</CbACCCoeff2>
    <CrACCCoeff2>18 22</CrACCCoeff2>
  </VisualDescriptor>
</LowLevelQuery>
</BilVideoQuery>

```

Color Structure Feature Query: Color Structure Descriptor (CSD) is used to express local color features in images [18]. CSD is obtained by scanning the image with an 8x8 block in a sliding window approach. With each shift of the structuring element, the number of times a particular color is contained in the structure element is counted, and a color histogram is constructed in this way.

```

<BilVideoQuery outputType="[output_type]">
  <LowLevelQuery searchIn = "[Search_In_Type]">
    <VisualDescriptor colorQuant="2" xsi:type="ColorStructureType">
      <Values>0 85 2 0 0 0 0 0 0 0 255 69 12 8 0 0 0 0 0 0 0 0
      84 89 46 16 37 36 22 2 6 16 6 0 3 2 2 0 50 43 14 7 45 37 22 2
      24 25 12 4 5 7 2 0 27 34 38 16 7 7 2 2</Values>
    </VisualDescriptor>
  </LowLevelQuery>
</BilVideoQuery>

```

GoF/GoP Color Feature Query: Group-of-Frames/Group-of-Pictures Color Descriptor (GoF/GoP CD) defines a structure for color features representation of a collection of pictures or video frames by means of the SCD. It consists of the average, median, and intersection histograms of groups of frames, which are calculated based on the individual frame histograms [18].

```

<BilVideoQuery outputType="[output_type]">
  <LowLevelQuery searchIn = "[Search_In_Type]">
    <VisualDescriptor aggregation="Average" xsi:type="GoFGoPColorType">

```

```

<ScalableColor numOfBitplanesDiscarded="0" numOfCoeff="256">
  <Coeff>255 -50 -127 -11 -12 -24 -15 11 -31 -39 -9 -5 -12
  -10 -20 19 7 -13 -2 4 15 -11 1 6 15 -7 1 1 15 2 1 -4 -3 3
  1 4 3 1 0 -1 -4 -1 -6 0 0 -4 4 -1 -1 -3 -8 -3 2 0 2 -2 0
  -15 -3 -2 1 0 -3 -3 -1 1 2 -2 -1 -1 -2 1 -1 2 5 0 3 2 2 1
  -1 1 -3 -4 -1 -3 -4 -2 -1 -2 -2 1 1 3 2 0 -1 3 2 -4 -3 -1
  1 2 2 -7 -1 1 3 3 1 0 -4 0 -2 -1 -3 -7 1 0 1 -2 1 0 1 0 -1
  1 -1 0 0 0 0 0 0 0 1 1 -3 0 1 -1 0 0 1 1 -3 -1 1 -1 1 0 1
  0 0 -1 0 -3 1 3 1 1 -1 1 1 0 -1 0 2 0 -3 -1 1 -2 0 0 1 1
  -3 -1 0 0 1 -2 0 3 -1 0 0 0 1 -1 0 1 0 0 1 -2 -1 1 1 1 -3
  -2 1 -1 1 0 0 1 -2 -1 0 1 1 -2 0 3 0 0 0 1 0 -1 -1 -1 0 0
  3 -3 0 -1 1 -1 0 0 1 0 0 -1 0 1 0 0 0 1 0 -1 0 2 0 0 0 1
  0 1</Coeff>
</ScalableColor>
</VisualDescriptor>
</LowLevelQuery>
</BilVideoQuery>

```

Homogeneous Texture Feature Query: Homogeneous Texture Descriptor (HTD) characterizes the region texture using the energy and energy deviation values of the Fourier transform of the image. HTD vectors are made up of an image intensity mean, a standard deviation, 30 energy values and 30 energy deviations. For further details about the extraction of the vectors, see [17].

```

<BilVideoQuery outputType="[output_type]">
  <LowLevelQuery searchIn = "[Search_In_Type]">
    <VisualDescriptor xsi:type="HomogeneousTextureType">
      <Average>123</Average>
      <StandardDeviation>91</StandardDeviation>
      <Energy>179 198 200 202 193 192 210 188 174 190 162 151 198
      174 135 170 125 147 151 136 118 167 121 95 158 95 77 139 81
      61</Energy>
      <EnergyDeviation>176 197 204 200 198 193 200 180 173 190 152

```

```

    142 188 169 120 159 109 142 145 117 111 156 114 88 161 82 58
    140 56 50</EnergyDeviation>
  </VisualDescriptor>
</LowLevelQuery>
</BilVideoQuery>

```

Edge Histogram Feature Query: Edge Histogram Descriptor (EHD) is the most commonly used structure to represent any global feature composition of an image. It is invariant to image translation and rotation, and normalizing the histogram leads to scale invariance [22]. EHD captures spatial distribution of edges, like color layout descriptor. After partition of image into 16 non-overlapping blocks of equal size, edge information is calculated for each block in five edge categories: (i) vertical, (ii) horizontal, (iii) 45°, (iv) 135°, and nondirectional edge [18].

```

<BilVideoQuery outputType="[output_type]">
  <LowLevelQuery searchIn = "[Search_In_Type]">
    <VisualDescriptor xsi:type="EdgeHistogramType">
      <BinCounts>2 3 0 0 1 1 4 5 3 4 1 4 2 0 2 3 3 1 0 2 2 1 1 0
      0 6 0 0 1 3 0 1 0 0 0 2 0 2 0 0 3 1 0 0 1 4 1 6 1 5 0 2 0
      2 0 2 1 0 1 0 1 6 0 3 2 0 4 0 4 5 0 3 0 0 2 1 1 0 0 0
    </BinCounts>
  </VisualDescriptor>
</LowLevelQuery>
</BilVideoQuery>

```

Region Shape Feature Query: Region Shape Descriptor expresses pixel distributions within a 2-D object or region [15]. It employs a 2-D *Angular Radial Transformation* (ART), which is defined on a unit disk. For further information about the region shape descriptor used in MPEG-7, see [1].

```

<BilVideoQuery outputType="[output_type]">
  <LowLevelQuery searchIn = "[Search_In_Type]">
    <VisualDescriptor xsi:type="RegionShapeType">

```

```

    <MagnitudeOfART>3 5 2 5 6 2 5 3 6 2 3 4 5 6 2 3 3 5 3 2 2
    5 6 2 3 6 2 3 5 3 6 2 3 5 2</MagnitudeOfART>
  </VisualDescriptor>
</LowLevelQuery>
</BilVideoQuery>

```

Contour Shape Feature Query: Contour Shape Descriptor expresses object outline (contour) shape properties. The objects for which the characteristic shape features are contained in the contour are efficiently described by the contour shape descriptor [1]. Contour Shape Descriptor is robust to noise, scale, and orientation. For details about this descriptor, see [1].

```

<BilVideoQuery outputType="[output_type]">
  <LowLevelQuery searchIn = "[Search_In_Type]">
    <VisualDescriptor xsi:type="ContourShapeType">
      <GlobalCurvature>27 4</GlobalCurvature>
      <PrototypeCurvature>3 11</PrototypeCurvature>
      <HighestPeakY>56</HighestPeakY>
      <Peak peakX="16" peakY="2"/>
      <Peak peakX="4" peakY="2"/>
      <Peak peakX="56" peakY="5"/>
    </VisualDescriptor>
  </LowLevelQuery>
</BilVideoQuery>

```

3.1.6 Composite Query

BilVideo v2.0 has another important query type, named composite query. Composite queries allow the user to enter any number of above mentioned queries in any order and query the database. For composite queries, the user needs to decide the weights that will be used when merging the query results from all types. The syntax of the composite query is as follows:


```

<BilVideoQuery outputType = "[output_type]"
keywordQWeight="[weight]" temporalQWeight="[weight]"
spatialQWeight="[weight]" trajectoryQWeight="[weight]"
lowLevelQWeight="[weight]">
  <KeywordQuery>
    <FreeText>[keyword_search_text]</FreeText>
  </KeywordQuery>
  <TemporalQuery type="[temporal_relation_type]">
    <Object1>[name_of_first_object]</Object1>
    <Object2>[name_of_second_object]</Object2>
  </TemporalQuery>
  <SpatialQuery type="[spatial_relation_type]">
    <Object1>[name_of_first_object]</Object1>
    <Object2>[name_of_second_object]</Object2>
  </SpatialQuery>
  <LowLevelQuery searchIn = "[Search_In_Type]"
weight="[Low_Level_Feature_Query_Weight]">
    ...
  </LowLevelQuery>
  <TrajectoryQuery>
    <Params>
      <KeyTimePoint>
        <MediaRelIncrTimePoint>[t0]</MediaRelIncrTimePoint>
        <MediaRelIncrTimePoint>[t1]</MediaRelIncrTimePoint>
        <MediaRelIncrTimePoint>[t2]</MediaRelIncrTimePoint>
        <MediaRelIncrTimePoint>[t3]</MediaRelIncrTimePoint>
      </KeyTimePoint>
      <InterpolationFunctions>
        <KeyValue>[x0]</KeyValue>
        <KeyValue>[y0]</KeyValue>
        <KeyValue>[x1]</KeyValue>
        <KeyValue>[y1]</KeyValue>
        <KeyValue>[x2]</KeyValue>

```

```

    <KeyValue>[y2]</KeyValue>
    <KeyValue>[x3]</KeyValue>
    <KeyValue>[y3]</KeyValue>
  </InterpolationFunctions>
</Params>
</TrajectoryQuery>
</BilVideoQuery>

```

Each query type may exist 0 or more times in a composite query. Note that `lowLevelQueryWeight` is not the same as the weight in `<LowLevelQuery>` element. `lowLevelQueryWeight` in `<BilVideoQuery>` is the weight of the low level query among the other query types (i.e., keyword, temporal, etc.). On the other hand, the weight in `<LowLevelQuery>` is the weight of that feature among other features.

3.2 Query Samples

To have a solid idea about the supported queries in *BilVideo v2.0* and its capabilities, some sample queries and their specifications are given below.

Return videos where Blair appears

```

<BilVideoQuery outputType = "Video">
  <KeywordQuery>
    <FreeText>Blair</FreeText>
  </KeywordQuery>
</BilVideoQuery>

```

Return shots where “Blair and Clinton” or “Blair and Hillary and Cheryl” appears in the same frame.

```

<BilVideoQuery outputType = "Shot">
  <KeywordQuery>
    <FreeText>
      (Blair and (Clinton or (Hillary and Cheryl)))
    </FreeText>
  </KeywordQuery>
</BilVideoQuery>

```

Return key-segments where Blair appears while Clinton is appearing.

```

<BilVideoQuery outputType = "Keysegment">
  <TemporalQuery type="during">
    <Object1>Blair</Object1>
    <Object2>Clinton</Object2>
  </TemporalQuery>
</BilVideoQuery>

```

Return shots where Blair appears after Clinton.

```

<BilVideoQuery outputType = "Shot">
  <TemporalQuery type="after">
    <Object1>Blair</Object1>
    <Object2>Clinton</Object2>
  </TemporalQuery>
</BilVideoQuery>

```

Return shots where Blair appears on the left of Clinton in a frame.

```

<BilVideoQuery outputType = "Shot">
  <SpatialQuery type="left">
    <Object1>Blair</Object1>
    <Object2>Clinton</Object2>
  </SpatialQuery>
</BilVideoQuery>

```

Return videos where Golfer appears on the northwest (upperleft) of Golf-car in a frame.

```
<BilVideoQuery outputType = "Video">
  <SpatialQuery type="northWest">
    <Object1>Golfer</Object1>
    <Object2>Golf-car</Object2>
  </SpatialQuery>
</BilVideoQuery>
```

Return shots where an object moves in a similar trajectory to:

- (206, 99) at frame t,
- (122, 191) at frame t+5,
- (78, 283) at frame t+10, and
- (301, 325) t+15.

```
<BilVideoQuery outputType="Shot">
  <TrajectoryQuery>
    <Params>
      <KeyTimePoint>
        <MediaRelIncrTimePoint>0</MediaRelIncrTimePoint>
        <MediaRelIncrTimePoint>5</MediaRelIncrTimePoint>
        <MediaRelIncrTimePoint>10</MediaRelIncrTimePoint>
        <MediaRelIncrTimePoint>15</MediaRelIncrTimePoint>
      </KeyTimePoint>
      <InterpolationFunctions>
        <KeyValue>206</KeyValue>
        <KeyValue>99</KeyValue>
        <KeyValue>122</KeyValue>
        <KeyValue>191</KeyValue>
```

```

    <KeyValue>78</KeyValue>
    <KeyValue>283</KeyValue>
    <KeyValue>301</KeyValue>
    <KeyValue>325</KeyValue>
  </InterpolationFunctions>
</Params>
</TrajectoryQuery>
</BilVideoQuery>

```

Return shots [with the same spatial and keyword query weights] where Blair, Clinton and Hillary appears, and Blair is on the left of Clinton.

```

<BilVideoQuery outputType = "Shot" keywordQWeight = "1"
spatialQWeight="1">
<KeywordQuery>
<FreeText>Blair and Clinton and Hillary</FreeText>
  </KeywordQuery>
  <SpatialQuery type="left">
    <Object1>Blair</Object1>
    <Object2>Clinton</Object2>
  </SpatialQuery>
</BilVideoQuery>

```

Return shots [with the same spatial and keyword query weights] where Blair, Clinton and Hillary appears, and Blair is on the left of Clinton.

```

<BilVideoQuery outputType = "Shot" keywordQWeight = "1"
spatialQWeight="1">
<KeywordQuery>
<FreeText>Blair and Clinton and Hillary</FreeText>
  </KeywordQuery>
  <SpatialQuery type="left">
    <Object1>Blair</Object1>

```

```

    <Object2>Clinton</Object2>
  </SpatialQuery>
</BilVideoQuery>

```

Return videos where Golfer appears with weight of $(3/(1+3)) = 0.75$ and Color Structure of a moving region is as specified with weight of $((1/(1+3))(2/(1+2))) = 0.17$ and Color Layout of a frame is as specified with weight of $((1/(1+3)) * (1/(1+2))) = 0.08$.*

```

<BilVideoQuery outputType = "Video" lowLevelQWeight = "1"
keywordQWeight="3">
  <KeywordQuery>
  <FreeText>Golfer</FreeText>
  </KeywordQuery>
  <LowLevelQuery searchIn = "MovingRegion" weight="2">
    <VisualDescriptor colorQuant="2" xsi:type="ColorStructureType">
      <Values>0 85 2 0 0 0 0 0 0 0 255 69 12 8 0 0 0 0 0 0 0 0 0
      84 89 46 16 37 36 22 2 6 16 6 0 3 2 2 0 50 43 14 7 45 37 22
      2 24 25 12 4 5 7 2 0 27 34 38 16 7 7 2 2</Values>
    </VisualDescriptor>
  </LowLevelQuery>
  <LowLevelQuery searchIn = "WholeFrame" weight="1">
    <VisualDescriptor xsi:type="ColorLayoutType">
      <YDCCoeff>23</YDCCoeff>
      <CbDCCoeff>28</CbDCCoeff>
      <CrDCCoeff>32</CrDCCoeff>
      <YACCCoeff5>9 10 15 22 12</YACCCoeff5>
      <CbACCCoeff2>6 14</CbACCCoeff2>
      <CrACCCoeff2>18 22</CrACCCoeff2>
    </VisualDescriptor>
  </LowLevelQuery>
</BilVideoQuery>

```

Chapter 4

Query Processing

4.1 Software Architecture

BilVideo v2.0 queries are processed and results are generated by the *query processor*. The query processor is a multi-threaded server side component that listens a configured TCP port and processes the queries that are sent to it through the network. The components of the query processor and their responsibilities are as follows (see Figure 4.1):

1. *Query listener*: This module is responsible for listening the configured TCP port. When a client connects to the server, it generates a new thread to handle this client. Then, it sends each query from the connected client to the query parser module. It is also responsible for sending the result back to the requesting client.
2. *Query parser*: This module is responsible for parsing the queries received from clients. The queries are parsed according to the rules specified in the next section. In the final process of parsing, the module decomposes the query into sub-queries in case it may contain multiple queries (i.e., the query is a composite query). It generates a query structure and sends it to the result fusion module. Then, it sends each sub-query to its related query

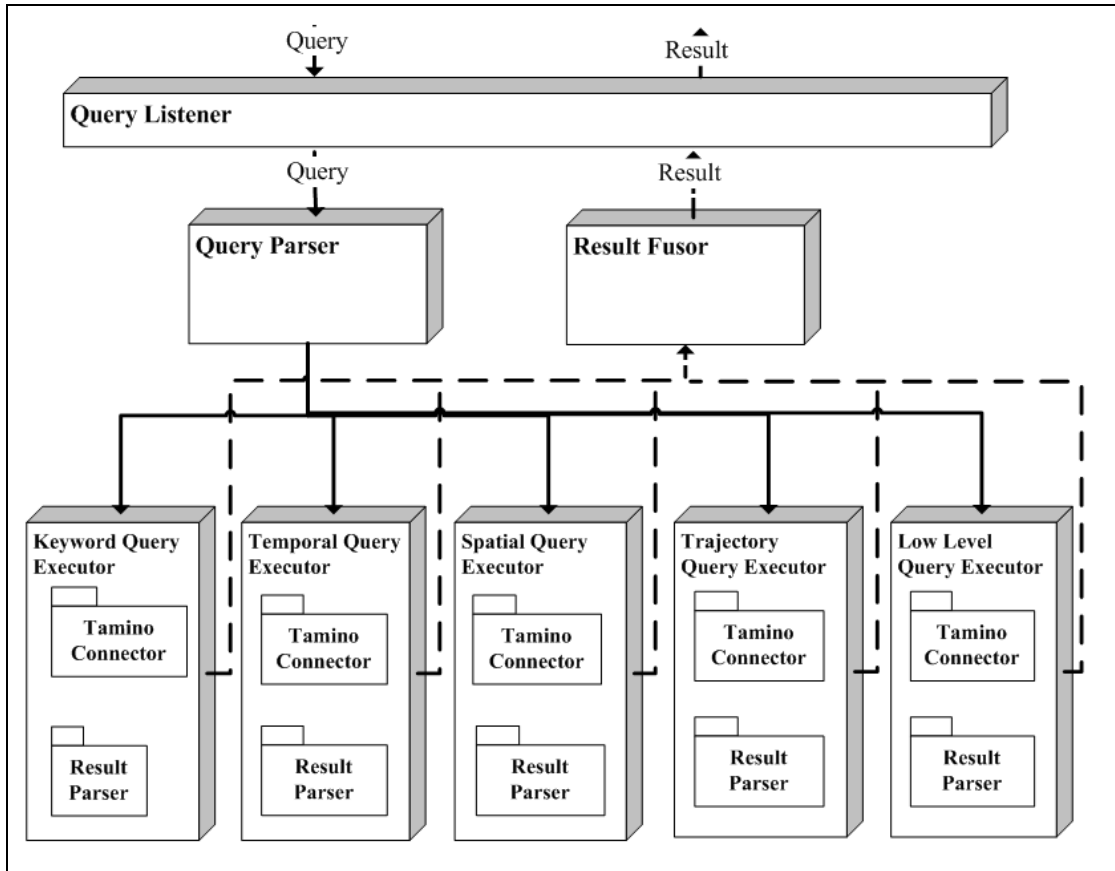


Figure 4.1: The software architecture of the query processor

execution module.

3. *Query executor*: This module has 5 sub-modules:

- the keyword query executor,
- the temporal query executor,
- the spatial query executor,
- the trajectory query executor, and
- the low level query executor.

Each sub-module is responsible for executing the sub-queries that they receive from the query parser according to the query type. The sub-queries are executed by performing one or more XQueries in XML database, and merging the results of these XQueries. After the sub-query result is generated, the module sends the result to the result fusion module.

4. *Result fusor*: It is responsible for the fusion of the query results. The results are merged according to the user-specified weights.

4.2 Query Parsing

Since BilVideo queries are structured as XML statements, they are parsed through a widely used XML parser library for C++, named Xerces. Using the library, the XML structures are parsed and query properties are extracted. As a result of this extraction, `SQuery` and `SPartialQuery` structures are generated. The properties of `SQuery` and `SPartialQuery` are as follows:

```
struct SQuery
{
    id of the query
    query output type (video, shot or key-segment)
    requesting client
    number of subqueries
    arrival time of the query
    execution completion time
    query data
    results of the query
    weights of query types
}

struct SPartialQuery
{
    id of parent query
    query output type (video, shot or key-segment)
    type of the query (keyword, spatial, etc.)
    searchInType
    name of first object
    name of second object
}
```

```

    freetext
    subquery type (before, after, west, dominant color, etc.)
    feature list
}

```

SQuery structure contains general query properties. For each received query the parser generates a unique ID and assigns it. Result parser also timestamps the query when it arrives. Hence, the query processing time can be calculated when the execution is completed. “Number of subqueries” and “weights of the query types” properties are needed by the result fusion module to combine subquery results. After construction of **SQuery**, the structure is passed to the result fusion module for later use in merging of subquery results.

Partial query structure contains data needed to perform the query by any type of query executor.

- “searchInType” and “feature for list” properties are used by low level query execution module.
- “name of first object” and “name of second object” properties are used by temporal (spatial) query execution modules to identify the objects that have temporal (spatial) relation.
- “free-text” property is used by keyword query execution module as the free-text keyword(s); e.g., “((blair and hillary) or (clinton and cheryl))”.
- “subquery” type is used by all query execution modules.

After the construction of **Partial** query structures, each subquery structure is sent to its related execution module that is defined in “type of query property” for further processing.

4.3 Query Execution

Query execution takes place in 5 self-threaded sub-modules that are specific to the kind of the query. These 5 sub-modules share the same infrastructure for performing common tasks. These common tasks include Tamino connectivity, which is for connecting to Tamino database and sending XQueries to Tamino, and result parsing, which is for parsing the result obtained from Tamino XML database. The components to perform these duties are Tamino connector component and result parsing component, respectively.

1. *Tamino connector component* is responsible for connecting to Software AG Tamino XML database using Tamino API for C and sending XQuery statement for querying descriptors of videos. In case of replacement of XML database used in the system, this component could be replaced by another component with the same interface.
2. *Result parsing component* is responsible for parsing the result from XML database. Since the results of XML database are again in XML structured form, they require further processing to get ready for using. Result parsing component parses a result from Tamino and generates a structure called `TaminoResult`. By doing so, it enables query executors to use the results from XML database.

The properties of `TaminoResult` structure are as follows:

```
struct STaminoResult
{
    name of the video
    id of the shot
    output starting frame
    output end frame
    actual starting frame
    actual end frame
}
```

```

    positions for trajectory query
    feature vector for low level query
}

```

In this structure, (actual starting frame, output starting frame) and (actual end frame, output end frame) pairs draw attention. The ones that start with output means the results that the user will get as the final result, on the other hand the ones that start with actual means the exact start and end frames that satisfy the condition of the query. For instance, if a query condition is satisfied in the interval of [345, 900] in a video with 1000 frames, and if the output type is video, then, the properties would be as follows:

- *actual starting frame:* 345
- *actual end frame:* 900
- *output starting frame:* 0
- *output end frame:* 1000

As it is seen from the example, the output start and end frames change according to the query output type (i.e., video, shot or key-segment).

4.3.1 Keyword Query Execution

Keyword query execution takes place in the sub-module keyword query executor. First, the free text in `partial query` structure is parsed and an XQuery statement is generated according to this query text. Then, this XQuery statement is executed using the Tamino connector component. Finally, the Tamino result is parsed and a new structure `SPartialQueryResult` is sent to the result fusion module. The properties of `SPartialQueryResult` are given below:

```

struct SPartialQueryResult
{
    id of the query
    output type of the query
    type of the query
    type of the subquery
    list of tamino results
}

```

In `SPartialQueryResult` structure, the most important property is the list of Tamino results. This list contains the results that satisfy the query condition.

4.3.2 Temporal Query Execution

Temporal query execution is performed in temporal query execution module. The major difference between the execution of a temporal query and a keyword query is that in a temporal query there are two XQueries, which are sent to the XML database. One query is used for finding the temporal existence of the first object and the second query is for the second object. The results returned from Tamino for these two queries are merged according to the rule of temporal relations. For instance, for the equal relation, the results from database are compared whether they exist in the same time interval. The remaining execution is the same as keyword query execution.

The merge rules for temporal queries according to the subquery type are given below (Notation: $Start_1$ means start of temporal existence of the first object):

VideoName₁ = VideoName₂ AND ShotName₁ = ShotName₂ AND
 KeySegmentName₁ = KeySegmentName₂ AND

- *before*: End₁ < Start₂
- *after*: Start₁ > End₂

- *equal*: $\text{Start}_1 = \text{Start}_2 \text{ AND } \text{End}_1 = \text{End}_2$
- *notEqual*: $\text{End}_1 < \text{Start}_2 \text{ OR } \text{End}_2 < \text{Start}_1$
- *during*: $\text{Start}_1 > \text{Start}_2 \text{ AND } \text{End}_1 < \text{End}_2$
- *contains*: $\text{Start}_1 < \text{Start}_2 \text{ AND } \text{End}_1 > \text{End}_2$
- *overlaps*: $\text{Start}_1 < \text{Start}_2 \text{ AND } \text{End}_1 > \text{Start}_2 \text{ AND } \text{End}_1 < \text{End}_2$
- *overlappedBy*: $\text{Start}_2 < \text{Start}_1 \text{ AND } \text{End}_2 > \text{Start}_1 \text{ AND } \text{End}_2 < \text{End}_1$
- *meets*: $\text{Start}_1 < \text{Start}_2 \text{ AND } \text{End}_1 > \text{Start}_2$
- *metBy*: $\text{Start}_2 < \text{Start}_1 \text{ AND } \text{End}_2 > \text{Start}_1$
- *starts*: $\text{Start}_1 = \text{Start}_2$
- *finishes*: $\text{End}_1 = \text{End}_2$

4.3.3 Spatial Query Execution

Spatial query execution is performed in spatial query execution module. Like temporal query execution, two XQueries are needed to find spatial locations of two objects in the database. The query results from database are merged according to the rule of spatial relations. The important point in merging is not only the spatial locations should fit into spatial relation rule, but also the frame that the rule is satisfied should be the same for two objects. After the merge operation, the partial query result is sent to the result fusion module.

The merge rules for spatial queries according to the subquery type are given below (Notation: X_{11} means X_1 of the first object, X_{21} means X_2 of the first object):

$\text{VideoName}_1 = \text{VideoName}_2 \text{ AND } \text{ShotName}_1 = \text{ShotName}_2 \text{ AND}$
 $\text{KeySegmentName}_1 = \text{KeySegmentName}_2 \text{ AND}$
 $\text{FrameNumber}_1 = \text{FrameNumber}_2 \text{ AND}$

- *west*: $x_{11} \leq x_{12}$ AND $y_{11} < y_{22}$ AND $y_{21} > y_{12}$
- *east*: $x_{11} \geq x_{12}$ AND $y_{11} < y_{22}$ AND $y_{21} > y_{12}$
- *north*: $y_{11} \leq y_{12}$ AND $x_{11} < x_{22}$ AND $x_{21} > x_{12}$
- *south*: $y_{11} \geq y_{12}$ AND $x_{11} < x_{22}$ AND $x_{21} > x_{12}$
- *northWest*: $y_{11} < y_{12}$ AND $y_{21} < y_{22}$ AND $x_{11} < x_{12}$ AND $x_{21} < x_{22}$
- *northEast*: $y_{11} < y_{12}$ AND $y_{21} < y_{22}$ AND $x_{11} > x_{12}$ AND $x_{21} > x_{22}$
- *southWest*: $y_{11} > y_{12}$ AND $y_{21} > y_{22}$ AND $x_{11} < x_{12}$ AND $x_{21} < x_{22}$
- *southEast*: $y_{11} > y_{12}$ AND $y_{21} > y_{22}$ AND $x_{11} > x_{12}$ AND $x_{21} > x_{22}$
- *left*: $x_{11} \leq x_{12}$ AND $y_{11} < y_{22}$ AND $y_{21} > y_{12}$
- *right*: $x_{11} \geq x_{12}$ AND $y_{11} < y_{22}$ AND $y_{21} > y_{12}$
- *below*: $y_{11} \geq y_{12}$ AND $x_{11} < x_{22}$ AND $x_{21} > x_{12}$
- *above*: $y_{11} \leq y_{12}$ AND $x_{11} < x_{22}$ AND $x_{21} > x_{12}$

4.3.4 Trajectory Query Execution

This query execution procedure is completely different from the previously mentioned execution processes. In this execution, first of all, motion trajectories of all moving objects, which are stored in the XML database, are retrieved. Then, the positional differences of the objects from the query and database are compared according to the Equation 4.1, which makes use of the position similarity calculation of the motion trajectory similarity equation, as described in [11].

$$M_P(T1, T2) = \frac{\sum_{i=1}^n \sqrt{(x_{1i} - x_{2i})^2 + (y_{1i} - y_{2i})^2}}{n} \quad (4.1)$$

where $M_P(T1, T2)$ is the linear weighting of two object positions, i is the index of the trajectory sample point and n is the number of trajectory sample points used in comparison.

After the calculation of the trajectory distances, they are normalized using the diagonal length of the video images, and all the distances are mapped to the $[0,1]$ interval and the trajectory ranks are calculated using the following formula:

$$rank_i = (1 - \frac{d_{iq}}{d_{diag}}),$$

where i is the index of the moving region and d_{iq} is the distance between the trajectory of i^{th} moving region and the trajectory query. d_{diag} is the maximum diagonal length of a frame in the database. After the ranks are calculated, the results are sorted according to the rank and the ones whose normalized distances are smaller than 0.5 are passed to the result fusion module.

4.3.5 Low Level Feature Query Execution

Low level query execution is performed in low level query execution module. This execution starts in a similar fashion with trajectory query execution. All the feature vectors are retrieved from the database according to the search in type. For instance: if search in type is still region and low level query type is scalable color, then the scalable color features of all still regions in the database are retrieved. Then, the distances between these feature vectors and query feature vector are calculated. The retrieved distances are normalized using the maximum of distances of that low level feature. In this way, the distances are mapped within the interval $[0,1]$. Then, ranks of the still regions are calculated using the following formula:

$$rank_i = (1 - \frac{d_{iq}}{d_{max}}) \times weight,$$

where i is the index of the still region and d_{iq} is the distance between the i^{th} still region feature vector and the query feature vector. The d_{max} is calculated

Low level feature	Maximum distance
Color structure	6701
Scalable color	1549
Color layout	571
Edge histogram	423
Homogeneous texture	12403

Table 4.1: The maximum distances for the keyframes in TRECVID 2008 Development Data

by comparing each pair of keyframes from TRECVID 2008 development data. The list of maximum distance values obtained from this collection are given in Table 4.1. After the ranks are calculated, the results are sorted and the ones whose normalized distances are smaller than 0.5 are passed to the result fusion module.

4.4 Result Fusion

Result fusion is the final step of query processing and is performed by result fusion module, which is a self-threaded module. Result fusion module receives `SQuery` structure from the query parsing module. In this structure query id and number of subqueries of that query are specified. The result fusion module waits for `SPartialQueryResult` structures that have the same parent query id with `SQuery` structure. When all partial results for this `SQuery` are received, the result fusion module starts the fusion process. In fusion process, all partial query results are ranked using the weights for query types. We have described the method of calculating low level query ranks before. The ranks given there are different from the fusion ranks. The fusion rank calculation is done in the following way:

$$\text{rank_of_query_result}_i = \sum_{j=1}^{n_s} r_j, \quad (4.2)$$

where n_s is the number of subqueries,

$$r_j = \begin{cases} 1 \times \text{weight}_{\text{temporal_query}} & \text{if } sqt \text{ is temporal query} \\ 1 \times \text{weight}_{\text{spatial_query}} & \text{if } sqt \text{ is spatial query} \\ \text{low_level_rank}_j \times \text{weight}_{\text{low_level_query}} & \text{if } sqt \text{ is low level query} \\ \text{trajectory_rank}_j \times \text{weight}_{\text{trajectory_query}} & \text{if } sqt \text{ is trajectory query} \end{cases}$$

where sqt is the subquery type. After the ranks are calculated according to Equation 4.2, the results are sorted and top 10 results are sent to the client.

Chapter 5

User Interface and System Performance

In the first part of this chapter, the visual query interface and its architecture are introduced. Then, some sample queries generated through the visual query interface and their results are presented. Finally, some sample query process durations are provided to give an idea about the performance of *BilVideo v2.0*

5.1 Visual Query Interface

A visual query interface has been developed for BilVideo v2.0 to simplify the query specification for the users who are not familiar with XML and MPEG-7.

The visual query interface is implemented using WxWidgets library for C++. WxWidgets is a cross-platform C++ framework providing GUI and other facilities. It supports MS Windows, UNIX with GTK+, UNIX with Motif and MacOS. It was developed at University of Edinburgh and made publicly available [19].

Visual Query Interface also makes use of the well known computer vision library OpenCV for image processing, some parts of XM Software, which is the reference software of MPEG-7 for feature extraction and comparison, and JSegLib

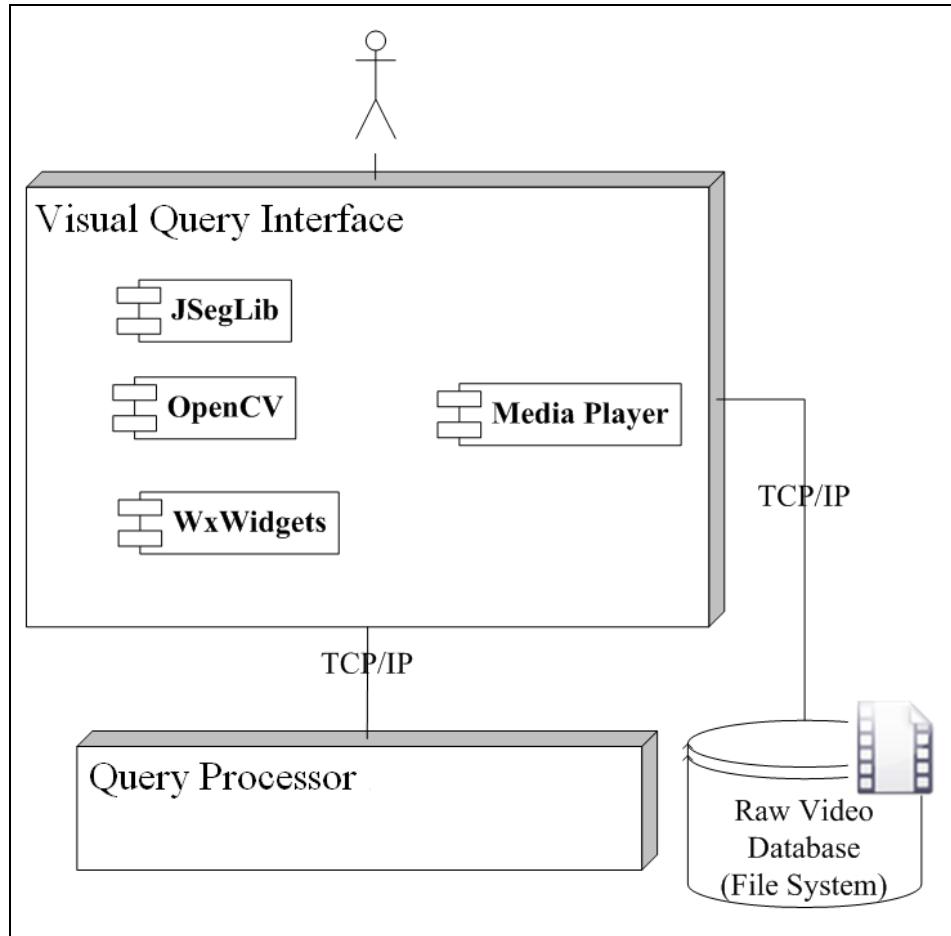


Figure 5.1: The architecture of visual query interface

for image segmentation. The architecture and interfaces of Visual Query Interface is depicted in Figure 5.1.

A user can query the system using:

- keywords (in keyword query, temporal query and spatial query),
- sketches (in temporal query, spatial query and trajectory query), and
- input images (in low level query).

When the input images are used, the user can use the whole input image or a part of the image. JSegLib comes to play when the user wants to use a part of the

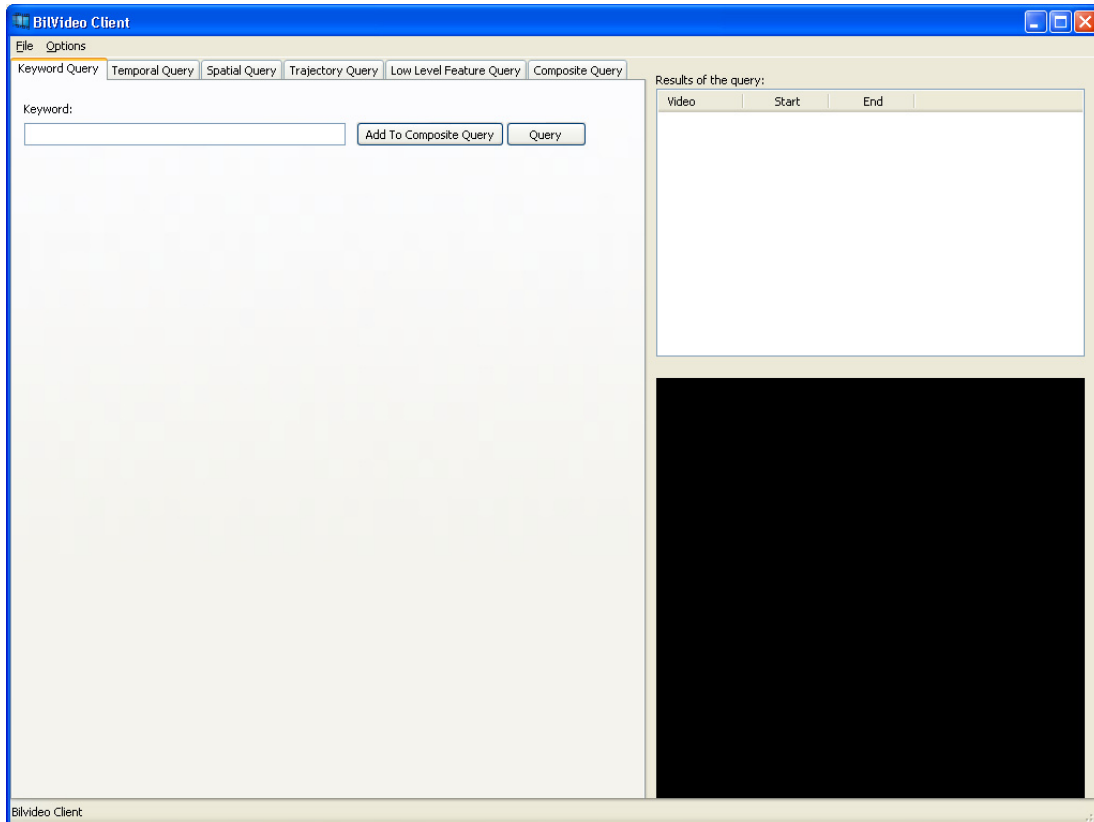


Figure 5.2: The top level window of the visual query interface

query image. A query image is segmented when the user wants to automatically segment the image and wants to use one or more segments as the query.

5.1.1 General View

The visual query interface is a form-based application that connects to the query processor to send queries and retrieve the results (see Figure 5.2). When the visual query interface is started, the user can see a tabbed page on the left, the query result panel on the upper right corner, and a media player on the lower left part of the form. The tabbed page contains the interfaces for the keyword query, the temporal query, the spatial query, the trajectory query, the low level query, and finally, the composite query.

The results of the query are shown in the results list panel. When the user

selects an item from this list, and double clicks on it, the result video or video part is played in the media player.

Additionally, there are two menus (the file menu and the options menu) in the form. From the file menu, the user can open the Video TOC (Table of Contents), from where the videos, shots, still regions and moving regions in the database can be viewed (see Figure 5.3). From the options menu, the user can select the output type of the query (see Figure 5.4).

5.1.2 Keyword Query Interface

This interface is used to prepare the keyword queries (Figure 5.5). There is only a single textbox to prompt the user to enter the query words. The “Query” button, which is common in all query interfaces, are used to query the database. The “Add to Composite Query” button, which is again common in all interfaces, except the composite query interface, is used to add the query to the composite query.

5.1.3 Temporal Query Interface

This interface is used to prepare temporal queries (Figure 5.6). Temporal query can be prepared either by keywords or by drawing sketches. When using query by keyword, the user specifies two objects and their temporal relationship by choosing the relationship from a drop down menu. When querying-by-sketch, the user defines the relationship by drawing a sketch in a time-line.

5.1.4 Spatial Query Interface

This interface is used to prepare spatial queries (Figure 5.7). Like a temporal query, user can specify the queries by keyword or by sketch. In query-by-keyword, the user writes the name of the objects and selects the spatial relationship type

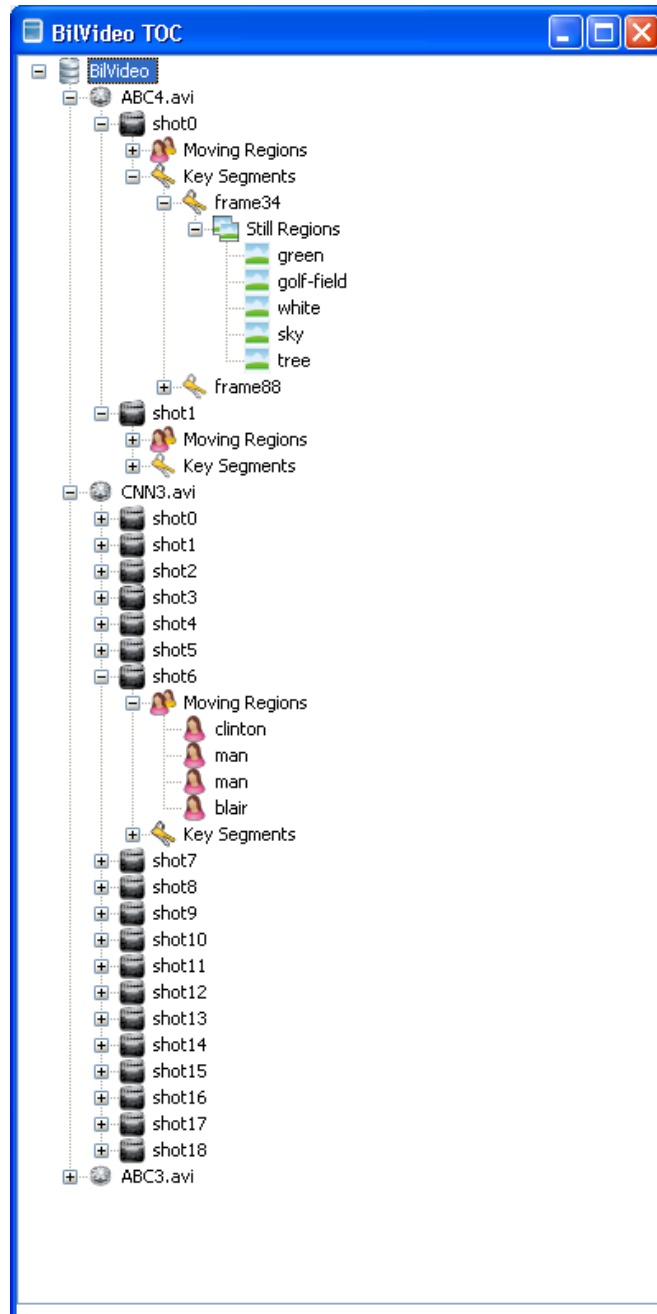


Figure 5.3: The video table of contents

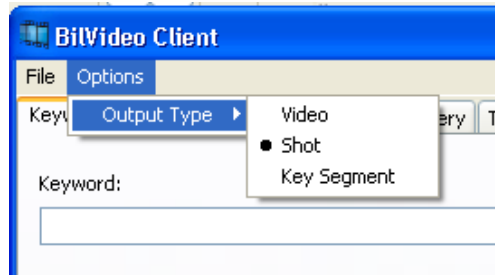


Figure 5.4: The options menu: the user selects the output type of the query

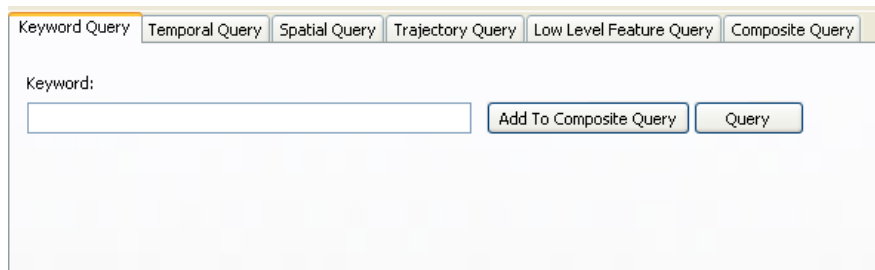


Figure 5.5: The keyword query interface

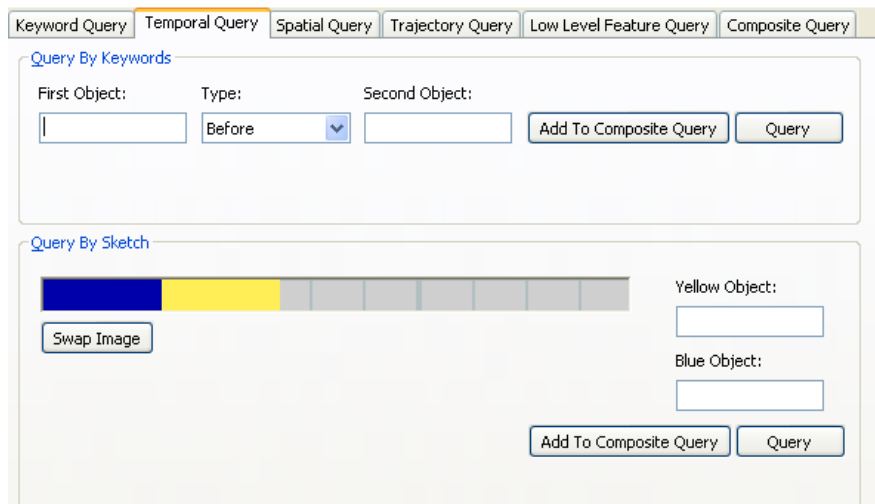


Figure 5.6: The temporal query interface

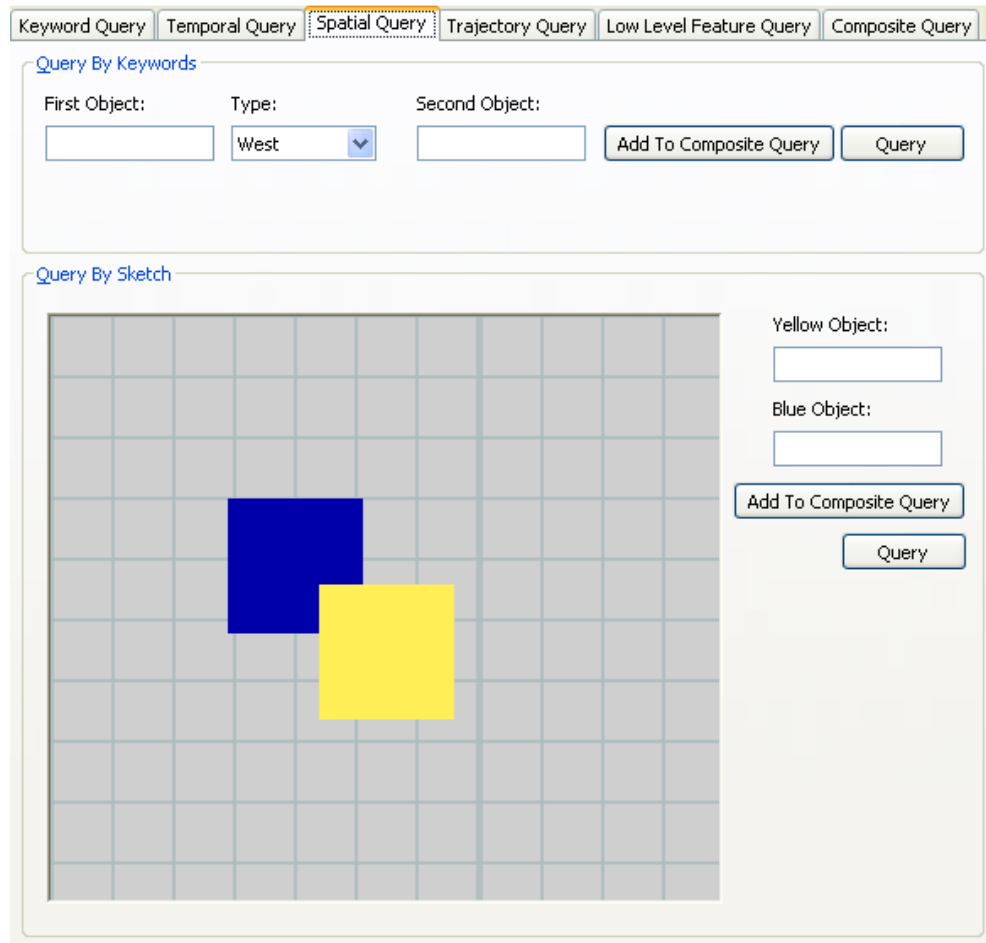


Figure 5.7: The spatial query interface

from a drop down. In query-by-sketch, the user defines the relationship type by drawing a sketch of the relationship in a canvas.

5.1.5 Trajectory Query Interface

Using this interface, the user can specify trajectory queries (see Figure 5.8). In this interface, there is a canvas, where users can insert trajectory path mark by right clicking. Users can also change the location of a specified path mark by dragging it. The inserted path marks are used as the query trajectory and the moving regions that moved in this trajectory are retrieved from the database.

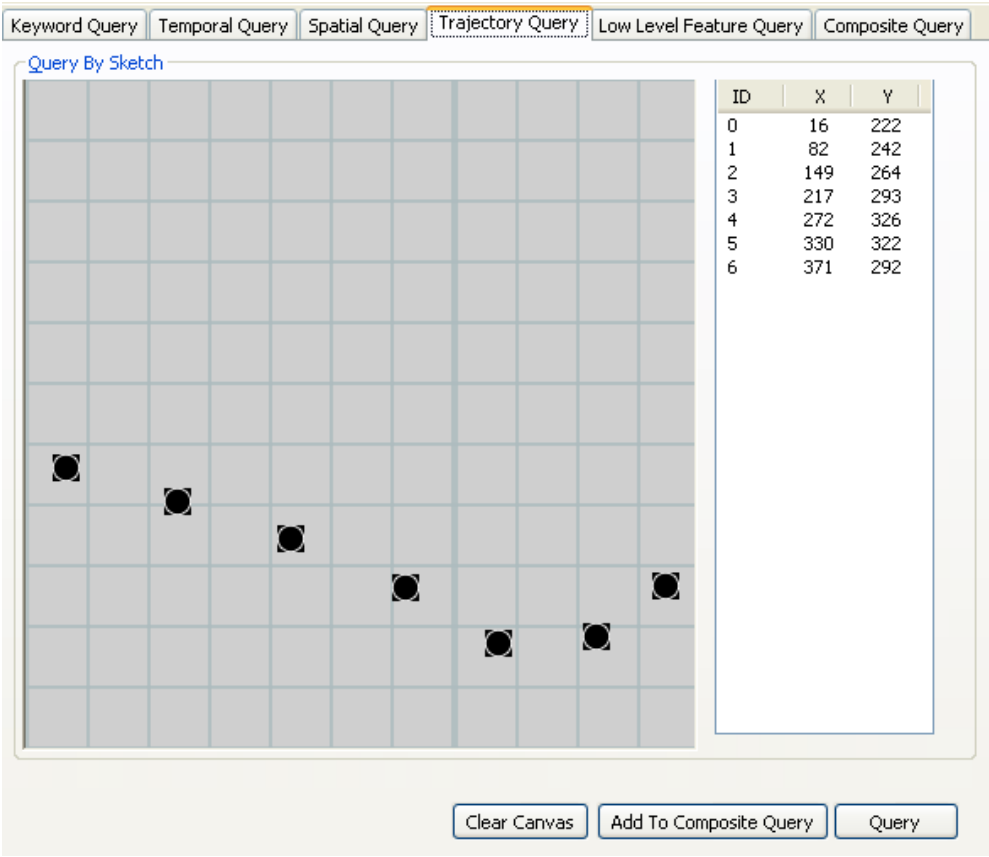


Figure 5.8: The trajectory query interface

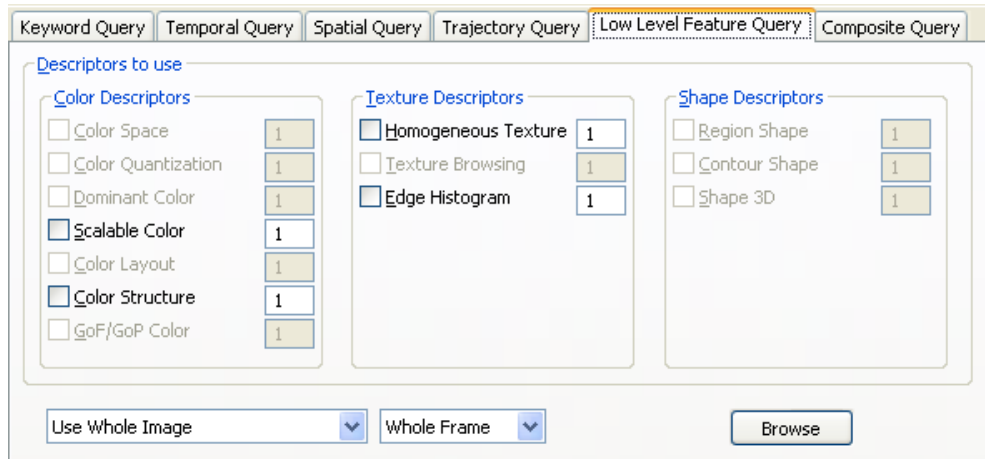


Figure 5.9: The low level query interface

5.1.6 Low Level Query Interface

The low level query interface is the most complicated interface. We explain the low level query specification step by step in the sequel.

Figure 5.9 is the general view of the low level query interface. From the check boxes in the upper part, the user can select the low level descriptors that will be used in the query. Then, from the first drop down menu, the user chooses the part of the query image that will be used. The user can select the whole image or any part of the image. If the whole image is selected, the low level features of the query image are extracted. If the user wants to use a part of the image, then the image part can be selected manually or with the help of automatic segmentation. Another drop down menu is used to specify the features to search in the database. The user can search the low level features within shots, moving regions, still regions, or frames. After these drop down menu selections, the user browses for an image file, which will be used as the query image.

In the case of segmentation, the user has to select the segments or parts of the image as query image, as shown in Figure 5.10.



Figure 5.10: The result of the segmentation process

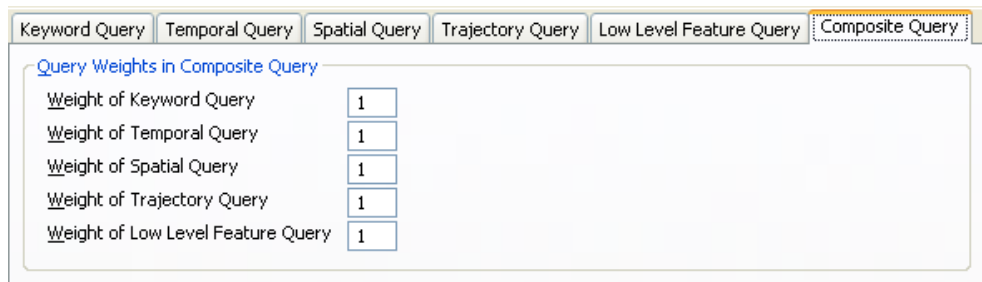


Figure 5.11: The composite query interface

5.1.7 Composite Query Interface

This interface is used for preparing composite queries, in fact for defining the weights of query types that will be used in composite query (see Figure 5.11). The query statements are defined in the related query interface and inserted to the composite query statement by pressing “Add to Composite Query” button.

5.2 Sample Queries

In this section, examples of the query results displayed for different query types are presented.

Keyword Query

INPUT: (blair and clinton) or (hillary and cheryl)

OUTPUT:

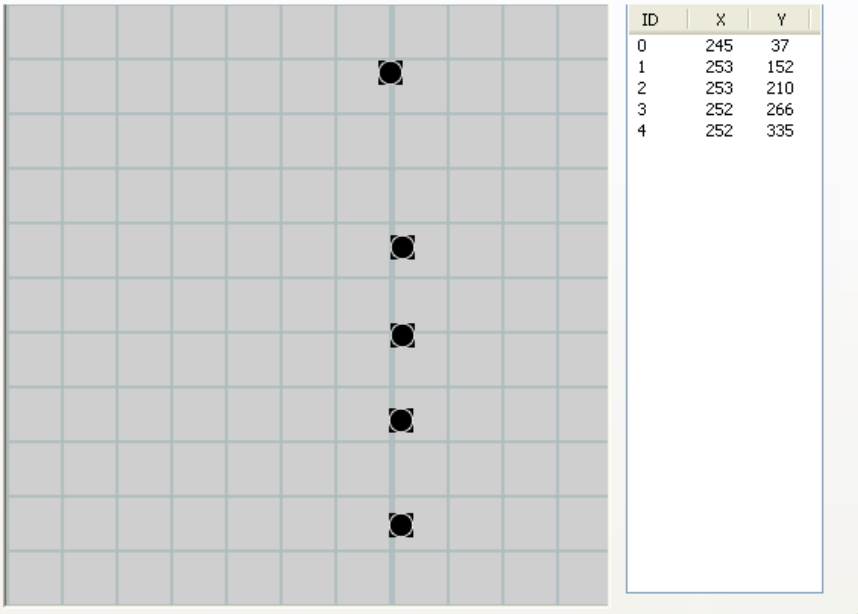


*Temporal Query***INPUT:** golfer equal golf-car**OUTPUT:***Spatial Query***INPUT:** blair right clinton**OUTPUT:**

Trajectory Query

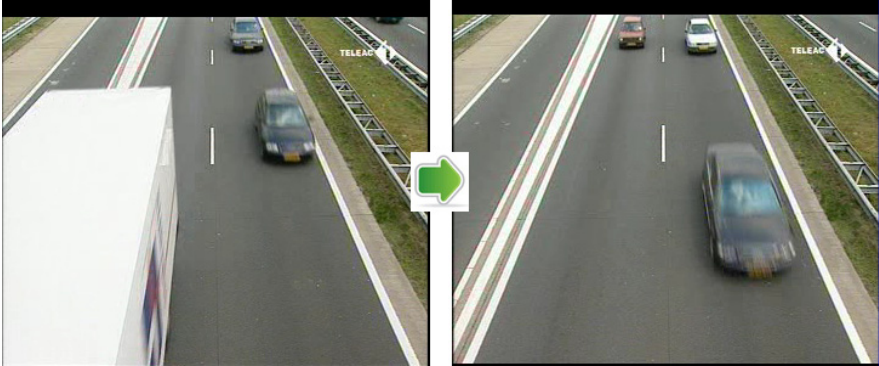
INPUT:

Query By Sketch



ID	X	Y
0	245	37
1	253	152
2	253	210
3	252	266
4	252	335

OUTPUT:



Low Level Query

INPUT: Scalable color of the following input image



OUTPUT:



*Composite Query***INPUT:** blair right clinton + blair equal clinton**OUTPUT:**

As it is seen from the output of the composite query, since the first two video parts (upper-left and upper-right) satisfy both conditions, they are shown with higher rank than the other video parts, which only satisfy the second query.

5.3 Performance

To evaluate the query processing performance, query execution times for different query types are calculated. When a query arrives at the query processor, it is timestamped. After producing the result of the query, the query is timestamped again, and the difference between these two timestamps are calculated.

In Table 5.1, the execution times for different types of queries are given in milliseconds. These values are obtained by performing the queries 25 times and taking the average of these 25 values. During the evaluation process, the database contains three videos which contain 5046 frames in total.

Query type	Content	Execution times (msec.)
Keyword query	-	54
Temporal query	-	135
Spatial query	-	124
Trajectory query	-	64
Low level query	1 feature	52
Low level query	2 features	190
Low level query	3 features	343
Composite query	2 keyword	87
Composite query	2 temporal	220
Composite query	2 spatial	224
Composite query	2 trajectory	207
Composite query	3 keyword	111
Composite query	3 temporal	305
Composite query	3 spatial	294
Composite query	3 trajectory	362
Composite query	keyword + temporal	128
Composite query	keyword + spatial	132
Composite query	keyword + trajectory	84
Composite query	keyword + low level	78
Composite query	temporal + spatial	298
Composite query	temporal + trajectory	129
Composite query	temporal + low level	121
Composite query	spatial + trajectory	145
Composite query	spatial + low level	132
Composite query	keyword + temporal + spatial	229
Composite query	temporal + spatial + low level	234

Table 5.1: The query execution times

As it is seen in Table 5.1, the spatial and temporal queries are more costly than the other query types. The reason for this difference is the 2 XQuery-Per-Query requirement of these two query types. For the composite query and the low-level query with more than one feature, it can be observed that the number of subqueries does not affect the processing time. This is achieved by using multi-threading.

Chapter 6

Conclusion and Future Work

In this thesis, query processing component of the *BilVideo v2.0* system is presented. *BilVideo v2.0* has been planned as a full-fledged multimedia database management system. Currently, it supports spatio-temporal, keyword, low-level feature (color, shape, texture), trajectory queries and composite queries for videos. *BilVideo v2.0* has its own XML-based query language. The query processing subsystem interprets the queries and generates XQuery statements. Using XQuery, the query processor queries the native XML Database (NXD), named Tamino, which stores the descriptions of videos in XMLs that are compatible with MPEG-7. Then, the results obtained from Tamino are processed further by the query processor to produce the final query result.

Keyword queries, spatial queries and trajectory queries are fully supported by the system. However, there may be some extensions to these queries like addition of new relations, addition of advanced keyword queries, etc. For the trajectory query case, the query trajectory is interpolated to fit the trajectories in the database. Some other approaches can be tried to allow retrieval of more accurate results. For instance, querying sub-trajectories may be one of these approaches. Another approach may be allowing the user to give the name of the object, whose trajectory is going to be queried.

For low level queries, the descriptors defined by MPEG-7 standard are queried.

Low level queries require the libraries from the video annotation tool. The development of the video annotation tool of *BilVideo v2.0* is ongoing. After its finalization, the low level queries that are not implemented or supported currently will also be implemented. By this way, the video part of the system would be completed.

Moreover, the composite query allows to query the database using any number of query types in the same query. Composite query allows definition of query weights that will be used by the query processor. By using composite query, users can narrow the result set that will be returned by the system.

BilVideo v2.0 is an evolving project and will serve as a framework for more researches in the multimedia area. Researchers who are interested in content based retrieval can use *BilVideo v2.0* to compare their results when proposing new distance calculations for MPEG-7 low level features. Moreover, it can be used for testing new features that are generated by extending MPEG-7 with DDL rules. On the other hand, some researches on query weighting can be conducted to find an optimum recall/precision. And some additional MPEG-7 related research works can be performed using *BilVideo v2.0*.

With its current version, *BilVideo v2.0* is a full-fledged video database system. In the near future, with the addition of audio and image querying support, *BilVideo v2.0* would achieve its mission of being a full-fledged multimedia database system. In addition to being a full-fledged video database system, additional innovative characteristics of *BilVideo v2.0* are the description model relying on XML-based MPEG-7 standard, a special simplified query language, and a query processor for MPEG-7 which supports composite queries.

Bibliography

- [1] M. Bober. Mpeg-7 visual shape descriptors. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(6):716–719, 2001.
- [2] S.-F. Chang, W. Chen, H. J. Meng, H. Sundaram, and D. Zhong. VideoQ: an automated content based video search system using visual cues. In *Proceedings of the Fifth ACM International Conference on Multimedia*, pages 313–324, 1997.
- [3] M. G. Christel. Carnegie Mellon University Traditional Informedia Digital Video Retrieval System. In *CIVR '07: Proceedings of the 6th ACM International Conference on Image and Video Retrieval*, pages 647–647, 2007.
- [4] M. G. Christel, J. Richardson, and H. D. Wactlar. Facilitating access to large digital oral history archives through informedia technologies. In *JCDL '06: Proceedings of the 6th ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 194–195, 2006.
- [5] M. Doller and H. Kosch. The MPEG-7 Multimedia Database System (MPEG-7 MMDB). *Journal of Systems and Software*, 81(9):1559–1580, 2008.
- [6] M. E. Dönderler, Özgür Ulusoy, and U. Güdükbay. A rule-based approach to represent spatio-temporal relations in video data. In *ADVIS '00: Proceedings of the First International Conference on Advances in Information Systems*, pages 409–418. Springer-Verlag, 2000.
- [7] M. E. Dönderler, Özgür Ulusoy, and U. Güdükbay. A rule-based video database system architecture. *Information Sciences*, 143(1-4):13–45, 2002.

- [8] M. E. Dönderler, Özgür Ulusoy, and U. Gündükbay. Rule-based spatiotemporal query processing for video databases. *The VLDB Journal*, 13(1):86–103, 2004.
- [9] A. Hauptmann, D. Ng, R. Baron, M.-Y. Chen, M. Christel, P. Duygulu, C. Huang, W.-H. Lin, H. Wactlar, N. Moraveji, N. Papernick, C. Snoek, G. Tzanetakis, J. Yang, R. Yan, and R. Jin. Informedia at TRECVID 2003: Analyzing and Searching Broadcast News Video. In *Proceedings of (VIDEO) TREC 2003 (Twelfth Text Retrieval Conference)*, November 2003.
- [10] T. Huang, S. Mehrotra, and K. Ramchandran. Multimedia Analysis and Retrieval System (MARS) Project. In *Proceedings of 33rd Annual Clinic on Library Application of Data Processing - Digital Image Access and Retrieval*, 1996.
- [11] S. Jeannin and A. Divakaran. MPEG-7 Visual Motion Descriptors. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(6):720–724, 2001.
- [12] E. Kasutani and A. Yamada. The MPEG-7 Color Layout Descriptor: a compact image feature description for high-speed image/video segment retrieval. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, volume 1, pages I–674 – I–677, 2001.
- [13] J.-H. Lee, H.-J. Kim, and W.-Y. Kim. Video/image retrieval system based on MPEG-7 (VIRS). In *Proceedings of the International Conference on Information Technology: Research and Education (ITRE2003)*, pages 79–83, 2003.
- [14] B. Manjunath, J.-R. Ohm, V. Vasudevan, and A. Yamada. Color and texture descriptors. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(6):703–715, 2001.
- [15] B. S. Manjunath, P. Salembier, and T. Sikora, editors. *Introduction to MPEG-7 Multimedia Content Description Interface*. Wiley, England, 2002.

- [16] G. Pavlović-Lažetić. Native XML databases vs. relational databases in dealing with XML documents. *Kragujevac Journal of Mathematics*, 30:181–199, 2007.
- [17] Y. M. Ro, M. Kim, H. K. Kang, B. Manjunath, and J. Kim. MPEG-7 Homogeneous Texture Descriptor. *Electronics and Telecommunications Research Institute (ETRI) Journal*, 23(2):41–51, 2001.
- [18] T. Sikora. The MPEG-7 Visual Standard for Content Description-An Overview. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(6):696–702, 2001.
- [19] J. Smart, R. Roebling, V. Zeitlin, and R. Dunn et al. *wxWidgets 2.8.6: A portable C++ and Python GUI toolkit*. <http://docs.wxwidgets.org/2.8.6/>, 2007.
- [20] Software AG. *Tamino XML Server Documentation*, 4.4.1 edition, 2006.
- [21] M. Stonebraker, L. A. Rowe, and M. Hirohama. The implementation of POSTGRES. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):125–142, 1990.
- [22] C. S. Won, D. K. Park, and S.-J. Park. Efficient Use of MPEG-7 Edge Histogram Descriptor. *Electronics and Telecommunications Research Institute (ETRI) Journal*, 24(1):23–30, 2002.
- [23] K.-M. Wong, L.-M. Po, and K.-W. Cheung. Dominant color structure descriptor for image retrieval. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, volume 6, pages VI–365 – VI–368, 2007.
- [24] World Wide Web Consortium. *Extensible Markup Language (XML)*. <http://www.w3.org/XML/>, 2003.
- [25] World Wide Web Consortium. *XQuery 1.0: An XML Query Language*. <http://www.w3.org/TR/xquery/>, 2007.