SEMANTIC QUERY EXECUTION IN A VIDEO DATABASE SYSTEM

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

Cemil ALPER

August, 2004

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.

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

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

Prof. Dr. Enis Çetin

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. Varol Akman

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

Assist. Prof. Dr. Selim Aksoy

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute Engineering and Science

ABSTRACT

SEMANTIC QUERY EXECUTION IN A VIDEO DATABASE SYSTEM

Cemil ALPER
M.S. in Computer Engineering
Supervisors: Prof. Dr. Özgür Ulusoy and
Assist. Prof. Dr. Uğur Güdükbay
August, 2004

In this thesis, we have extended a video database management system, called BilVideo, with semantic querying capability. Our work is based on a video data model for the extraction and storage of the semantic contents of videos, and a query language to support semantic queries on video data. The Web based query interface of BilVideo has also been modified to handle semantic queries both visually and textually.

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

ÖZET

BİR VİDEO VERİTABANI SİSTEMİNDE ANLAMSAL SORGULARIN ÇALIŞTIRILMASI

Cemil ALPER
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticileri: Prof. Dr. Özgür Ulusoy ve
Assist. Prof. Dr. Uğur Güdükbay
Ağustos, 2004

Bu tezde, BilVideo isimli video veritabanı yönetim sistemine anlamsal sorgulama yeteneğini kazandırılmıştır. Çalışmamız, videoların anlamsal içeriğinin çıkartılıp saklanması için tanımlanmış olan video veri modeline ve video verisini sorgulamayı destekleyen sorgu diline sahip olan bir çalışmaya dayanmaktadır. Anlamsal sorguların görsel ve yazılı olarak girilmesini desteklemesi için BilVideo'nun Internet tabanlı arayüzünde de değişiklikler yapılmıştır.

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

Contents

List of Figures

Chapter 1

Introduction

During the last 10 years, creation, storage and distribution of videos have improved so much with the help of advances in compression techniques, decreasing storage costs and availability of high speed transmission. These improvements have led to the emergence of new application areas for videos like video archives, digital libraries, video-on-demand, and e-commerce. As a multimedia data type, video draws more attention with the increase in the amount of video data. Since traditional data management techniques are not sufficient for management and retrieval of large amount of video data, a new concept called video database management has emerged. In the simplest form, a video database is composed of digitized video plus some textual information about these video that are used for retrieval purposes. The most important problems in the creation of a video database are the extraction of the textual data from videos and the indexing of this extracted data. In the literature, there have been a variety of research works conducted on those issues. The information to be extracted from videos can be categorized as follows:

- *Moving objects*: The information about objects of interest in a video like which object is seen in a specific video frame, what is the 2D and 3D relations between these objects and how these objects move among contiguous frames.

- *Low-level features of objects*: The information about objects of interest in a video like the color, shape and texture of them.
- *Semantic information*: The information related to the semantic content of a video. Most common semantic information extracted from videos are:
  - Metadata about video, like subject, production year, producer or director of the video.
  - Information about objects of interest in a video, like hair color, height or title of the person.
  - Information about the events that take place in a video, like a business meeting, a sports match or a party.

Video data can be processed automatically or manually to extract such semantic information from videos. Automatic techniques extract information by the help of algorithms like object detection, text recognition, speech recognition and motion segmentation. Then this data is processed to extract semantic information. Manual techniques are also developed for helping users to specify the objects of interest and the details of events in a video.

Although the automatic extraction concept sounds good at first, related techniques generally do not perform well. Because, automatic extraction of initial set of data is a very complicated process especially when the domain is not well known. Besides, the semantic information extraction by processing automatically extracted data is not an easy task when the domain of videos is unknown.

For the last several years, a video database system called BilVideo is being developed at Bilkent University [1, 2]. BilVideo System was initially designed for supporting spatio-temporal and trajectory queries over the videos stored in the database. For this purpose, an information extraction tool called *Fact Extractor* was designed and developed, which is used for extracting information about objects of interest in a video, like the appearance of objects in specific frames, 2D and 3D spatio-temporal relations among objects in frames, and the movement of objects among contiguous frames. The extracted information is kept as Prolog

facts and stored in a knowledge base. The system has its own SQL-like query language. The submitted queries are sent to the Prolog engine to be executed over the knowledge base. The system has a web based query interface for formulating queries both visually and textually.

In this thesis, our main aim is to add semantic querying capability to BilVideo. For this purpose, we use the semantic data model developed in a previous Master's Thesis as a starting point[3]. In that work, a semantic information extraction tool, which stores the extracted information in a relational database, was developed and a semantic query language grammar was specified as an extension to the original query language of the BilVideo System. In our work, we have done necessary additions and modifications into the query processor of BilVideo to make the system accept and execute the semantic queries. We have also designed a new query interface for specifying semantic queries visually, and integrated it into the Web based query interface of BilVideo.

1.1 Organization of the Thesis

The thesis is organized as follows: Chapter 2 summarizes related work on semantic video modelling, semantic information extraction, storage of the semantic data and querying video databases. In Chapter 3, the architecture of the BilVideo system is described. In Chapter 4, the semantic video model that we used is explained. In Chapter 5, the execution strategies used for semantic queries are explained in detail. Implementation details of integration of semantic query execution capability into the BilVideo System are described in Chapter 6. Finally, Chapter 7 concludes the thesis.

Chapter 2

Related Work

In recent years, indexing and retrieval of video data according to its semantic content has become a very hot research topic. In the literature, the recent works on this topic can be categorized as follows:

- *Semantic video modelling*: Description of video data.
- *Semantic information extraction*: The extraction of semantic data from video.
  - *Feature extraction*: The automatic and manual ways of extracting low and high-level features from video.
  - *Semantic information deduction*: Ways of deducing extra semantic data from the extracted low and high-level features of video.
- *Storing semantic data*: The methods of storing extracted semantic information.
- *Querying video data semantically*: The methods of querying extracted semantic data.

In this chapter, main categories of the research conducted on indexing and retrieval of video data are presented. The research on semantic video modelling

is summarized in Section 2.1, semantic information extraction in Section 2.2, storing semantic data in Section 2.3, and querying video data semantically in Section 2.4.

2.1 Semantic Video Modelling

Indexing and retrieval of video data semantically require the extraction of semantic information from video clips stored in the database. This semantic information has to be extracted in an organized fashion to be able to algorithmically index and retrieve videos. A model has to be defined in terms of some predefined items in a video for defining the content of each video clip. There are many semantic video models proposed in the literature. As the items in a video that can be used to define a video are limited, many of those proposed models are quite similar. The most common items in these models are the objects of interest in a video and the events that occur in a video. Among the other items used are the place of the events, features of objects, backgrounds of the places where events occur, and so on.

In the semantic model proposed in [9], the main entities are events, objects that participate in these events and "actor" entities that describe object roles in the events. Events are described with name, location and time of the event and an event id.

In [19], a model called SemVideo is proposed. SemVideo model has the following types of information:

- *Videos*: The database manages many videos, each being represented by a unique identifier.
- *Video objects*: A set of video segments that satisfy some constraints. Video objects are abstract and not really stored in the database.
- *Semantic objects*: A semantic object is a description of knowledge about the video. It has a number of attributes, each having a corresponding value.

Each semantic object in the video has a unique identifier to differentiate from others.

- *Entities*: An entity can be either a video, a video object or a semantic object.
- *Relationships*: A relationship is an association between two entities. It can be time related or semantic related.

In [21], a video is modelled with the events associated to the shots, objects of interest in these shots and the static scene of these shots.

In [12, 13, 14, 15], a probabilistic multimedia object called multiject is proposed. Multijects can belong to any of the three categories: objects (car, men helicopter), sites (outdoor, beach) and events (explosion, man-walking).

2.2 Semantic Information Extraction

After defining a semantic video model for describing video clips, some methods to extract semantic information according to the defined model are needed. In the literature, many automatic and manual techniques of extracting low and high-level features from video clips have been proposed. For helping the automatic or manual extraction process, the developed tools use shot boundary detection, object detection, text recognition, speech recognition and motion segmentation algorithms. In [5, 7, 21, 22, 25], shot detection algorithms are used for identifying the main shots in video clips, and then the keyframes representing the identified shots are found. In [7], the keyframes are compared with preannotated database of images to find textual representations for them. In [8, 28], object detection with background subtraction algorithm for identifying the objects of interest in video clips is used. In [6, 7], text recognition algorithms are used for capturing the captions from videos. In [25, 27], speech recognition algorithms are used for extracting the speech from video clips to be used for indexing. In [8, 25, 28], some motion segmentation algorithms are used for detecting predefined specific

motions of objects in videos that are associated with semantic meanings. It is not always easy to extract semantic information automatically especially when the domain of videos is not restricted specifically. Thus, manual annotation is needed in such situations. In [5, 21, 29], annotation tools for extracting semantic information manually are proposed.

In [7], cuts are detected in the video sequence, where consecutive frames show a large difference, by the help of automatic shot detection algorithms. Keyframes are then extracted to represent each shot. Thus, a video sequence is condensed into a few images, hence this forms a compact key frame representation of the video. Each key frame is compared with an annotated database of images to obtain a textual description of the scene.

In [8], the motions of objects among consecutive frames are tracked by using motion segmentation and background subtraction algorithms. Then, the motions of objects are classified into following predefined events:

- appearance/disapperance of objects,
- merging/splitting of objects,
- entry/exit of objects, and
- occlusion of objects.

By using this extracted event information, summaries of videos are created according to the events that are selected as critical by the users.

In [11], a new model based on the clustering of video shots is proposed. Similar shots are clustered together and a new representation of the video data as a Time-Space Graph (TSG) is produced. A set of temporal relations between clusters are defined based on TSG. The clusters and their relationships describe the video as a Temporal Clusters Graph (TCG). Extraction of semantic units consists of exploring TCG according to the semantic of the temporal relations.

In [16], a semantic video indexing algorithm based on finite-state machines that exploits the temporal dependencies of low-level descriptors extracted from

videos is proposed. The proposed algorithm is applied to the analysis of soccer video sequences for identifying events such as, goals, shots to goal, and so on. Then summaries of videos are created by using the identified events.

In [20], an inductive decision-tree learning method to arrive at a set of if-then rules is directly applied to a set of low-level feature-matching functions. Most important low-level features used by the system are the motion, color and edge features of objects among contiguous frames. The extracted rules show the order and priority of each low-level feature in the classifier when multiple features are present, which is very important for on-line fast video understanding and indexing.

In [22], a method for finding the semantic scenes in a video, which are composed of semantically related shots, is proposed. By using shot detection algorithms, a video is divided into shots. Then, the system tries to find the shots that are related to each other according to the similarity of the frames in those shots. The combination of related shots forms a semantic scene in a video. The idea behind this method is the fact that closely related shots are put one after another during the editing phase of video creation.

In [24], a method for retrieving the content related to the spatio-temporal relations among objects in video data is proposed. The prototype system extracts moving objects from video data and evaluates their spatio-temporal correlation. Abstract representation of spatio-temporal correlation among objects is defined by a knowledge designer, and it is stored as an element of a knowledge base that is associated with a keyword representing the semantic contents of spatio-temporal correlation. As an example of possible applications, the authors concentrated on the case of video database of soccer games.

In [25], the low-level features extracted from videos are processed to find high level information that is used for querying the video database. Scene change detection, shot classification, text recognition (for captions), speech recognition and motion segmentation algorithms are used by the system to extract the low-level features.

In [26], the low-level features extracted from sports videos are used for identifying specific events in sports videos. For this purpose, both rule based and probabilistic inference are used.

In [27], a method to segment a sports video into semantic units to represent a broadcast sports video is proposed. Speech that takes place in videos is considered for this purpose by using speech recognition algorithms.

In [28], a method for extracting semantic information from videos by tracking the head and hand position and motions of a person is presented. They proposed that the posture of a human can be estimated by using the following information:

- *Position of head* implies not only a position where human is but also a posture whether he/she is standing or sitting.
- *Direction of head* implies what he/she is looking at.
- *Positions of hands* imply gestures and interactions with objects.

By matching the extracted information to a set of possible actions according to a probabilistic model, natural language sentences representing the frames are generated automatically.

In [5], a new type of video production framework A4SM (Authoring System for Syntactic, Semantic and Semiotic Modelling), which is for automated and semi-automated annotation of audiovisual media, is presented. It is claimed that annotation of a video after its creation is very hard for the people who are annotating it. Because of this reason, the annotation of the video during its creation is a better way. Thus, they designed a digital camera and a handheld device that is connected to the camera and runs an annotation program. The system is designed for news production that is to be used by the cameraman and the reporter for annotating the video. This extracted information is used by the news editors with the help of a post production editing tool for identifying the parts of the video that are important and are worth to be used and archived.

In [12, 13, 14, 15], a method that converts the indexing problem into a multimedia pattern recognition problem is proposed. A probabilistic multimedia object (multiject) is proposed that has a semantic label and summarizing a time sequence of a set of features extracted from multiple media. Multijects can belong to any of the three categories: objects (car, man, helicopter), sites (outdoor, beach), or events (explosion, man-walking). For identifying the multijects in a video, the system first segments the video clips into shots using shot detection algorithms and then each shot is processed to obtain regions homogeneous in color and motion.Then, the dominant regions are then tracked within the shot and labelled. All regions are labelled by a single person choosing from a list of semantic labels. Each region is then processed to extract a set of features that characterize the visual properties including the color, texture, motion, shape and edginess of each region. The regions (multijects) are then related to other multijects by a factor graph framework that uses sum-product algorithm in a tree and a graph called multinet is constructed, which is used for modelling the interactions between multijects.

In [21], a video summarization system for mobile devices is proposed. The system is composed of an *annotation tool* called VideoAnn for extracting semantic information from videos to be used for querying the system, a *semantic video summarizer* for creating a summary of the video according to the extracted semantic information, and a *transcoder* for converting the video format supported by the mobile device and an application interface on mobile device for querying and viewing the video summary. In VideoAnn, the annotation process starts with automatic detection of the shots and the keyframes representing these shots. Then, the users enter the objects, events and static scene in the detected shots. The users are also able to specify the exact locations of the objects in the keyframes by drawing a minimum bounding rectangle for the objects.

In [23], a method of retrieving video data from films by specifying the semantic content of scenes, is proposed. Related to the creation of films, there is a so-called "the grammar of the film", which is an accumulation of knowledge and rules for expressing certain semantics of a scene more effectively. In this research, the main emphasis is on the features of video that can be observed as a consequence of the

effects of the grammar of film. These features are classified as follows:

- *Combination of shot length*: The length of a shot implies quick or slow passing of time. When quickness or heavy action needs to be emphasized, a number of short shots are connected. On the other hand, slowness is emphasized by a sequence of long shots.
- *The visual dynamics of the shot*: Instead of extracting certain semantic objects, overall appearance of images is considered. The usage of static shots in which there is little difference between two consecutive frames and dynamic shots in which there is much difference between consecutive frames, are detected to find out if the scene is calm and quite, busy and active or if it contains heavy action.
- *The combination of similar shots*: Shots are regarded as similar if the color range and distributions are roughly the same for two shots. The combination of similar shots are tried to be matched with some predefined patterns to identify if the scene is a chase scene or not.

In [29], a practical application called OntoLog for searching and navigating in metadata for temporal media is presented. Currently, the OntoLog System consists of five main modules, *the Annotation Tool*, *the Resource Description Framework (RDF) Storage*, *the Temporal Media Source*, and *the Search and Browse Server* and its *Client*. The role of the Annotation Tool is to produce and edit annotations to form RDF based metadata describing the contents of the material stored in the Temporal Media Source. The information extracted by the tool is a list of people, places, events, topics or objects that occur in the video, and indicate the temporal intervals in which each information type is present and the extracted information is organized in hierarchical ontologies instead of flat lists.

2.3 Storing Semantic Data

Semantic information is extracted from videos according to the defined semantic video model by using different methods. This extracted information should be somehow stored to be used for indexing and retrieval purposes. In the literature, many different ways of representing and storing the extracted information are proposed.

In [17, 18, 21], it is proposed to represent and store the extracted semantic information in XML format. MPEG7 has its own standardization for representing the semantic information, which is described in [17]. In [21], MPEG7 standardization is directly used for representing and storing extracted semantic information, but in [18] they use their own XML format for semantic information representation instead of using the MPEG7 format.

Another approach for representing the semantic information is graph based representation. In this approach, there is a need for defining a way of mapping the semantic information into a graph based representation and this representation is used for storing the semantic information. In [12, 13, 14, 15], a graph based representation of semantic information is proposed which is called a multimedia network (multinet) composed of multimedia objects (multijects) and relations between them. In [9], a mapping between semantic information and a graph based representation is made in such a way that the main components of their semantic model (events, objects in events and roles of objects in events) is mapped to vertices and the relations between these components are mapped to edges in a graph. In [29], the Resource Description Framework (RDF) [30], which is a standard for machine-readable metadata built on the semantic network model, is used for mapping semantic information and as RDF Storage module, they use MySQL relational database that has RDF programming libraries available for it.

In [24], extracted semantic information is represented using Relation Description Factor, which is a set of primitive descriptors specified in the definition of knowledge for the representation of abstract spatial correlation among objects. These descriptors are stored as an element of a knowledge base that is associated

with a keyword representing the semantic contents of spatio-temporal correlation.

2.4 Querying Video Databases

In the video database management systems that store the extracted semantic information using graph-based represantation, the problem of querying the system is converted into a pattern matching problem. Such systems take the queries directly in graphical format or convert the queries into graphical format and then they try to match the query graph with the stored ones. In [12, 13, 14, 15], a Query-By-Keyword scheme is used for querying the system. The keywords entered by the users are used to search among a predefined set of keywords representing certain high-level or semantic concepts. Among the results of this search, the users select some of them as examples, whose multimedia network (multinet) representations are used for querying the whole set of stored multinets. In [9], the users directly enter their queries in a graphical format and this query graph is compared with the stored set of graphs using their graph matching algorithm.

In [19], two formal query languages are proposed for querying the video database. These languages are *Video Algebra* and *Video Calculus*. Queries in video algebra are composed using a collection of operators like boolean, physical, semantic, projection, composition and update operators. Video calculus is an extension to the relational calculus and it is an alternative to the video algebra. It allows users to describe the set of answers without being explicit about how they should be computed. As in relational calculus, the language for writing formulas is the heart of video calculus.

In the video database system presented in [21], users are able to specify their requests through a GUI in terms of:

- *Preference topics*: The users choose their preferred topics among predefined ones and then select videos from the list of videos belonging to those chosen

groups.

- *Topic rankings*: The users choose their preferred topics according to the topic rankings and then select videos from the list of videos belonging to those chosen groups.
- *Query keywords*: The users enter some keywords that are used for searching the video database and then they select a video from the results of the search.
- *Time constraint*: The users enter their time constraints which is used for searching the video database and then they select a video from the results of the search.

In [24], the extracted information is represented using the Relation Description Factor and stored in a knowledge base. The users enter their queries using Relation Description Factor, which are then compared with the ones stored in the knowledge base.

Chapter 3

BilVideo Video Database System

In this chapter, the BilVideo video database management system [1] in which this thesis is integrated into, is described. The organization of this chapter is as follows: The architecture of BilVideo is given in Section 3.1. The spatio-temporal data model proposed for video data type is described in Section 3.2. Types of queries that can be answered by the system are given in Section 3.3. The query processing approach is explained in Section 3.4.

3.1 System Architecture

The client/server architecture of the BilVideo system is shown in Figure 3.1. Client side of the system consists of the *Fact Extractor* and the *Video Annotator* tools for information extraction and the web-based query interface for video retrieval. Fact Extractor tool is used for extracting the spatio-temporal information of objects in videos and this information is stored in a knowledge base as Prolog type predicates. Video Annotator is used for extracting the semantic information from videos to construct a feature database and this feature database is stored in a relational DBMS[1]. The heart of the system is the Query Processor, which is responsible for answering user queries in a multi-threaded environment.

[1]Currently Oracle8i RDBMS Server is being used for this purpose

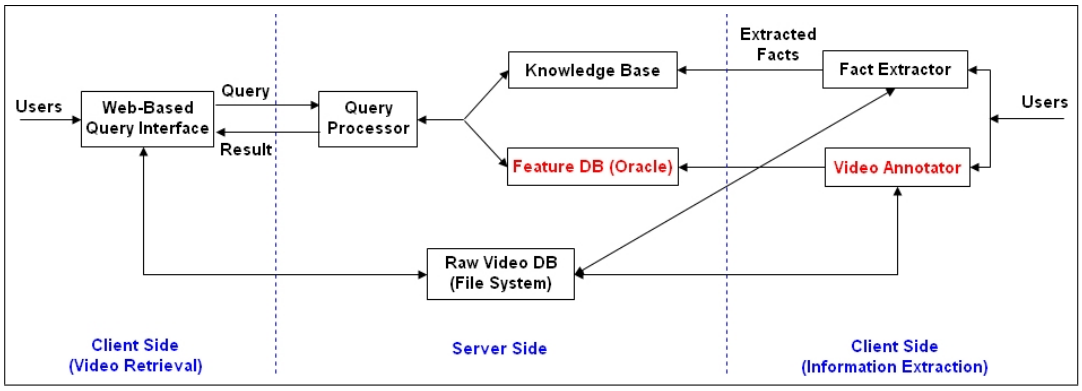

Figure 3.1: BilVideo System Architecture

The users interact with the query processor through the web-based query interface. This interface supports the entrance of textual and visual queries. A visual query is formed by a collection of objects with different attributes including object-trajectory with similarity measure and spatio-temporal ordering of objects in time. Motion is sketched as an arbitrary polygonal trajectory for each query object. The system has a SQL-like query language for textual queries. The earlier version of the system supports spatio-temporal, relative object-motion, object-appearance and similarity-based object-trajectory queries and it uses the knowledge-base to answer these types of queries. By using spatio-temporal relations, a restricted set of events can also be specified as query conditions.

3.2 Video Data Model

In BilVideo System, segmentation of video into shots is done using spatial relations of objects in video frames. Spatial relations can be grouped into mainly three categories: topological relations that describe neighborhood and incidence, directional relations that describe order in space, and distance relations that describe range between objects. The Fact Extractor tool is used for extracting the spatial relations among salient objects by using their minimum bounding rectangles which are specified manually.

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

3.3 Query Language

BilVideo System has a SQL-like query language for specifying spatio-temporal queries. In [3], an extension to this query language is proposed for specifying semantic queries. The details of this extension is described in Chapter 4. In this thesis, the integration of this extension to the BilVideo system is implemented. Details of this integration is described in Chapters 5 and 6. Without the semantic extension, the query language can be used to specify mainly five types of queries. These are:

- *Object queries* are used to retrieve objects, along with video segments where the objects appear.
- *Spatial queries* are used to query videos by spatial properties of objects defined with respect to each other. Three types of spatial properties are supported by the system which are topological relations that describe neighborhood and incidence in 2D-space, directional relations that describe order in 2D-space, and 3D-relations that describe object positions on z-axis of the three dimensional space.
- *Similarity-based object-trajectory queries* are used to query videos to find paths of moving objects.

- *Temporal queries* are used to specify the order of occurrence for conditions in time. Supported temporal operators in temporal queries are before, meets, overlaps, starts, during, finishes and their inverse operators.

- *Aggregate queries* are used to retrieve statistical data about objects and events in video data. There are three aggregate functions, *average*, *sum* and *count*.

3.4 Query Processing

In BilVideo System, the web-based query interface is used for specifying textual and visual queries. Visual queries are transformed into the SQL-like textual queries automatically by the interface. The resulting queries are sent to the query processor. Query processor is responsible for retrieving and responding to user queries in a multi-threaded environment. The queries are reconstructed by the query processor as Prolog-type knowledge-base queries. Results returned from the knowledge-base are sent to the web clients. The phases of query processing for spatio-temporal queries are as follows (Figure 3.2):

- *Query parsing*: The lexical analyzer partitions a query into tokens, which are passed to the parser with corresponding values for further processing. Using the predefined grammatical rules parser creates a parse tree to be used as a starting point for query processing. This phase is called *query recognition phase*.

- *Query decomposition*: The parse tree generated after the query recognition phase is traversed in a second phase, which is called *query decomposition phase*, to construct a query tree. Queries are decomposed into three basic types of subqueries: *plain Prolog subqueries* or maximal subqueries that can be directly sent to the inference engine Prolog, *trajectory-projection subqueries* that are handled by the trajectory projector and *similarity-based object-trajectory subqueries* that are processed by the trajectory processor.

- *Query execution*: The query tree is then traversed in pre-order in the next phase of query processing, *query execution phase*, executing each subquery separately and performing interval processing so as to obtain the final set of results.

- *Interval processing*: Results returned by subqueries are further processed in the internal nodes of the query tree to compute the final result set for user queries.

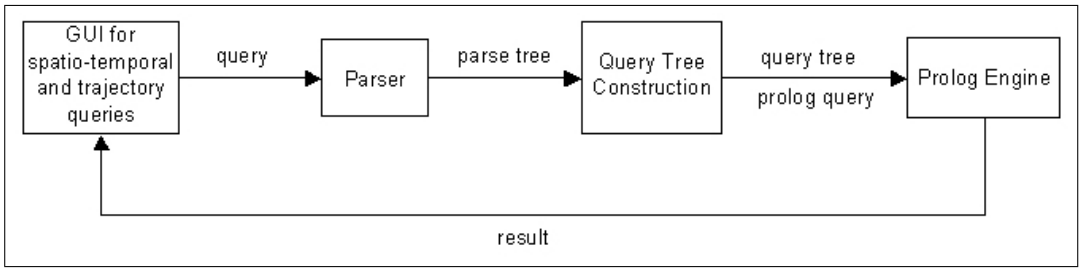

Figure 3.2: Phases of query processing steps for spatio-temporal queries.

A semantic video model and semantic extensions to the query language are proposed to support semantic queries [3]. To this end, a tool is implemented for extracting semantic information from video clips. The proposed extensions are integrated into BilVideo and also necessary additions are made to the web-based query interface for specifying semantic queries.

Chapter 4

Semantic Video Model

A video has two main layers. These are the *feature and context layer*, which deals with the low level details of video, and the semantic layer, which deals with the meaning perceived by humans. We have to map this data into an internal representation to capture semantic data from video. In this way, we can develop efficient methods for storing and retrieving this data. We need a model that defines semantic data to be able to make such a mapping. To this end, a hierarchical semantic video model was designed in [3] that captures the events, subevents, objects of interest and the bibliographic data about videos.

Event is an instance of an activity that may involve many different objects over a time period. Subevents are used to model the relations between objects and also to detail the activity into actions which are the acts performed by living objects. The information related to the video other than its context like name, producer, director, etc. are considered as bibliographic data [3].

In this chapter, main aspects of the semantic video model of BilVideo is explained. In Section 4.1, we explain the internal representation of the semantic video data. In Section 4.2, we explain how this data is stored. In Section 4.3, we present the video annotation tool called Video Annotator which is used for extracting semantic information from video clips. In Section 4.4, we present the semantic query language and in Section 4.5, we present the graphical user

interface (GUI) designed for entering semantic queries.

4.1 The Data Model

In a video, many different types of activities can be seen like business or political meeting, war or peace talks. To be able to distinguish the same type of activities in one video or among many videos, we need to identify these activities more specifically.

In our model, events are used for that purpose. If we think of activities as classes, then events are the instances of these classes. Extra information is held in events to make them distinguish among others like the objects that take place in the event, start and end time in video, roles of the objects in the event, location and time of the event. Suppose we have a news video in which it is talked about the political meetings held on that date. In this video, there are many video sequences that are about the activity type political meeting, but political meeting between USA and China is different than political meeting between France and Germany in many different ways like the attendees, location of the meeting, start and end time of the meeting sequences. In this example, it is seen that political meeting is an activity but USA-China political meeting and France-Germany political meeting are two events and in these events the delegations are the objects with attendee role, which is a role defined for meeting activity type.

Subevents are used to detail the events and to model the relations between objects of interest. For example USA-China meeting is an event and press conference held by USA delegation is a subevent of the main event. One event can contain many subevents in it.

In an event, there may be many objects of interest and each object may have some attributes that distinguish object from other objects. Thus, a model should also be able to capture those attributes.

The information that can be captured by our model can be categorized into

three main groups:

- *Bibliographic or metadata* : Data about video
- *Object data* : Data about objects of interest in video
- *Event data* : Activities and actions taking place in video

Video name, duration, producer, director, video types, audience and subject of video are considered as bibliographic data. The attributes of the objects of interests in a video are considered as object data and the data about the events and subevents are considered as event data. The location and time of the event, the activity type, the objects that take place in event and the begin-end times are considered as event specific data. The subactivity type, begin and end times and the objects in subevent are considered as subevent specific data.

A video consists of activities, an activity consists of actions and objects take place in both activities and actions. This depicts a hierarchical structure. By associating activities with events and actions with subevents, we carry out this hierarchical structure into our semantic model. In our model, we also use this hierarchy for segmentation of video into sequences and scenes by associating events with sequences and subevents with scenes.

4.2 The Database Schema

A database schema is needed for storing data captured by our semantic video model. According to the needs of our semantic video model, a relational database schema was developed which has fourteen database tables shown in Figure 4.1. The detailed table specifications can be found in Appendix A.

The database tables can be categorized into four groups:

- *Video metadata table*: TVIDEO is the one that holds bibliographic information about videos.

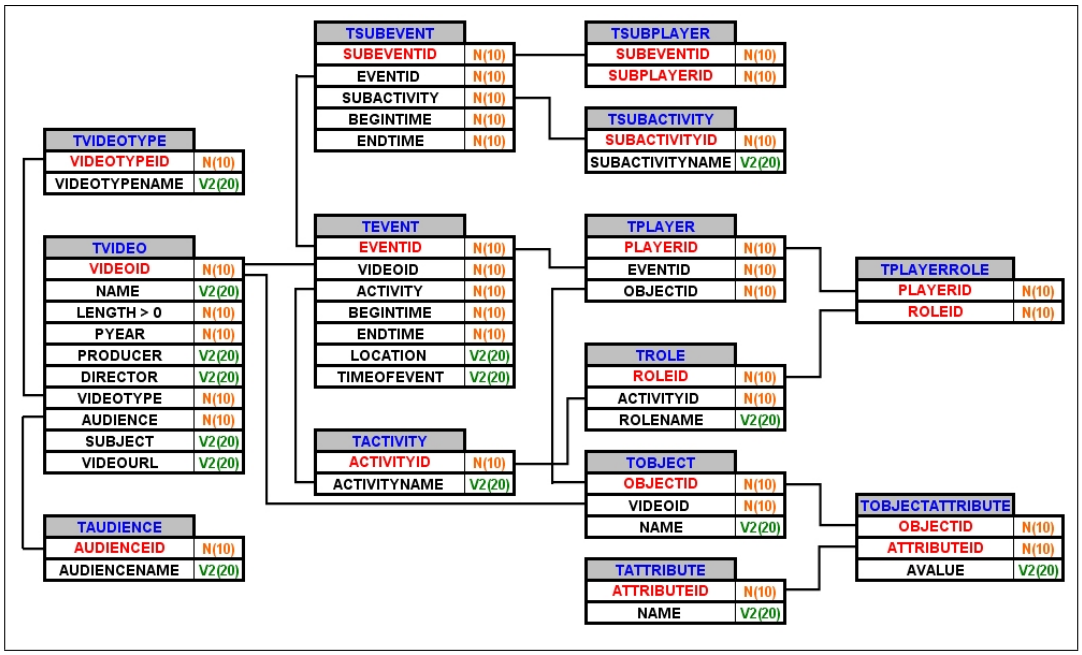

Figure 4.1: The Database Schema

- *Event tables*: TEVENT, TPLAYER, TSUBEVENT, TSUBPLAYER are the ones that hold the data related to events and subevents.
- *Object tables*: TOBJECT and TOBJECTATTRIBUTE are the ones that hold data about objects of interest in a video.
- *Utility tables*: TAUDIENCE, TVIDEOTYPE, TACTIVITY, TROLE, TSUBACTIVTY, TATTRIBUTE are the ones that hold utility data such as audiences, video types, activity types, roles for activity types and sub-activity types.

4.3 Video Annotator

A tool called the Video Annotator was developed To extract semantic data from videos [3], which is a database application coded in Java. The tool has two major functionalities; the first one is to play videos and the other is annotation

of video according to its semantic context. The video playing functionality of the tool is implemented using the Java Media Framework (JMF) API, which is a framework for adding time based media like audio and video functionality to Java applications and applets.

Features like playing all or part of the video and getting the details about the video file like the length, format and frame-rate of the video, are added to the tool by the help of JMF API. The annotation functionality of the tool lets users to extract semantic data from video clips according to our semantic model and also to view, edit and delete the extracted semantic data. Besides the extracted data is displayed by using a hierarchical tree structure and the users are also able to view the results of annotation by watching the parts that the event and subevents occur. The semantic data that the users can extract from a video can be categorized into five major groups:

- *Metadata about a video*: Video specific data such as video name, length of video, year of production, etc.
- *Object data*: Detailed information about the objects of interest in a video.
- *Event data*: Information related to the activities that take place in a video.
- *Subevent data*: Information related to the actions in activities.
- *Utility data*: Information about audiences, video types, activities, activity roles, subactivities and object attributes.

The Video Annotator tool is developed for extracting semantic data from videos according to our hierarchical semantic video model. The tool achieves this by forcing users for a specific annotation order that is basically based on our hierarchical model. During the annotation process, the users have to first start with annotation of the video metadata, then they have to annotate the events and subevents. The users can annotate objects whenever needed. Utility data annotation can be done anytime, as it is needed during annotation of other data types.

The main window of the Video Annotator tool shown in Figure 4.2. It has three major parts: the video player on the left, hierarchical video tree in the middle and the annotation control buttons on the right. Using the video player part, the users are able to open a new video which will be annotated or load a previously annotated video with its previously extracted semantic data. The input order of the annotation control buttons reflects the annotation order.

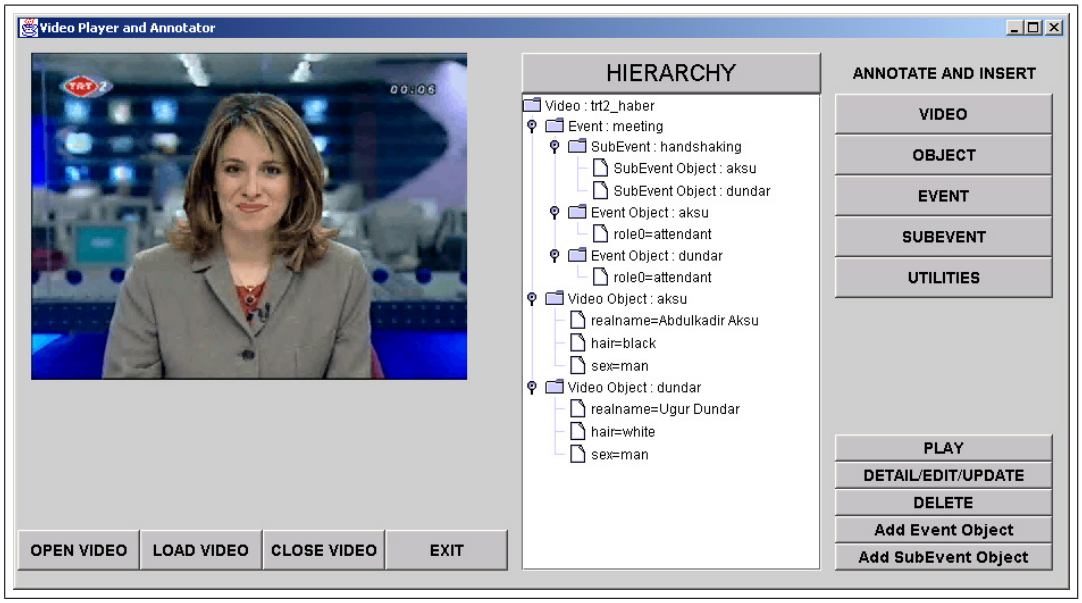

Figure 4.2: Video Annotator Main Window

For annotating a newly opened video, the users should click on the video, object, event, subevent and utilities button, which are located on the annotation control buttons part of the main window and which are corresponding to the semantic data types described above, to extract semantic data. After finishing the annotation, the users are able to see the result of the annotation by clicking the hierarchy button located at the top of hierarchical video tree part of the main window and then the tool constructs and displays the hierarchical structure of the extracted information. The users are able to watch the events and subevents of an annotated video by selecting them from the tree structure and clicking the play button located at the bottom of the annotation control buttons part of the main window and extracted data can be updated or deleted using the "Detail/Edit/Update" and "Delete" buttons. The users are also able to extend the

extracted data by adding new events, subevents and objects using the annotation control buttons. The details of the Video Annotator Tool can be found in [3].

4.4 The Query Language

Retrieval of the needed parts of a video by manually looking at the extracted semantic data can be possible for several number of videos but when the amount of data increases, a query language becomes a crucial need for efficient retrieval. Because of that reason, a query language was defined for semantic queries, which is a SQL-like language and which can be easily integrated into the existing SQL-like query language of BilVideo System [3]. This language makes it possible to query the system according to the video metadata, object attributes, event and subevent information. The main format of a query in the query language of BilVideo System is as follows:

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

The output type of the query is defined using *target*, the search domain is defined using *range* in the from clause and the conditions for the query is defined in the where clause. The possible output types are list of videos, list of segments in videos and variables. Two new output types were added that are sequences which are associated with events and scenes which are associated with subevents for the semantic query language of BilVideo. The possible values for *range* in BilVideo System are a list of videos or all of the videos in the database.

The conditions given in the where clause can be joined with logical (and, or, not) and temporal operators to create more complex conditions. The possible condition types supported in our semantic query language and their explanations are as follows:

- *Metadata conditions* are used for querying according to the video metadata such as length, title, producer or director of the video. These type of conditions are specified using the `meta` keyword as follows:

`meta ( meta_conditions )`

meta_conditions can be joined to create more complex conditions by using logical operators. The output type of metadata conditions is a list of videos where the specified meta conditions hold.

- *Event conditions* are used for querying according to the information extracted for the activities that take place in videos. This type of conditions are specified using the `etype` keyword as follows:

  `etype : event_name with ( event_conditions )`

  As an `event_condition` the users can give location, time, objects of interest with/without their role in the event and also the subevents that take place in the event. `event_conditions` can be joined to create more complex conditions by using logical and temporal operators. The output type of event conditions are a list of sequences in videos where the specified event conditions hold.

- *Subevent conditions* are used for querying according to the information extracted for the actions that take place in activities. This type of conditions are specified using the `setype` keyword as follows:

  `setype : subevent_name with player_list`

  As `player_list`, users can give the list of objects that appear in the video. The output type of subevent conditions is a list of sequences in videos where the specified players present.

- *Object conditions* are used for querying the objects (living or nonliving) of interest in videos. These type of conditions are specified using the `odata` keyword as follows:

  `odata ( object_attributes )`

  where `object_attribute := <attribute_name> : <attribute_value>`

  `object_attributes` can be joined using logical operators. The output type of object conditions is a list of videos where the specified object attributes hold.

- *Temporal conditions* are used for querying the system according to the order of occurrences of events in a video. This type of conditions are specified by joining event and subevent conditions with temporal connectors. The query language implements all temporal relations as temporal operators defined by Allens temporal interval algebra. The output types for temporal queries are decided according to the event condition types that are temporally joined.

4.5 The Semantic User Interface

The web based query interface of BilVideo [4] was developed using Java as an applet. It was developed in a tabbed manner in which a separate tab is designed for each query type and also a tab for combining different types of queries to form more complex and mixed queries. Following this tabbed structure of the original interface, we have also designed the GUI for entering semantic queries as a separate tab that can be easily integrated into the original one. We have developed semantic query tab GUI as a totally separate standalone Java application (Figure 4.3). The tab is composed of two main parts. These parts are the tree structure located on the left and the control buttons located on the right. The tree structure is used for representing the structure of the query hierarchically, and control buttons are used for adding or removing query items such as events, subevents, objects and video metadata to the query. After the development of the semantic query tab, it was integrated into the original web based query interface as a separate tab (Figure 4.4). After this integration, the users are now able to enter spatio-temporal, trajectory and semantic queries both visually and textually in different tabs, and also they are able to merge those queries to form more complex queries. The details of the semantic GUI are explained in Section 6.2.

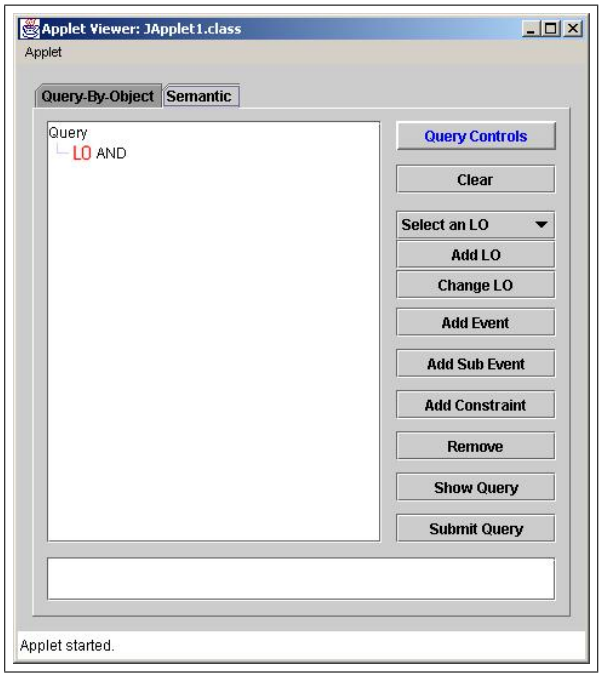

Figure 4.3: Standalone Semantic User Interface

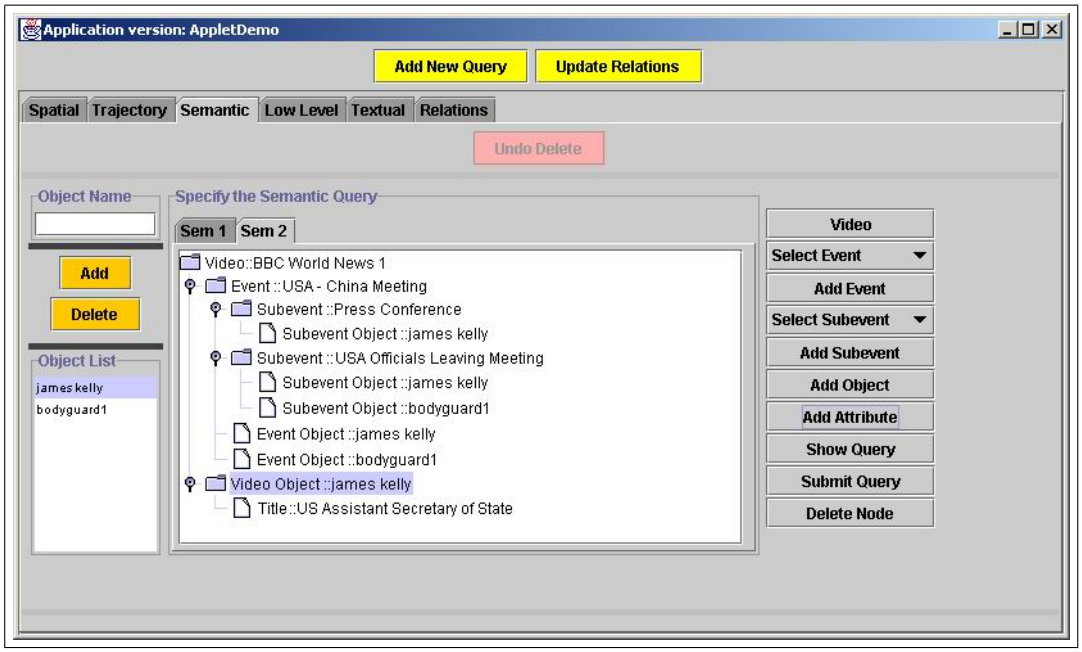

Figure 4.4: Integrated Semantic User Interface

Chapter 5

Semantic Query Processing

5.1 Query Processing in BilVideo

A SQL-like textual query language is designed for BilVideo to query the video data. The earlier version of the system supports spatio-temporal, trajectory, and low-level feature (color, shape, texture) queries. A Web based query interface is also implemented to support the entrance of spatio-temporal and trajectory queries both visually and textually. The GUI converts the entered visual and textual queries into the BilVideo's query language and sends it to the query processor of the system for execution.

The first thing done by the query processor is to create a syntactic parse tree for the query to check the syntax of it and to create a basis for the query tree creation phase. A query tree is formed by the query processor by traversing the parse tree, in which the Prolog equivalents of the subqueries of the original query and the way that their results will be merged is held as an information in the nodes of it. The constructed query tree is used as a guideline for executing the original query.

In the execution phase, the query tree is traversed in such a way that the Prolog subqueries are sent to the Prolog engine to be executed over the knowledge

base and then the results coming from the Prolog engine are merged according to the policies defined in the query tree. The final results are sent back to the web based query interface to be displayed to the end user. This whole process is shown in Figure 3.2.

5.2 Semantic Query Execution

The semantic query execution capability is an important feature for a video database management system to increase the querying power of the system. To support semantic queries, some parts of BilVideo need to be modified and also some new tools are needed.

For the spatio-temporal relations modelled by BilVideo, more information can be deduced from the extracted information. For example, for a specific frame suppose that we have the following information:

- object A is on the right of object B
- object B is on the right of object C

From this extracted information, we can deduce the following information:

- object A is on the right of object C
- object B is in the middle of object A and object C
- object B is on the left of object A
- object C is on the left of object A
- object C is on the left of object B

This example shows that by defining necessary inference rules, we can hold much more data than we actually have for spatio-temporal relations. In this

way, it is also possible to save storage space by not holding the data that can be deduced from the available data. Inference rules can also be used for trajectory rules easily. Doing this manually is a very hard job because making deductions according to the inference rules is a very complex task. All of these functionalities are present in Prolog. So instead of writing an engine for that, it was decided to use Prolog for inference. Because of this decision, currently the extracted information is converted to Prolog facts. The relations that can be deduced from the existing ones by using the inference rules are eliminated and then the rest are stored in a knowledge base. To answer the queries, this knowledge base and the inference rules are used to check if the query condition is satisfied or not.

The nature of spatio-temporal relations is very suitable for Prolog but it is not suitable for semantic data. It is not easy to deduce new semantic information from the available ones because it is very hard to define inference rules for semantic data. The relational database features are more suitable for handling semantic information. Thus, a new semantic video model was designed for handling semantic data, and a relational database was used for storing the data. As a relational database management system, we are currently using Oracle8i 8.1.7 Database Server. However, other database server products can be also used with our system when needed since our design is platform independent.

The major information extraction tool of the BilVideo System is the Fact Extractor. It was originally designed for extracting spatio-temporal and trajectory relations from videos. It processes video clips frame by frame and also its main concern among these frames is the selection of keyframes whenever a change occurs in the relations among objects. However, the main concern of semantic information is the identification of sequences and scenes in video clips. Because of this difference, adding the semantic information extraction capabilities to the Fact Extractor tool is very hard and also it will make the Fact Extractor tool so complex to be used by naive users whose only job is the information extraction. Thus, a new tool for semantic information extraction was needed, which is capable of extracting information according to the sequences and scenes in video clips. For this purpose, the tool called the Video Annotator tool was developed [3]. The main difference between the Fact Extractor and the Video Annotator

tools is the way they store the extracted information. Fact Extractor converts the extracted information to Prolog facts and stores them in a knowledge base. Video Annotator tool stores the extracted information in a relational database according to the database designed for the semantic video model.

The query execution steps of the earlier version of BilVideo design are shown in Figure 3.2. A user starts to interact with BilVideo System by forming a query visually and/or textually using the web based query interface of the system. Then, the user submits this query to system. The system first creates a parse tree for the query, which is used both for checking the syntax of the query and for storing the information that will be needed during the creation of the query tree. Then, the system creates a query tree, which is used as a guideline during the execution of the query. The subtrees of the parse tree are converted to plain Prolog queries and stored as nodes in the query tree.

During the query tree construction, subqueries of the original query are converted into plain Prolog queries using the information stored in the parse tree. These plain Prolog queries are stored as nodes in the query tree and these nodes are connected with nodes that hold the information of how the results of those Prolog queries will be merged. In the next phase, constructed query tree is traversed in a depth-first manner. The Prolog queries are sent to the Prolog engine and the results coming from the Prolog engine are merged according to the information stored in the query tree. The Prolog engine computes the results according to the facts that are stored in the knowledge base. When the traversal of the query tree is finished, we end up with the result of the original query submitted by the user. This result is sent back to the web based query interface and it is shown to the user visually.

In order to extend the querying capability of BilVideo with semantic queries, some modifications need to be involved in the query execution cycle shown in Figure 3.2. The modified query execution process is displayed in Figure 5.1.

The first modification is in the web based query interface of the system. The users should be able to enter semantic queries both visually and textually as well as the spatio-temporal and trajectory queries. Thus, a new GUI is developed for

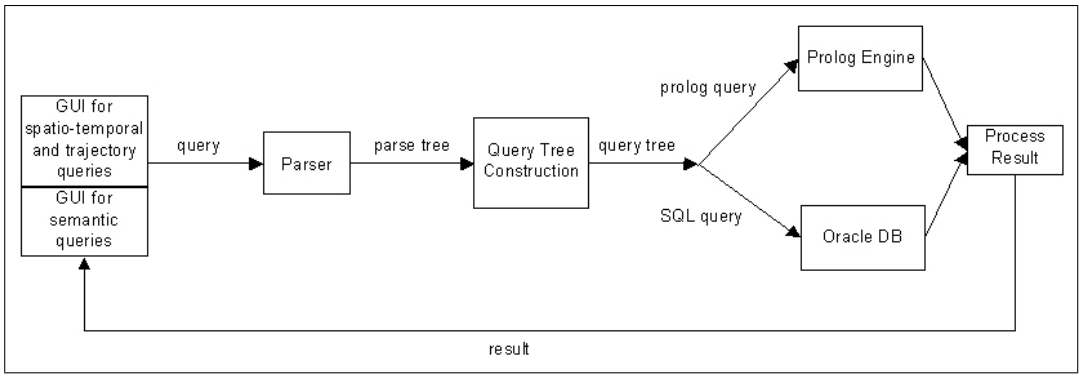

Figure 5.1: New query processing steps

semantic queries for entering queries visually and this new GUI is fully integrated into the web based query interface. By using this new version of the query interface, the users are able to enter spatio-temporal, trajectory and semantic queries separately and then they are able to combine these subqueries into one complex and mixed query. The first part of the system that encounters with the submitted query is the parser that was not able to handle semantic queries in the original design. Because of that we made necessary modifications in the parser to make it handle spatio-temporal, trajectory and semantic queries at the same time. In the query construction phase, the system takes the parse tree and constructs the corresponding query tree for it by using the information stored in the parse tree. In the new scheme, the parse tree not only holds the information for spatio-temporal and trajectory queries but also for semantic queries. Thus, some modifications are needed in the query tree construction phase for handling semantic queries. In the original design, subqueries are converted into plain Prolog queries because the extracted information is kept as Prolog facts in a knowledge base. However, the extracted semantic information is kept in a relational database in the new design. Hence, the semantic subqueries should be converted into plain SQL queries. For this purpose, the necessary code was added to the system for converting semantic queries into SQL queries. In the original design, instead of merging the results of the subqueries, the subqueries are merged before sending them to Prolog engine to get the merged result directly in some situations. In the new design, such situations became more complex because if one of the subqueries is semantic and

others are not, then it is not possible to merge subqueries beforehand because of the fact that semantic subqueries are converted to SQL queries and other are converted to plain Prolog quieries. Thus, the necessary code for identifying such problems was added to the system.

In the original design, the query tree is traversed and the subqueries are sent to Prolog engine after the query tree construction. However, with the edition of semantic query support, the SQL equivalents of the semantic subqueries should be directed to the relational database instead of Prolog engine. Thus, necessary code is added for identifying the subquery type and sending the Prolog and SQL equivalent of them to the correct location.

Retrieving the result of a SQL query from a relational database can be sometimes a complicated task. Hence, for hiding this complexity and making the code vendor-independent we have written an interface for it, which we call C++ Database Connectivity Interface (CDBC) that is similar to Java Database Connectivity Interface (JDBC). Using CDBC, the results of the SQL equivalents of semantic subqueries can be retrieved from the relational database very easily.

In the new design, the system identifies the type of the subquery and sends the Prolog queries to the Prolog engine and the SQL queries to relational database using CDBC interface. The system merges the results coming from the Prolog engine and relational database according to the information stored in the query tree. After the traversal, we end up with the result of the original query and this result is sent back to the web based query interface for displaying the result of the query visually to the user.

After the additions and modifications made to the query execution steps, BilVideo is now able to handle semantic queries as well. The users are able to construct complex mixed queries and the system is able to handle them.

The semantic query execution process can be explained comprehensively with a sample query.

Query: “Retrieve all news videos produced in 2003 that have “USA - CHINA

Meeting" event in which one of the attendees "James Kelly", the US Assistant Secretary of State, is giving a press conference behind microphones."

The equivalent of the query in BilVideo System's query language is as follows:

```
select video
  from all
  where meta(vtype:news and pyear:2003)
    and etype:'USA - CHINA Meeting'
          with ('James Kelly':role = Attendee
                and setype:'Press Conference')
    and odata('James Kelly'(Title:'US Assistant Secretary of State'))
    and behind('James Kelly',Microphones);
```

The query processor of BilVideo first extracts the target and range information from this query. According to this information, it retrieves all videos in the database that satisfy the conditions in the where clause. Then, it creates a parse tree for the query. During the creation of the parse tree, the query processor identifies different condition types included in the where clause and creates a subtree for each condition type of the condition types. Then, these subtrees are connected using the nodes created for representing connectors employed in the query. The where clause of the sample query contains four major condition types, which are:

- Meta condition:

  ```
  meta(vtype:news and pyear:2003)
  ```

- Event condition:

  ```
  etype:'USA - CHINA Meeting'
    with ('James Kelly':role = Attendee
          and setype:'Press Conference')
  ```

- Object condition:

  ```
  odata('James Kelly'(Title:'US Assistant Secretary of State'))
  ```

- Spatio-Temporal condition:

```
behind('James Kelly',Microphones)
```

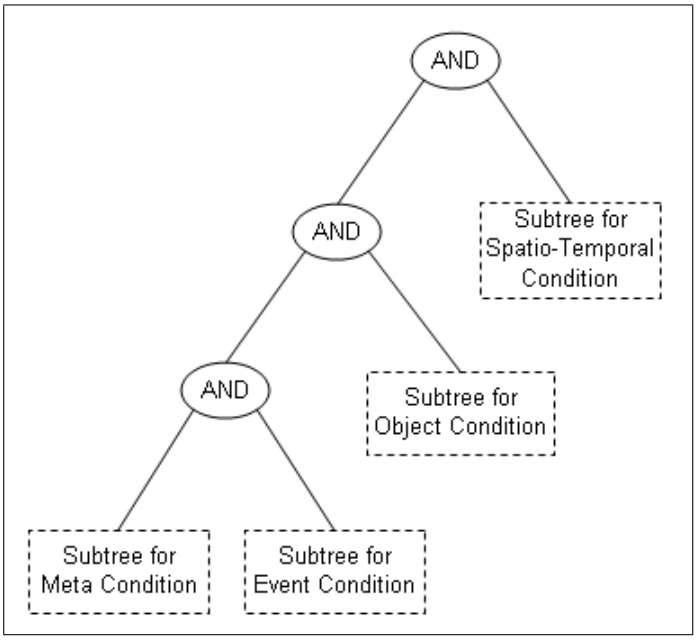

Figure 5.2: The parse tree for the conditions in the where clause of the sample query (rectangular nodes represent subparse tree equivalents of the corresponding condition types)

The parse tree for the conditions in the where clause of the query is shown in Figure 5.2. After the construction of the parse tree, the query processor creates a query tree from this parse tree in which the subtrees representing different types of conditions are converted into nodes representing the Prolog and SQL equivalents of them. The SQL and Prolog queries corresponding to the sample query are as follows:

- Meta Condition is converted into one node that contains the following SQL query.

```
select videoid into videolist
  from TVIDEO
  where pyear = 2003 and videotype = 'NEWS';
```

- Event Condition is converted into a subtree that contains nodes for representing the event name and the conditions specified in the with clause. This is because different parts of an event condition need querying different tables in the relational database. After the creation of the query tree, an optimizer can traverse this subtree and create one node representing an efficient equivalent of it. The subtree for the event condition is shown in Figure 5.3. The corresponding SQL queries are as follows:

  Query for event name: `etype:'USA - CHINA Meeting'`

```
select eventid into eventlist
  from TEVENT e, TACTIVITY a
  where e.activity = a.activityid
  and a.activityname = 'USA - CHINA Meeting';
```

  Query for subevent: `setype:'Press Conference'`

```
select eventid into eventlist
  from TSUBEVENT se, TSUBACTIVITY sa
  where se.subactivity = sa.subactivityid
  and sa.subactivityname = 'Press Conference';
```

  Query for object in event: `'James Kelly':role = Attendee`

```
select eventid into eventlist
  from TACTIVITYROLE ar, TPLAYER p, TOBJECT o
  where ar.roleid = p.roleid
    and o.objectid = p.objectid
    and o.objectname = 'James Kelly'
    and ar.rolename = 'Attendee';
```

Query for Event Condition is as follows:

```
select videoid into videolist
  from TEVENT
  where eventid in evenlist;
```

- Object Condition is converted into one node that contains the following SQL query:

```
select video into videolist
  from TOBJECT o, TOBJECTATTRIBUTE oa, TATTRIBUTE a
  where o.objectid = oa.objectid
    and oa.attributeid = a.attributeid
    and a.name = 'Title'
    and oa.value = 'US Assistant Secretary of State'
    and o.name = 'James Kelly';
```

- Spatio-Temporal Condition is converted into one node which contains the following Prolog query:

```
p_behind('James Kelly',Microphones,F);
```

After the construction of the query tree, the query processor traverses this tree starting from the leaves for finding the result set. On each query node, Prolog subqueries are sent to the inference engine and SQL queries are sent to the relational database. Then, the results are merged according to the connector nodes. If a connector node contains an AND operator, the intersection of the results of its child nodes is taken. If the connector node contains an OR operator, the union of the results of its child nodes is taken. When the query processor reaches to the root of the query tree, it will end up with the result of the query.

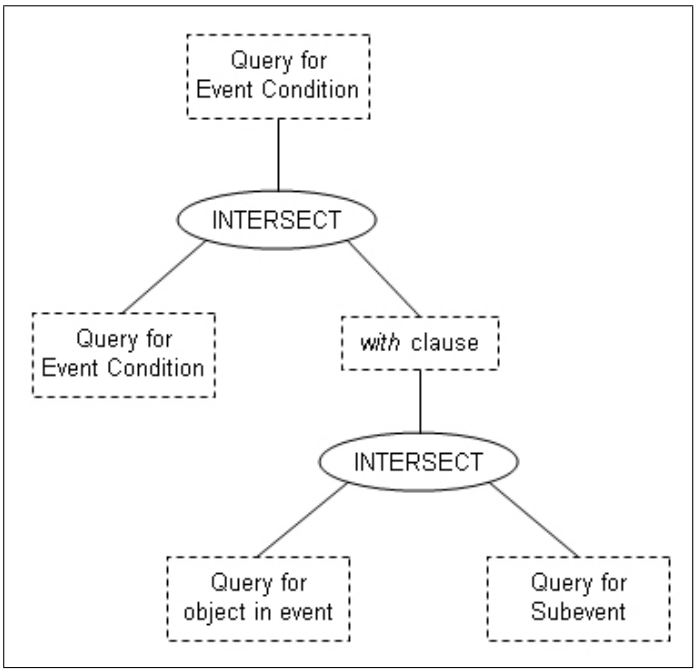

Figure 5.3: The subtree for the event condition of the sample query (rectangular nodes represent subparse tree equivalents of the corresponding condition types)

Chapter 6

Implementation Details

As mentioned before, our main contribution is to add semantic querying capability to the BilVideo System. The main components of BilVideo and the modifications we performed on each component can be described as follows:

- *Information extraction* component is used for collecting information about videos to be used for indexing and retrieval purposes. Video Annotator tool was originally developed for Microsoft Access DBMS which is not an adequate for our project especially in terms of scalability. Thus, necessary changes were made in the code to make Video Annotator work with Oracle RDBMS.

- *Web based query interface* is used for entering queries both visually and textually. This tool was designed in such a way that there exists a separate tab for each query type and there is a tab for combining different types of queries. A new tab was added to it for entering semantic queries visually.

- *Parser* is used for parsing the query submitted from the web based query interface for checking it syntactically and also for creating a parse tree for the query. Necessary additions were made to the code of this component for adding the capability of parsing semantic queries, and new node types were introduced to the parse tree for semantic queries.

- *Query tree constructor* is used for constructing a query tree from the parse tree generated by the parser. Necessary additions were made to the code of it to support the new node types introduced in parse tree (especially for constructing SQL equivalents of semantic subqueries) and also for merging subqueries before sending them to the relational database or the Prolog engine.

- *Database communication* component is used for sending SQL queries to the relational database and receiving the result of them. An interface called CDBC was developed for easing this process.

- *Query tree traversal* component is used for traversing the query tree for executing the query and constructing the result of it. Necessary additions were made to the code of it to handle the nodes for both Prolog and SQL equivalents of the subqueries. Also, this component was extended for calculating the results of SQL equivalents of the subqueries connected by temporal connectors.

- *Subquery result merge* component is used for merging the results coming from the Prolog engine and the database for subqueries according to the information stored in the query tree. In the query tree construction phase, the code for connector nodes were written in such a way that the connector nodes don't have to know the subquery types for merging. This gives us the chance to use the result merging code as is.

The main additions and modifications made on each part of the BilVideo System are explained in detail in the following sections. In Section 6.1, the changes made in the Video Annotator tool are explained. In Section 6.2, the details of the semantic query interface added to the web based query interface are described. In Section 6.3, extensions made on the parser of the BilVideo System are presented. The new properties on query tree construction are presented in Section 6.4. The modifications made in the execution of the queries by traversing the query tree are explained in Section 6.5, and lastly the details of the interface that is used for communicating with the Oracle RDBMS server are described in Section 6.6.

6.1 Information Extraction

The tool developed for semantic information extraction is called Video Annotator [3]. The tool was implemented in Java language and it uses JDBC for connecting to the database. Because of the fact that Microsoft Access and Oracle RDBMS require different versions of JDBC drivers, we first changed the JDBC driver used by the tool to the Oracle's driver. This is done by changing the driver package that is delivered with the tool and the code for defining the JDBC driver to be used before opening a connection to the database. We didn't need to change the code for opening a connection, sending the queries to the database and receiving the results. Because JDBC provides a generic interface for all of these operations that doesn't need to be changed when the database system changes. The only problem that we encounter was the use of automatic increment facility of Microsoft Access database. In Microsoft's databases, there is an automatic increment facility that can be used for automatically assigning unique values for primary key columns when user doesn't give a value for it in insert statements. This feature doesn't exist in Oracle RDBMS, instead there is a structure called "sequence" to be used for that purpose. Sequence is like a variable which provides the next available unique number for a column. In Oracle, sequences can not be associated with columns of a table automatically. It should be handled manually in the code. During creation of sequences a start point, max value and increment amount is specified. Whenever a new row is to be inserted to a table, the values of the columns that need to be associated with a sequence are retrieved from those sequences and put into the insert statement manually. When the next available value is requested from a sequence, it increments its current value by the increment amount and returns this new value. So it always provide a unique value. In the data model, realized with Microsoft Access, all of the primary key columns were created with automatic increment. So, for the data model created in Oracle, a sequence is created for each primary key column of the tables. In the source code of Video Annotator, before each insert statement, the necessary code for getting a unique value and putting it in the insert statement for the primary key columns is added.

After changing the JDBC driver and the addition of sequence structures to the data model, the Video Annotator tool became fully functional and ready to be used with Oracle RDBMS.

6.2 Semantic GUI

The earlier version of the web based query interface of BilVideo was initially handling only spatio-temporal and trajectory queries. Creating a textual query using a query language can become a very complex task when the number of conditions increases. To make the querying process easier, we designed a GUI for entering semantic queries (Figure 4.3). The users are able to enter semantic queries using this GUI without knowing the semantic query language. They enter their queries visually according to our hierarchical semantic model by using a tree structure similar to the one used in the Video Annotator tool, which is used for showing the results of annotation process. The GUI was developed in Java as a standalone application considering the fact that it would be integrated into the web based query interface of the BilVideo System.

The main window of the semantic GUI is composed of two main parts which are the tree structure located at the left for showing the entered query visually and the query control buttons for entering the query target, range and conditions. The ways of specifying semantic condition types are as follows:

- *Event conditions*: The user should open the dialog designed for entering the event name of the event condition by clicking on the "Add Event" button. In this dialog, enter the event name either manually or by selecting from the event names stored in the database.
- *Subevent conditions*: The user should open the dialog designed for entering the subevent name of the subevent condition by clicking on the "Add Subevent" button. In this dialog, enter the subevent name either manually or by selecting from the subevent names stored in the database.

- *Constraints*: There are four types of constraints that can be entered using the GUI. These are:
  - Meta conditions
  - Object conditions
  - Event conditions
  - Subevent conditions

  For entering these constraints, the users have to first select the corresponding nodes in the tree structure that the constraint will be located into. Then they have to click on the "Add Constraint" button. After that, a dialog is opened for entering the constraint that are available for the selected node.

- *Logical operators*: These are used for combining different types of conditions and constraints for creating more complex ones. For adding a logical operator to the semantic query, users have to first select the place to put the logical operator from the tree structure, and then select the type of the logical operator, and lastly press the "Add Operator" button.

After the condition part of the query is constructed, the users click the "Submit Query" button for finishing the query specification by specifying the target and the range of the query in the opened dialog.

While integrating the semantic GUI, slight modifications were done to the query interface as summarized below:

- *Removal of logical operator addition buttons*: In the original web based query interface, there exists a separate tab called "Relations" for combining subconditions using logical and temporal operators. Users first form their desired subconditions using corresponding tabs, and the textual equivalents of those subconditions are passed to the Relations tab to be combined using logical and temporal operators. Thus, after the integration of the semantic GUI, buttons for logical operators became useless. Because of this reason, they were removed from the semantic tab.

- *Functionality change of submit button*: In the standalone version, the submit button was used for sending the query directly to the query processor. In the web based interface, it is used for passing the conditions defined in the tab to the Relations tab.

- *Multiple query support*: As the original query interface supports submission of multiple queries of the same type at the same time for spatio-temporal and trajectory queries, we also added this functionality to the semantic tab by showing the tree structure of each entered semantic query in a separate tab.

6.3 Parser

The parser of BilVideo has two major functionalities. The first functionality is to check the syntax of the submitted query whether it is syntactically and grammatically correct or not. The second functionality is to construct a parse tree for the submitted query that represents the main structure of the query and holds the necessary information to be used during the construction of the query tree.

The parser code was written in C++ language and it uses the Lex and Yacc utilities for constructing the parse tree. Lex is used for decomposing the submitted query into smaller tokens according to the predefined lexical rules. If it encounters a problem during this decomposition, it is understood that there is a syntactical error in the submitted query. Yacc takes the tokens created by the Lex utility as an input and it uses them for constructing a parse tree for the submitted query according to predefined grammatical rules. If it encounters a problem during the construction of the parse tree, this means that there is a grammatical error in the submitted query.

The BilVideo system already has a Lex file for defining lexical rules and a Yacc file for defining grammatical rules for the query language of the system. Hence, we integrated the lexical and grammatical rules of the semantic query language

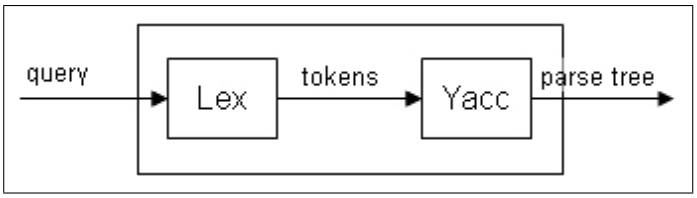

Figure 6.1: Parser of the BilVideo System

into those files.

Since the new semantic query language that we are integrating into the original query language has introduced several new token types, lexical rules for those token types were added to the Lex file of the system. Major semantic token types and the keywords that are associated with them are as follows:

- *Target types*: sequence, scene
- *Semantic conditions*: meta, odata, event, sevent
- *Metadata conditions*: vtype, audience, title, length, pyear, producer, director, subject
- *Event conditions*: location, time, role

Grammatical rules of the semantic query language were integrated into the original rules specified in the Yacc file. Major modifications in the grammatical rules are as follows:

- *Main query definitions*: In the BilVideo query language a query is defined using the following grammatical rule:

```
Query : SELECT target FROM range WHERE condition;
```

  To be able to support the new target types introduced by the semantic query language, following two query definitions are added:

```
Query : SELECT SEQUENCE FROM range WHERE condition
      | SELECT SCENE FROM range WHERE condition
```

- *Main condition types*: The condition types already supported in BilVideo are the spatio-temporal and trajectory conditions. Other than these, the following semantic condition types and their corresponding subqueries were added to the grammar specification:
  - *Metadata conditions*:

    ```
    metaconditions : META '(' metaconditionlist ')' ;
    ```

  - *Object conditions*:

    ```
    objectconditions : ODATA '(' objectconditionlist ')' ;
    ```

  - *Event conditions*:

    ```
    eventcondition : ETYPE ':' strvalue
                        WITH '(' eventconditionlist ')' ;
                   | ETYPE ':' strvalue ;
    ```

  - *Subevent conditions*:

    ```
    subeventcondition : SETYPE ':' strvalue WITH playerlist ;
                      | SETYPE ':' strvalue
    ```

These modifications in the grammatical rules also yield some changes in the resulting parse trees. New node types and subparse tree structures were introduced for semantic subqueries. With the use of these new node types, for each semantic query type different subparse tree structures are constructed by the system. These subparse trees are shown in Figure 6.2.

6.4 Query Tree Construction

The parser of BilVideo constructs a parse tree for the submitted query, which corresponds to the main structure of its condition part given in the where clause.

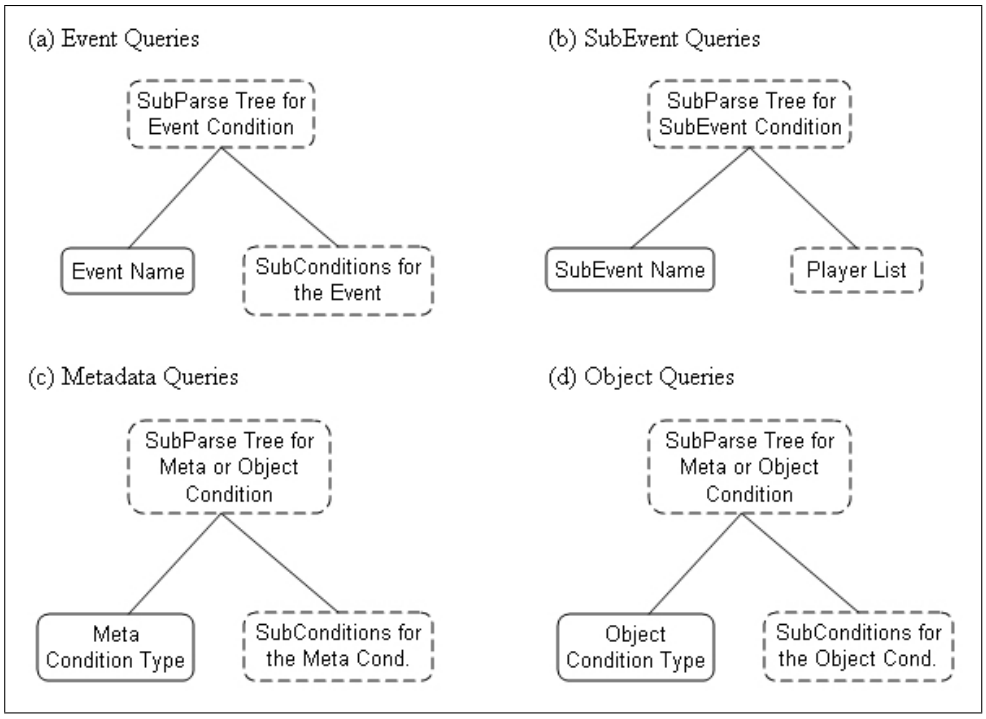

Figure 6.2: Subparse trees for semantic query types (dotted nodes represent subparse trees that have more child nodes than the nodes with solid lines)

In Figure 6.3, a sample query and its corresponding parse tree are shown.

The parse tree generated by the parser is then traversed and a query tree is constructed from it, which is used for executing the submitted query over the previously extracted information.

In a query tree, there can be mainly two types of nodes. These are:

- *Subquery nodes*: In the earlier version of BilVideo, these are the nodes created for holding the plain Prolog query equivalents of the subqueries of the submitted query which represents a specific condition type like spatio-temporal or trajectory conditions. There are a certain number of such subquery types and for each of these there exists a specific subparse tree structure. Hence, while traversing the parse tree, the system detects those subquery types by identifying the corresponding subparse tree structures

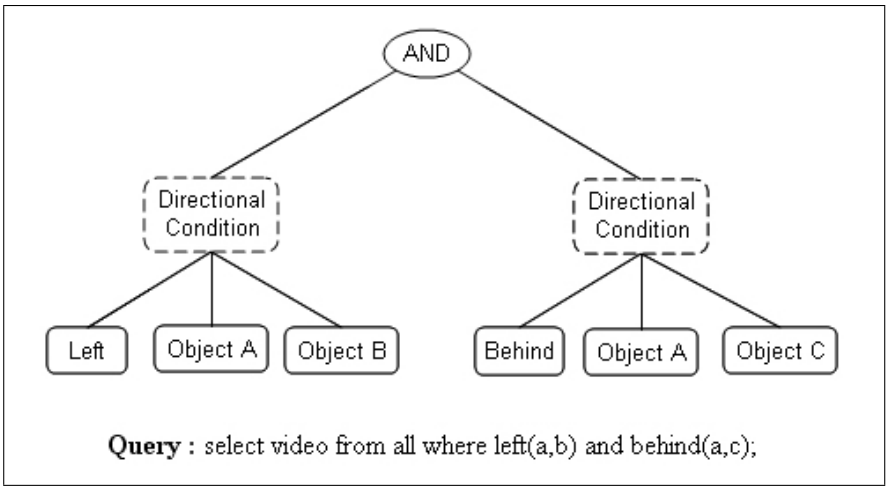

Figure 6.3: A sample query and its corresponding parse tree (dotted nodes represent condition types)

and creates a subquery node for each subquery. For example, in the sample query shown in Figure 6.3, there are two subqueries, which are `left(a,b)` and `behind(a,c)`. The subparse tree equivalents of them are shown in Figure 6.4.

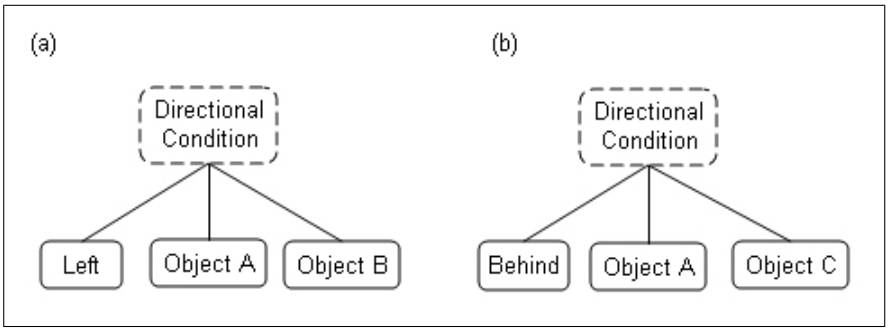

Figure 6.4: Subparse tree structures of the subqueries for the sample query (dotted nodes represent condition types)

During the traversal of the query tree, when the system detects these subparse tree structures, it creates two subquery nodes for each that holds the plain Prolog queries `p_left(a,b,F)` and `p_behind(a,c,F)`, respectively.

- *Connector nodes*: These are the nodes that hold the necessary information

to merge the results coming from child nodes. During the traversal of the parse tree, when a node representing a connector of the submitted query is encountered, the system checks the child nodes that are created beforehand. If it is possible to write a single plain Prolog query for representing the two child query tree nodes and the connector, then two child nodes are removed from the query tree and a single subquery node is created for representing them. If it is not possible to write a single Prolog query equivalent, then a connector query tree node is created and the child nodes are connected to this node. The type of the connector determines the methods for merging the results coming from its child nodes.

In the earlier version of BilVideo, which supports only spatio-temporal and trajectory queries, the possible subquery types, their corresponding query templates and their plain Prolog equivalents are, as follows:

- *Appearance Queries*
  - *Query template*

    `appear(object_list)`

    Objects in the object_list appears in the video.
  - *Prolog equivalent*

    `p_appear(object_list, F)`

- *Directional Queries* Possible directions that can be specified is left, right, above, below, west, east, south, north, swest(southwest), seast(southeast), nwest(northwest), neast(northeast).
  - *Query template*

    `direction(object1, object2)`

    object1 is on the specified direction of the object2.
  - *Prolog equivalent*

    `p_direction(object1, object2, F)`

- *Topological Queries* Possible topologies that can be specified is equal, overlap, disjoint, touch, inside, cover, coveredby, contains.
  - *Query template*

    `tpred(object1, object2)`

    object1 and object2 are located in the specified topology.
  - *Prolog equivalent*

    `p_tpred(object1, object2, F)`

- *Queries for 3D Relations* Possible 3D relations are infrontof, behind, sinfrontof(strictly infrontof), sbehind (strictly behind), tfbehind, tdfbehind, samelevel.
  - *Query template*

    `tdpred(object1, object2)`

    Specified 3D relation (tdpred) holds for object1 and object2.
  - *Prolog equivalent*

    `p_tdpred(object1, object2, F)`

- *Trajectory Queries*
  - *Query template*

    `tr(object1, path)`

    object1's movement is similar to the given path.
  - *Prolog equivalent*

    `p_tr(object1, path, F)`

During the construction of the query tree, the parse tree is traversed recursively in a depth first manner. During the traversal, the system tries to match the query tree with a subparse tree structure. Then, the system finds the Prolog equivalent of the subparse tree and returns this query. On the connector nodes

of the parse tree, the system checks the results coming from the traversal of its child nodes and it decides to form a single Prolog query or to create a connector query node according to the rules defined for that connector. The pseudo-code for the parse tree traversal for constructing the query tree is as follows:

In the query tree construction algorithm (Figure 6.5), some rules are applied when a parse tree node representing a connector is encountered. These rules are explained below:

- *Parenthesis rules*: If the child node is a query node, then enclose the query that is held in the query node with parenthesis and return this query node. Otherwise, create a connector node for representing parenthesis, make this the parent of the child node and return this newly created node. The detailed explanation of Parenthesis rules is given in Appendix B.3.
- *Not rules*: These rules create a connector node for representing not, make this the parent of the child node and return this newly created node. The detailed explanation of Not rules is given in Appendix B.4.
- *Temporal connector rules*: These rules create a connector node for representing the temporal connector, make this the parent of the child nodes and return this newly created node.
- *And-Or rules*: These are the most complex rules. There many different possible child node combinations which we have to deal with. All of these combinations are explained in Appendix B.1 in detail. The rules for the major combinations are as follows:
  - If both child nodes are query nodes, then the system creates one query node for representing these two child nodes and the connector, and then returns the query node.
  - If both of the child nodes are connector nodes of the same type with the current connector and each has one child query node, then the system restructures the nodes in such a way that a query node is created by combining query nodes, a connector node is created by merging the

```
constructQueryTree
    input  = pnode (parse tree node)
    output = data

    // Take the results coming from the child nodes
    if pnode->leftchild != null
        dataleft  = constructQueryTree(pnode->leftchild)
    if pnode->rightchild != null
        dataright = constructQueryTree(pnode->rightchild)

    // If node is not a connector, then
    // try to match it with a subparse tree type
    if pnode->type != connector
        if match with a subparse tree type
            data->content = prolog equivalent of the subparse tree
            return data

    // If node is a connector, then
    // apply corresponding connector rules
    if pnode->type is a connector
        if pnode->type = "()"
            data = apply parenthesis rules
        else if pnode->type = "Not"
            data = apply not rules
        else if pnode->type in (temporal_connectors)
            data = apply temporal connector rules
        else if pnode->type in (and, or)
            data = apply and-or rules
        return data

// The structure used for holding results for nodes.
struct data {
    string content
    querynode qnode
}
```

Figure 6.5: The algorithm for query tree construction

remaining parts of the child subtrees, and then a parent connector node is created for those newly created child nodes.

– If both of the child nodes are connector nodes, then a connector node is created for representing the current connector type, this node is made the parent of the child nodes and it is returned.

With the addition of the semantic query capability to the parser of BilVideo, new subparse tree structures are introduced for new semantic subquery types. As the semantic information extracted from videos is stored in a relational database, the subparse trees for semantic subquery types should be converted to the SQL queries. The possible semantic query types whose corresponding subparse trees are shown in Figure 6.2 and their SQL query equivalents are as follows:

- *Event queries*
  - *Query template*

    ```
    etype : <event_name> with ( <event_conditions> )
    ```

  - *SQL equivalent*

    ```
    select begintime, endtime, videoid
      from TEVENT
      where eventid in (select te.eventid
                          from TEVENT te, TACTIVITY ta
                          where te.activity = ta.activity
                          and ta.activityname = <event_name>)
        and eventid in ( <event_conditions> );}
    ```

- *Subevent queries*
  - *Query template*

    ```
    setype : <subevent_name> with <player_list>
    ```

  - *SQL equivalent*

```
select tse.begintime, tse.endtime, tsp.videoid
  from TSUBEVENT tse ,
      (select tsp.subeventid subeventid, to.videoid videoid
         from TSUBPLAYER tsp , TOBJECT to
         where tsp.subplayerid = to.objectid
           and tsp.subeventid in
              (select tse.subeventid
                 from TSUBEVENT tse, TSUBACTIVITY tsa
                 where tse.subactivity = tsa.subactivityid
                 and tsa.subactivityname = <subevent_name>)
           and to.name in ( <player_list> ) ) tsp
  where tse.subeventid = tsp.subeventid;
```

- *Meta condition*
  - *Query template*

    ```
    meta ( <meta_conditions> )
    ```

  - *SQL equivalent*

    ```
    select 0, length, videoid
      from TVIDEO
      where <meta_conditions>;
    ```

- *Object condition*
  - *Query template*

    ```
    odata ( <object_conditions> )
    ```

  - *SQL equivalent (when target is video)*

    ```
    select 0, length, videoid
      from TOBJECT
      where objectid in ( <object_conditions> );
    ```

  - *SQL equivalent (when target is segments of a video)*

```
select te.begintime, te.endtime, te.videoid
  from TEVENT te, TPLAYER tp
  where te.eventid = tp.eventid
    and objectid in ( <object_conditions> );
```

The addition of new subparse tree types for semantic conditions yields some changes in query tree construction algorithm. In the earlier version of the algorithm, when the system matches a subparse tree rooted by the current parse tree node with a subparse tree type, then it directly converts it to Prolog. For supporting subparse tree types representing semantic conditions, this part of the algorithm is changed as follows:

```
if pnode->type != connector
    if match with a subparse tree type for semantic condition
        data->content = SQL equivalent of the subparse tree
    else
        data->content = Prolog equivalent of the subparse tree

    return data
```

The earlier version of the query tree construction algorithm of BilVideo assumes that all the child query nodes hold Prolog equivalents of conditions given in the where clause of the original query. However, with the addition of semantic querying capability, the query nodes can also hold SQL equivalents of the conditions. This situation creates a problem in the connector rules for And-Or connectors. To handle this problem, the system should check before merging the condition types if they are of the same type or not. Thus, taking this situation into consideration, we updated the connector rules for the And-Or connectors. According to these changes, the system currently checks for the condition types and if they are of the same type, it applies the original rules. However, if they are different type or at least one of them is of mixed type, then the system applies the updated rules (The details of the changes in the And-Or connector rules are presented visually in Appendix B.2).

6.5 Query Tree Traversal

After the construction of the query tree, the system traverses it recursively in a depth first manner for executing the submitted query over the previously extracted information. The pseudo-code for the algorithm that is used for traversing the query tree is as follows:

```
processQTree
    input  = qnode (query tree node)
    output = result (keyframe intervals)

    if qnode->type is a connector node
        if qnode has a left child
            resultOfLeft = processQTree(qnode->leftChild)
        if qnode has a right child
            resultOfRight = processQTree(qnode->rightChild)

        // Call corresponding interval processing method
        result = processInterval(resultOfLeft, resultOfRight,
                                 qnode->getType)

    else if qnode is a query node
        if qnode holds a prolog query
            // Send query to the Prolog Engine
            result = prologQuery(qnode->content)
        if qnode holds a SQL query
            // Send query to the relational database
            result = sqlQuery(qnode->content)

    return result
```

During traversal of the query tree, when the system encounters a connector node, it gets the results form its child nodes. Then, it invokes the interval processing algorithm corresponding to the type of the connector node. At this point,

the system doesn't have to know about the types of its child nodes, because it gets the results from its child nodes in a specific format which doesn't depend on the type of the child nodes. Thus, we didn't have to modify the code for interval processing.

When the system encounters a query node, it checks if its content is a plain Prolog or a SQL query. If it is a Prolog query, then the system sends it to the Prolog engine, and if it is a SQL query, the system sends this to the relational database.

6.6 Database Connection

As it is explained before, we chose the Oracle RDBMS for storing the extracted semantic information. As our query processor code was written in C++, there are 3 possibilities for sending a SQL query to Oracle:

- OCI (Oracle Call Interface)
- ODBC driver
- ODBC Bridge

To be able to use the first two of those, it is needed to install a software called Oracle Client on the machine which runs the query processor. The size of the Oracle Client software is approximately 200MB and it is not an easy task to install it on a Linux or Unix server. The third option requires installation of a bridge software on both the client side where the query processor runs and the server side where Oracle RDBMS is installed. When we need to send a query in this architecture, it is first sent to the local bridge. The local bridge passes this query to the bridge located on the server side, and it is then sent to Oracle. Then the result of the query is sent back in the reverse path. This process is shown in Figure 6.6.

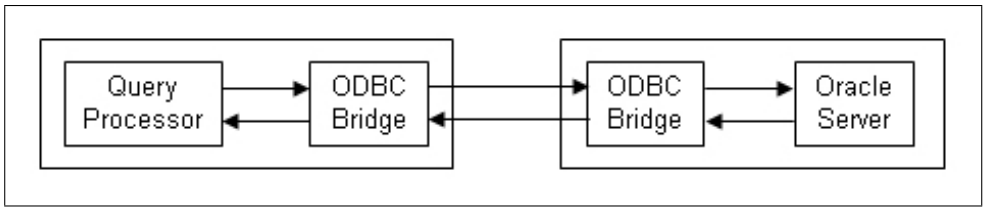

Figure 6.6: ODBC Bridge Solution

There are two problems about this architecture. One is the increased communication cost by the communication between bridges and the other is the installation of a bridge software on the server side which is not a desired thing because most of the database administrators will not allow to install a software on the server side due to the security concerns. Because of the problems of those choices, we develop our own solution for providing a connection from C++ code to Oracle. We wrote our own bridge software which only needs to be installed on the client side and which has a size of 4KB. We implemented the bridge software in Java language and it uses JDBC interface for connecting to Oracle. The java bridge is written as a multi threaded application so that it can serve more than one query processor at the same time. For sending the queries to this bridge we implemented an interface called CDBC in C++. This interface was implemented in such a way very similar to that of the JDBC interface which is very easy to use and very efficient in terms of the resource management. The architecture of this solution is shown in figure Figure 6.7 and a sample code for sending a query from C++ using the CDBC interface is presented in Figure 6.8.

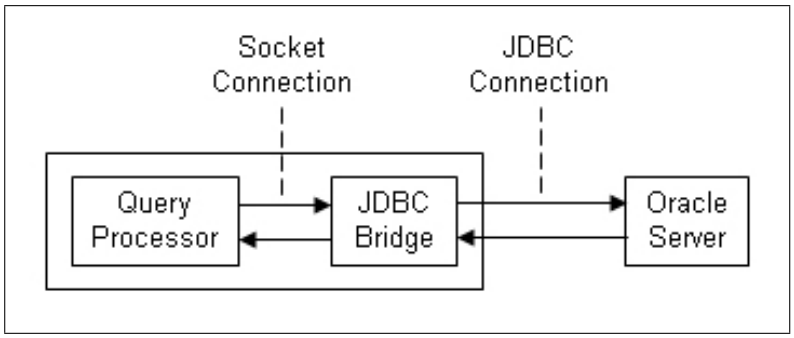

Figure 6.7: JDBC Bridge Solution

```
void main(){
    // Connect to Bridge
    Connection *conn = new Connection(4444,"100.100.100.4");

    // Connect to DB
    conn->createConnection("100.100.100.6","1521","PCVIDEO",
               "annotator","annotator");
    // Create a statement for sending query to the DB
    Statement *stmt = conn->createStatement();

    // Sample query = Get the names of the Videos in the DB
    // Send the query to the DB and get the result of the query
    ResultSet *rset = stmt->executeQuery("SELECT name FROM tvideo");

    // Print the results
    while(rset->next()){
        cout<<"Video Name="<<rset->getString("1");
    }

    // Release the resources
    rset->close();
    stmt->close();
    conn->close();
}
```

Figure 6.8: Sample code for sending a query to the database and getting the result of it

Chapter 7

Conclusion

As the attention on the video as a multimedia data type has increased by the time, the amount of research on it has also increased very rapidly. Semantic analysis of video data is one of the major topics investigated by the researchers in multimedia systems field. The main aspects that we have to deal with while working on semantic issues of video data can be given as follows:

- Defining a semantic video model for describing the contents of a video semantically.
- Extraction of semantic information from videos.
- Storing the extracted information.
- Querying the videos stored in the database.

In Bilkent Multimedia Database Group, a solution for each of these aspects was developed. First, a semantic model was defined. According to this model, a video is composed of events, events consist of subevents and there exist objects of interest that takes place in both events and subevents. A data model was also defined, which lets us keep the details of events, subevents and objects like the location and time of occurrence of an event, the objects that involved in an event or subevent, the details of an object such as the name, height or title of a

person and so on. By defining the video in such a comprehensive and detailed fashion, we increased the querying capabilities of the system considerably and in this work, we fully integrated the semantic querying capability to BilVideo.

There exist many automatic semantic information extraction methods proposed in the literature. But most of these methods do not perform good enough. It is also very hard to extract information automatically in the detail level of our model. Therefore, we have developed a semi-automatic information extraction tool. As a future work, we are planning to involve the automatic extraction techniques in our system such as automatic shot detection for easing the process of manual annotation.

By analyzing the detail level of the extracted semantic information and the relational nature of it, we decided to use a relational database to store this information. We designed a database model for storing the extracted data. We use a SQL-like query language for formulating semantic queries, which is integrated into the query language of BilVideo. Besides, we designed and integrated a GUI into the web based query interface of BilVideo for the specification of semantic queries visually.

Currently the semantic querying capability was fully integrated into BilVideo. The system is now capable of executing spatio-temporal, trajectory and semantic queries together. The users are able to submit complex, mixed and detailed queries to the system both visually and textually through we based query interface.

Bibliography

[1] M.E. Dönderler, Ö. Ulusoy and U. Güdükbay. A Rule-based Video Database System Architecture. *Information Sciences*, Vol. 143, No. 1-4, pp. 13-45, 2002.

[2] M.E. Dönderler, Ö. Ulusoy and U. Güdükbay. Rule-based Spatiotemporal Query Processing for Video Databases. *the VLDB Journal*, Vol. 13, No. 1, pp. 86-103, January 2004.

[3] U. Arslan. A Semantic Data Model and Query Language for Video Databases. *Masters Thesis Bilkent University Department of Computer Engineering*, 2002.

[4] E. Saykol. Web-Based User Interface for Query Specification in a Video Database System. *Masters Thesis Bilkent University Department of Computer Engineering*, 2001.

[5] F. Nack and W. Putz. Saying What it Means: Semi-Automated (News) Media Annotation. *Multimedia Tools and Applications*, 22, pp. 263-302, 2004.

[6] H. Kosch, A. Mostefaoui, L. Böszörmenyi, and L. Brunei. Heuristics for Optimizing Multi-Clip Queries in Video Databases. *Multimedia Tools and applications*, 22, pp. 235-262, 2004.

[7] A. Jain and S. Chaudhuri. A Fast Method for Textual Annotation of Compressed Video. *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing* (ICVGIP 2002), 2002.

[8] A. Mahindroo, B. Bose, S. Chaudhury and G. Harit. Enhanced Video Representation using Objects. *Proceedings of the Indian Conference on Vision, Graphics and Image Processing* (ICVGIP 2002), 2002.

[9] A. Ekin, A. M. Tekalp, and R. Methrotra. Integrated semantic-syntactic video event modeling for search and retrieval. *Proceedings of 2002 International Conference on Image Processing 2002* (ICIP'02), Vol. I, pp. 141-144, 2002.

[10] C. A. Goble, A. Carole, C. Haul, S. Bechhofer. Describing and classifying multimedia using the description logic GRAIL, *Proceedings of SPIE*, Vol. 2670, pp. 132-143, Storage and Retrieval for Still Image and Video Databases IV, Ishwar K. Sethi; Ramesh C. Jain; Eds. 1996.

[11] R. Hammoud, L. Chen and D. Fontaine. An Extensible Spatial-Temporal Model for Semantic Video Segmentation. *Proceedings of the First International Forum on Multimedia and Image Processing*, Anchorage, Alaska, 1998.

[12] T. S. Huang and M. R. Naphade. MARS (Multimedia Analysis and Retrieval System): A test-bed for video indexing, browsing, searching, filtering and summarization. *International Workshop on Multimedia Data Storage, Retrieval, Integration and Applications*, Hong Kong Polytechnic University, 2000.

[13] M. R. Naphade, I. Kozintsev, T. S. Huang and K. Ramchandran. A Factor Graph Framework for Semantic Indexing and Retrieval in Video. *Proceedings of the IEEE Workshop on Content-based Access of Image and Video Libraries* (CBAIVL'00), pp. 35-39, 2000.

[14] M. R. Naphade, T. S. Huang. A Probabilistic Framework for Semantic Indexing and Retrieval in Video. *Proceedings of the IEEE International Conference on Multimedia and Expo* (ICME'00), pp. 475-478, New York, 2000.

[15] M. R. Naphade, T. S. Huang. Semantic Video Indexing using a probabilistic framework. *Proceedings of the International Conference on Pattern Recognition* (ICPR'00), Vol. 3, pp. 3083, Barcelona, Spain, 2000.

[16] A. Bonzanini, R. Leonardi and P. Migliorati. Exploitation of Temporal Dependencies of Descriptors to Extract Semantic Information. *Proceedings of VLBV 2001*, pp. 177-180, Athens, Greece, 2001.

[17] A. B. Benitez, H. Rising, C. Jorgensen, R. Leonardi, A. Bugatti, K. Hasida, R. Mehrotra, A. M. Tekalp, A. Ekin and T. Walker. Semantics of Multimedia in MPEG-7, *Proceedings of the International Conference on Image Processing 2002* (ICIP'02), Vol. 1, pp. 137-140, 2002.

[18] A. Yao and J. Jin. The development of a video metadata authoring and browsing system in XML. *Proceedings of the Pan-Sydney Workshop on Visualisation*, Vol. 2, pp. 39-46, Sydney, Australia, 2000.

[19] D. A. Tran, K. A. Hua and K. Vu. Semantics Reasoning Based Video Database Systems. *Proceedings of Database and Expert Systems Applications*, pp. 41-50, 2000.

[20] W. Zhou, A. Vellaikal and C.-C. Jay Kuo. Rule-based Video Classification System for Basketball Video Indexing. *ACM Multimedia 2000*, pp. 213-216, Los Angeles, CA, USA, 2000.

[21] B. L. Tseng, C.-Y. Lin and J. R. Smith. Video Summarization and Personalization for Pervasive Mobile Devices. *Proceedings of SPIE Electronic Imaging 2002 - Storage and Retrieval for Media Databases*, pp. 359-370, San Jose, CA, USA, 2002.

[22] J. M. Corridoni and A. D. Bimbo. Film semantic analysis. *Proceedings of the International Conference on Computer Architecture for Machine Perception* (CAMP'95), pp. 202-209, Como, Italy, 1995.

[23] A. Yoshitaka, T. Ishii, M. Hirakawa, T. Ichikawa. Content-based retrieval of video data by the grammar of film. *Proceedings of the 1997 IEEE Symposium on Visual Languages* (VL '97), pp. 310-317, 1997.

[24] A. Yoshitaka, T. Ishii, M. Hirakawa, T. Ichikawa. Content-based retrieval of video data by the grammar of film. *Proceedings of the 1997 IEEE Symposium on Visual Languages* (VL '97), pp. 310-317, 1997.

[25] J. Assfalg, M. Bertini, C. Colombo, and A. Del Bimbo. Extracting semantic information from news and sport video. *Proceedings of the 2nd International Symposium on Image and Signal Processing and Analysis* (ISPA'01), pp. 4-11, 2001.

[26] B. Li and M.I. Sezan. Event Detection and Summarization in Sports Video. *Proceedings of the IEEE Workshop on Content-based Access of Image and Video Libraries* (CBAIVL'01) , pp. 132-138, 2001.

[27] N. Nitta, N. Babaguchi and T. Kitahashi. Story Based Representation for Broadcasted Sports Video and Automatic Story Segmentation. *Proceedings of the IEEE International Conference on Multimedia and Expo* (ICME'02), pp. 813-816, 2002

[28] A. Kojima, T. Tamura and K. Fukunaga. Textual description of human activities by tracking head and hand motions. *Proceedings of the 16th International Conference on Pattern Recognition*, Vol. 2, pp. 1073-1077, 2002.

[29] P. H. Meland, J. Austvik, J. Heggland and R. Midtstraum. Using Ontologies and Semantic Networks with Temporal Media. *Proceedings of the SIGIR'2003 Semantic Web Workshop*, Toronto, Canada, 2003.

[30] World Wide Web Consortium. Resource Description Framework (RDF). http://www.w3.org/RDF/.

Appendix A

Database Table Specifications

TVIDEO Stores bibliographic information about videos. `videotype` and `audience` values are references to `TVIDEOTYPE` and `TAUDIENCE` tables.

Column Name	Data Type	Constraints
VIDEOID	Number(10)	Primary Key
NAME	Varchar2(20)	
LENGTH	Number(10)	Check(Length >= 0)
PYEAR	Number(10)	
PRODUCER	Varchar2(20)	
DIRECTOR	Varchar2(20)	
VIDEOTYPE	Number(10)	References TVIDEOTYPE(VIDEOTYPEID)
AUDIENCE	Number(10)	References TAUDIENCE(AUDIENCEID)
SUBJECT	Varchar2(20)	
VIDEOURL	Varchar2(20)	

TVIDEOTYPE Stores video type names, like adventure, horror, science-fiction, romance, etc.

Column Name	Data Type	Constraints
VIDEOTYPEID	Number(10)	Primary Key
VIDEOTYPENAME	Varchar2(20)	

TAUDIENCE Stores audience names like, teenager, children, adult, everyone, etc.

Column Name	Data Type	Constraints
AUDIENCEID	Number(10)	Primary Key
AUDIENCENAME	Varchar2(20)	

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

Column Name	Data Type	Constraints
EVENTID	Number(10)	Primary Key
VIDEOID	Number(10)	References TVIDEO(VIDEOID)
ACTIVITY	Number(10)	References TACTIVITY(ACTIVITYID)
BEGINTIME	Number(10)	
ENDTIME	Number(10)	
LOCATION	Varchar2(20)	
TIMEOFEVENT	Varchar2(20)	

TSUBEVENT Stores data about subevents. Subactivity, begin and end times are stored. `eventid` and `subactivity` are references to `TEVENT` and `TSUBACTIVITY` tables.

Column Name	Data Type	Constraints
SUBEVENTID	Number(10)	Primary Key
EVENTID	Number(10)	References TEVENT(EVENTID)
SUBACTIVITY	Number(10)	References TSUBACTIVITY(SUBACTIVITYID)
BEGINTIME	Number(10)	
ENDTIME	Number(10)	

TPLAYER Stores the objects that appear in events plus their roles in the events. Objects can have many roles in an event. `eventid` and `objectid` fields are references to `TEVENT` and `TOBJECT` tables respectively.

Column Name	Data Type	Constraints
PLAYERID	Number(10)	Primary Key
EVENTID	Number(10)	References TEVENT(EVENTID)
OBJECTID	Number(10)	References TOBJECT(OBJECTID)

TPLAYERROLE Stores the roles of the players in the events.

Column Name	Data Type	Constraints
PLAYERID	Number(10)	Primary Key
ROLEID	Number(10)	Primary Key

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

Column Name	Data Type	Constraints
SUBPLAYERID	Number(10)	Primary Key
SUBEVENTID	Number(10)	Primary Key

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

Column Name	Data Type	Constraints
OBJECTID	Number(10)	Primary Key
VIDEOID	Number(10)	References TVIDEO(VIDEOID)
NAME	Varchar2(20)	

TACTIVITY Stores activity names, like party, wedding, murder, war, etc.

Column Name	Data Type	Constraints
ACTIVITYID	Number(10)	Primary Key
ACTIVITYNAME	Varchar2(20)	

TROLE Stores rolenames for each activity. For example, for the murder activity the role names are murderer and victim; and for the party activity the role names are host and guest.

Column Name	Data Type	Constraints
ROLEID	Number(10)	Primary Key
ACTIVITYID	Number(10)	References TACTIVITY(ACTIVITYID)
ROLENAME	Varchar2(20)	

TSUBACTIVITY Stores actions such as talking, eating, dancing, fighting, etc.

Column Name	Data Type	Constraints
SUBACTIVITYID	Number(10)	Primary Key
SUBACTIVITYNAME	Varchar2(20)	

TATTRIBUTE Stores attribute names for video objects like realname, sex, color, speed, etc.

Column Name	Data Type	Constraints
ATTRIBUTEID	Number(10)	Primary Key
NAME	Varchar2(20)	

TOBJECTATTRIBUTE Stores the attribute values for each attribute defined for each object. `objectid` and `attributeid` are references to `TOBJECT` and `TATTRIBUTE` tables.

Column Name	Data Type	Constraints
OBJECTID	Number(10)	References TOBJECT(OBJECTID)
ATTRIBUTEID	Number(10)	References TATTRIBUTE(ATTRIBUTEID)
AVALUE	Varchar2(20)	

Appendix B

Query Tree Construction Rules

B.1 Earlier Version of AND-OR Rules without Semantic Queries

During the traversal of parse tree for query tree construction, when a connector node representing "AND" or "OR" is encountered, AND-OR rules are applied for that node by considering the results coming from its child nodes.

AND-OR rules are applied in two phases. In the first phase, the results coming from the child nodes are reorganized to make the application of the AND-OR rules easier. In the second phase, AND-OR rules are applied to the reorganized results coming from child nodes.

B.1.1 Phase1 - Reorganization

Input to Phase1 is the results coming from the child nodes. The results can be a root node of a subtree and/or a textual content which contains a subquery. Output of Phase1 is also a root node of a subtree and/or a textual content which contains a subquery. There can be six different input combinations. These input

combinations and their corresponding outputs are shown in Figures B.1-B.6.

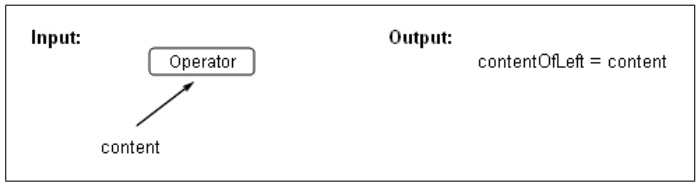

Figure B.1: Only content is coming from left child

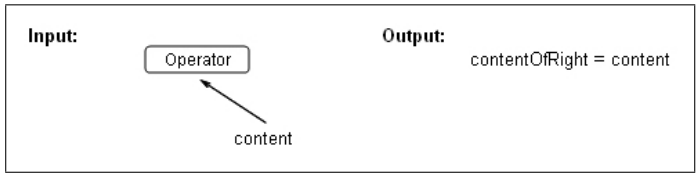

Figure B.2: Only content is coming from right child

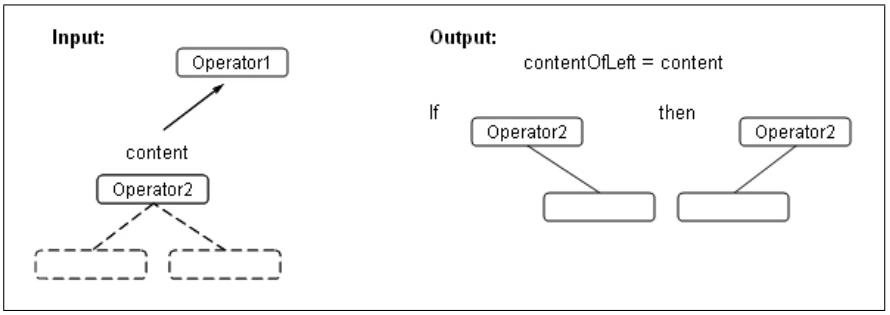

Figure B.3: Content and node (same with the current operator) coming from left child

B.1.2 Phase2 - Query Tree Construction

Input to Phase2 is the reorganized results coming the child nodes. The results can be a root node of a subtree and/or a textual content which contains a subquery. Output of Phase2 is also a root node of a subtree and/or a textual content which contains a subquery. There can be seven different input combinations. These input combinations and their corresponding outputs are shown in Figures B.7-B.13.

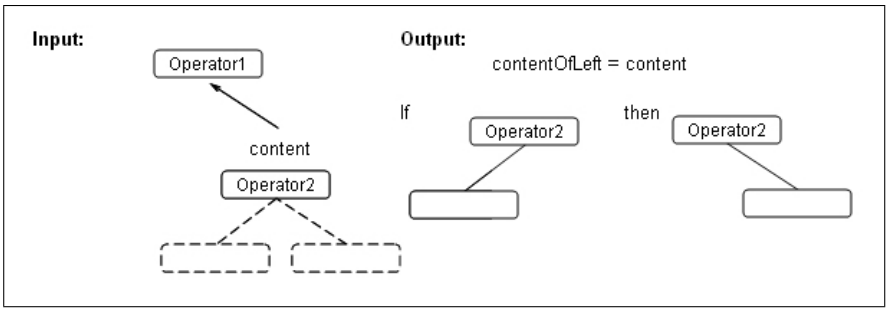

Figure B.4: Content and node (same with the current operator) coming from right child

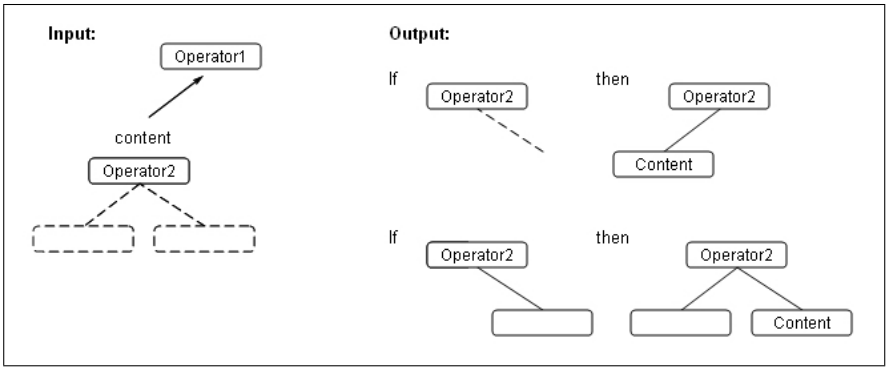

Figure B.5: Content and node (different than the current operator) coming from left child

B.2 Updated AND-OR Rules with Semantic Queries

The AND-OR rules for query tree construction need some modifications, when the nodes for semantic queries come into the picture. The updated AND-OR rules for supporting semantic queries are explained in this chapter.

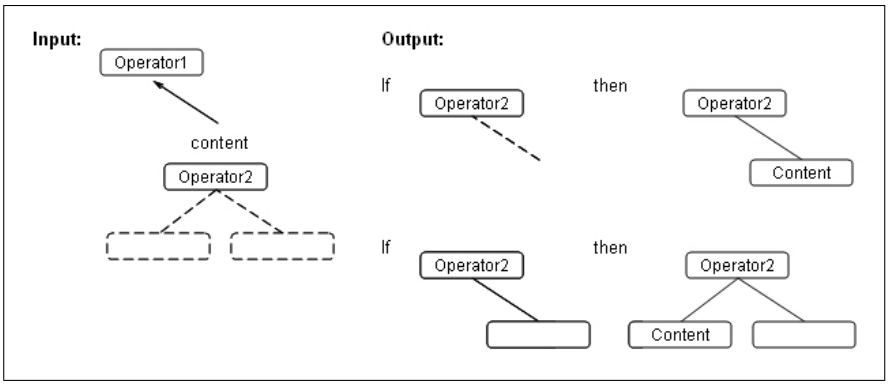

Figure B.6: Content and node (different than the current operator) coming from right child

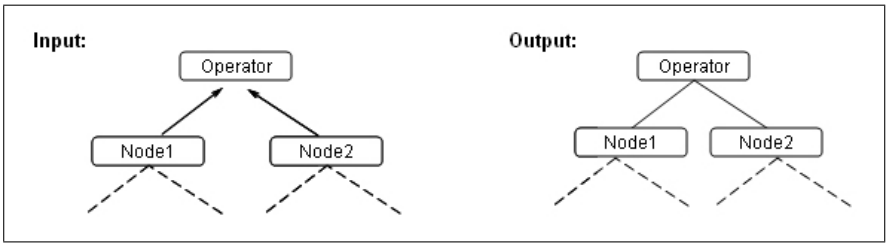

Figure B.7: Only nodes come from both child

B.2.1 Phase1* - Different Child Types

Input to Phase1* is the results coming from the child nodes. Child nodes should belong to different query types. The results can be a root node of a subtree and/or a textual content which contains a subquery. Output of Phase1* is also a root node of a subtree and/or a textual content which contains a subquery. The input combinations and their corresponding outputs are shown in Figure B.14.

B.2.2 Phase2* - Different Child Types

Input to Phase2* is the reorganized results coming the child nodes. Child nodes should belong to different query types. The results can be a root node of a subtree

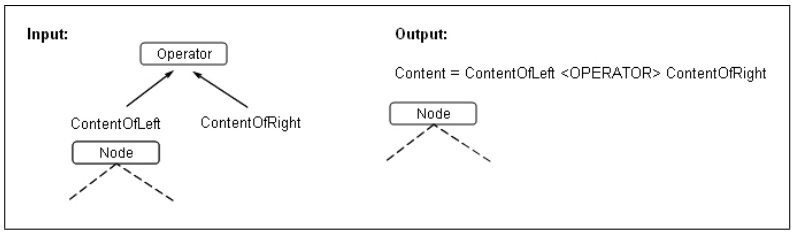

Figure B.8: Content and node coming from left child and content coming from right child

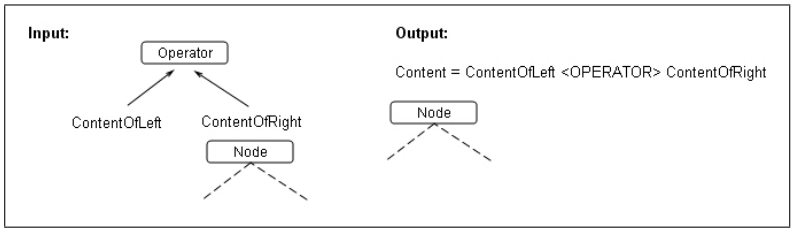

Figure B.9: Content coming from left child and content and node coming from right child

and/or a textual content which contains a subquery. Output of Phase2* is also a root node of a subtree and/or a textual content which contains a subquery. The input combinations and their corresponding outputs are shown in Figure B.15 and B.16.

B.2.3 Phase2** - At Least One Mixed Child Type

Input to Phase2** is the reorganized results coming the child nodes. Query type of at least one of the child nodes should be mixed. The results can be a root node of a subtree and/or a textual content which contains a subquery. Output of Phase2** is also a root node of a subtree and/or a textual content which contains a subquery. The input combinations and their corresponding outputs are shown in Figure B.17 and B.18.

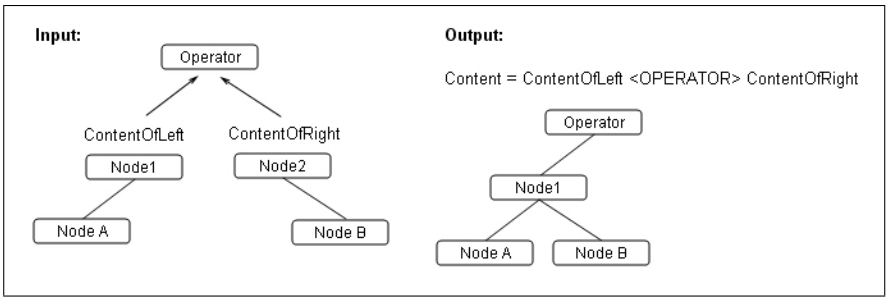

Figure B.10: Content and node coming from both children

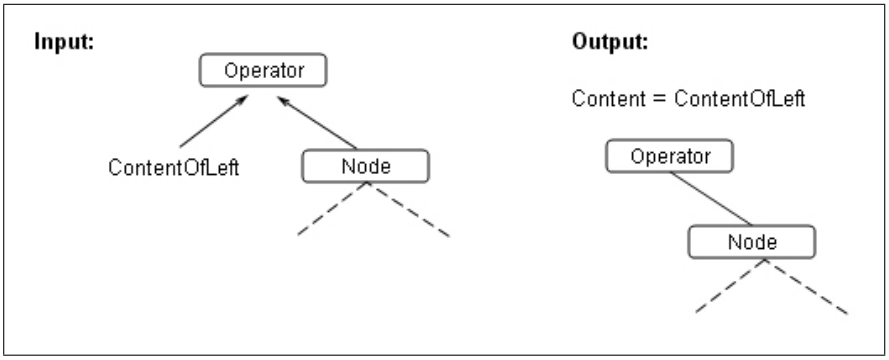

Figure B.11: Content from left child and node coming from right child (opposite child result order for input yields opposite output)

B.3 Parenthesis Rules

During the traversal of parse tree for query tree construction, Parenthesis rules are applied for that node by considering the results coming from its child node when a node representing a parenthesis is encountered. The input combinations and their corresponding outputs are shown in Figure B.19 and B.20.

B.4 NOT Rules

During the traversal of parse tree for query tree construction, NOT rules are applied for that node by considering the results coming from its child node when

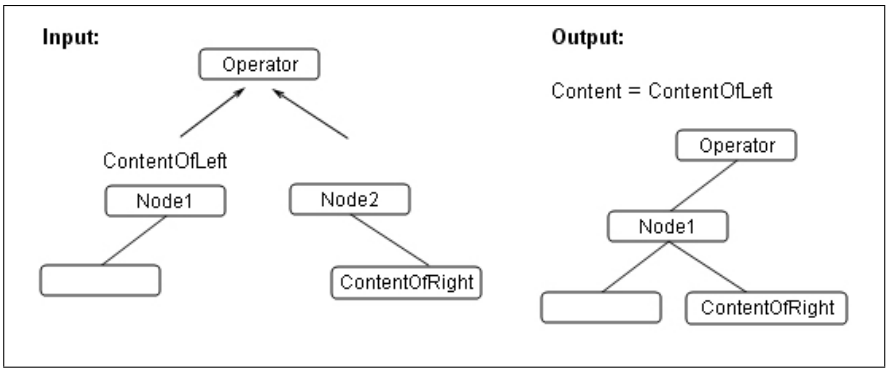

Figure B.12: Content and node coming from left child and node (left child empty) coming from right child (opposite child result order for input yields opposite output)

a node representing a "NOT" operator is encountered. The input combinations and their corresponding outputs are shown in Figure B.21 and B.22.

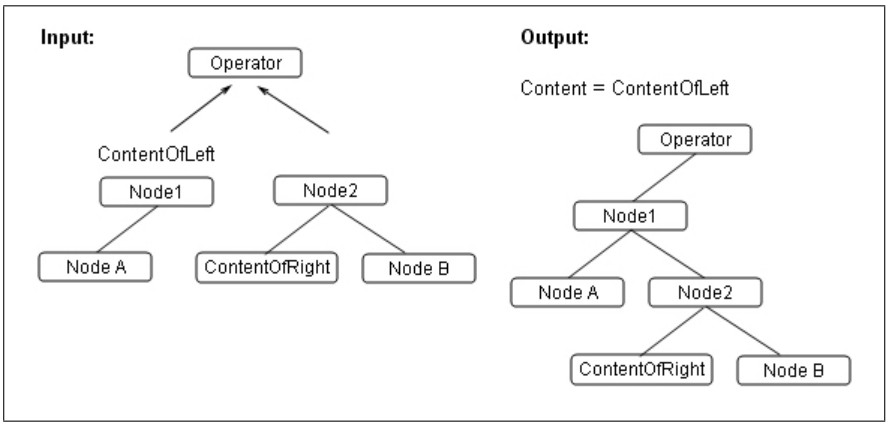

Figure B.13: Content and node coming from left child and node coming from right child (opposite child result order for input yields opposite output)

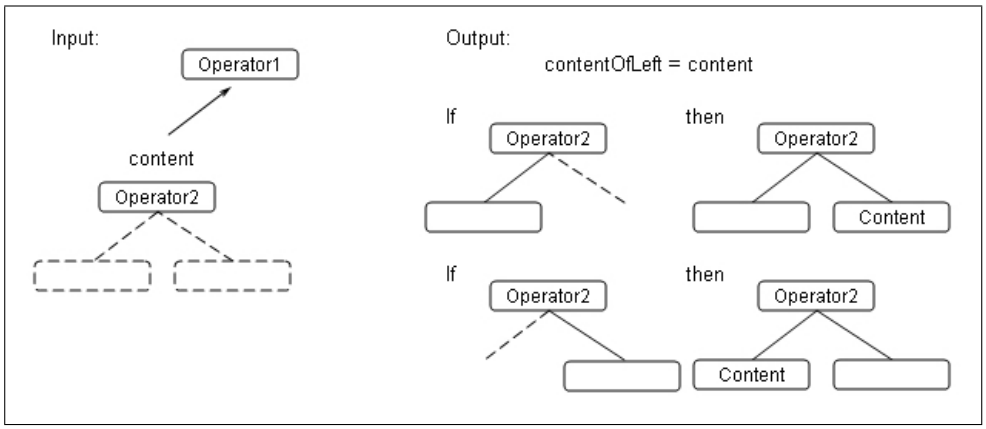

Figure B.14: Content and node (same with the current operator) coming from left child (opposite child result order for input yields opposite output)

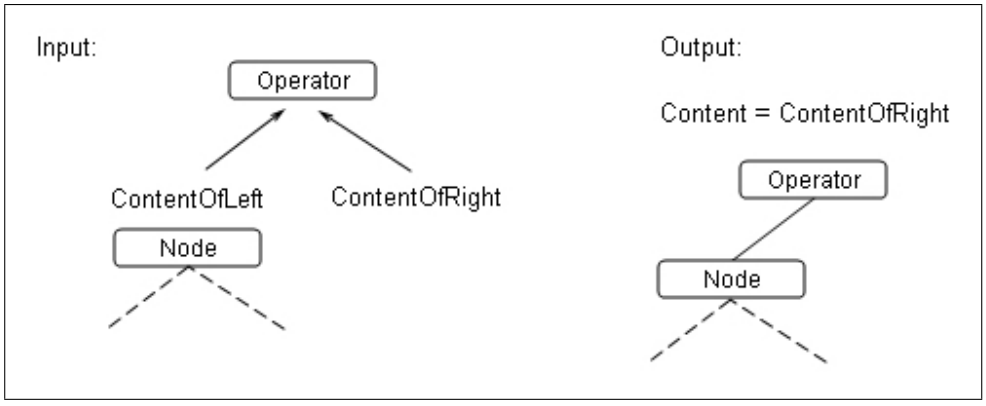

Figure B.15: Content and node coming from left child and content coming from right child (opposite child result order for input yields opposite output)

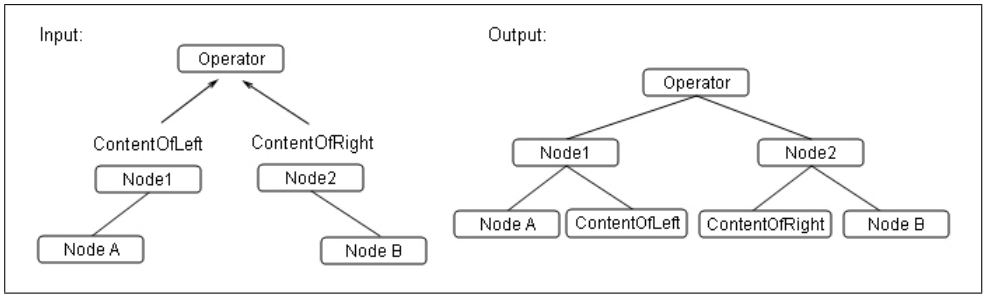

Figure B.16: Content and node coming from both children

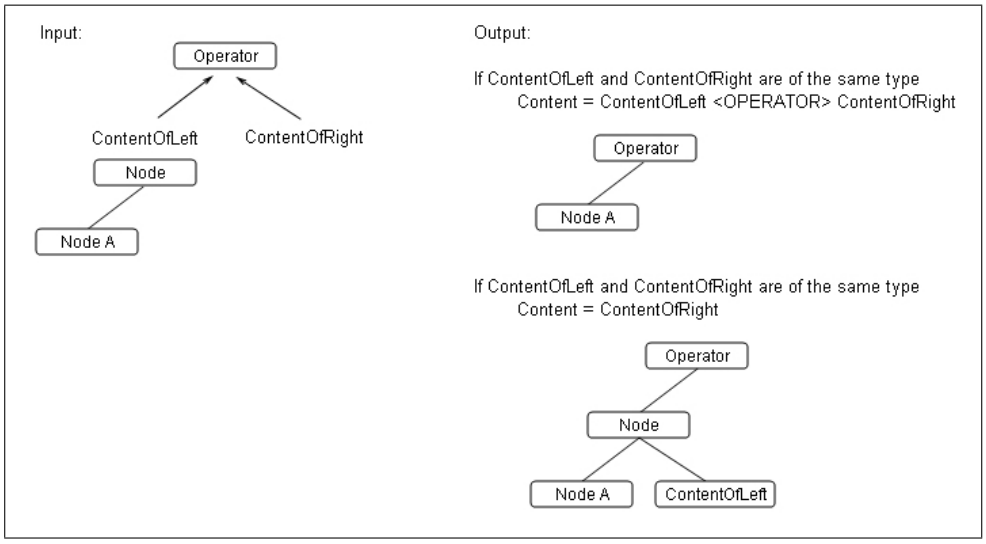

Figure B.17: Content and node coming from left child and content coming from right child (opposite child result order for input yields opposite output)

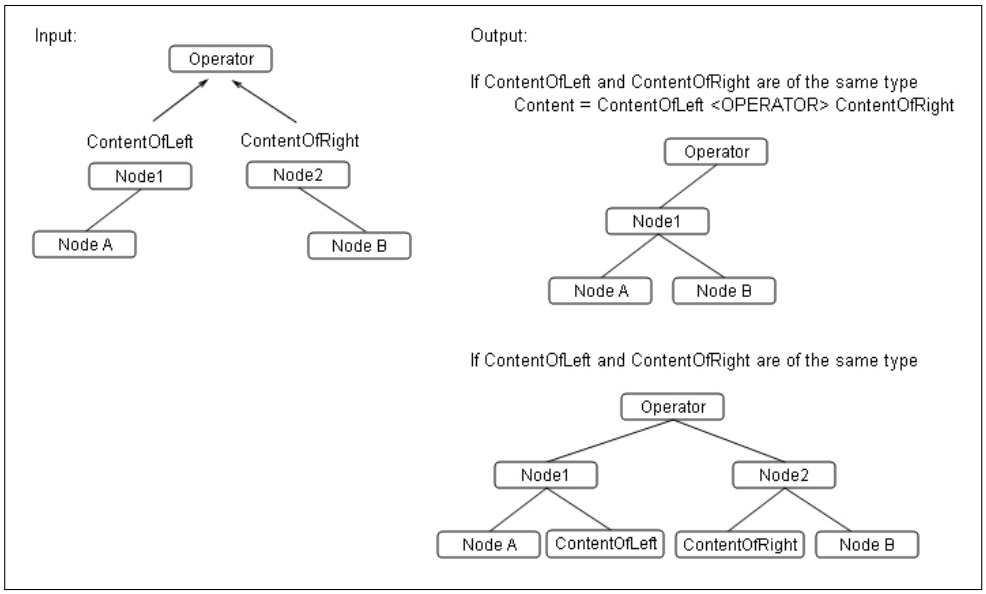

Figure B.18: Content and node coming from both children

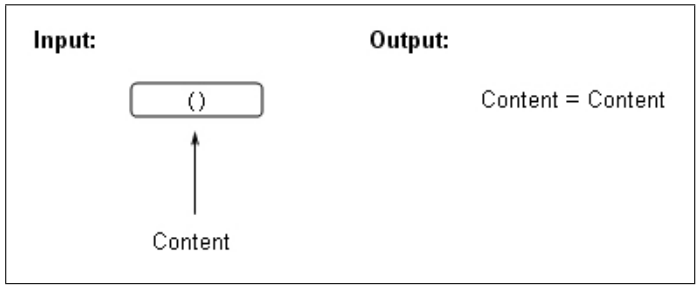

Figure B.19: Content coming from child

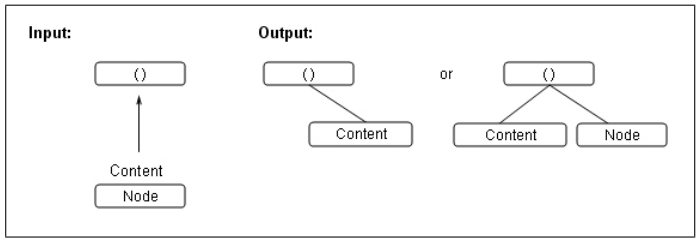

Figure B.20: Content and node coming from child

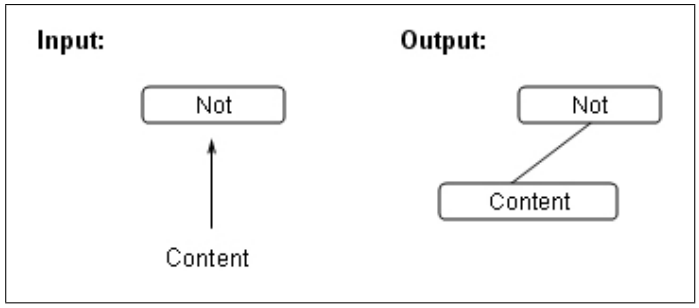

Figure B.21: Content coming from child

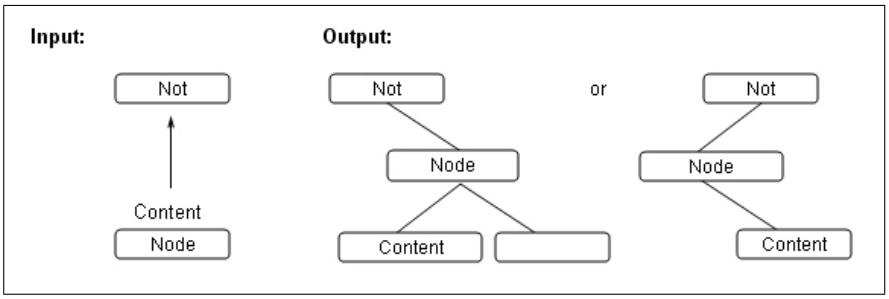

Figure B.22: Content and node coming from child