University of Alberta

Modeling and Querying Multimedia Data

by

John Z. Li
Laboratory for Database Systems Research
Department of Computing Science
University of Alberta
Edmonton, Alberta
Canada T6G 2H1
zhong@cs.ualberta.ca

Technical Report TR 98–05
March 1998

DEPARTMENT OF COMPUTING SCIENCE
The University of Alberta
Edmonton, Alberta, Canada

Modeling and Querying Multimedia Data

Abstract

Amidst the dramatic developments of computer technology in the last decades, we have seen a growing interest in supporting multimedia data. Compared with traditional data, the most noticeable features of multimedia data are their *spatiality* (the spatial layout of objects and their relationships), *temporality* (time-dependence in media such as video and audio), and *presentation* requirements (imposed in response to user queries, such as synchronization of different media). Developing database management support for multimedia data poses interesting challenges.

Many multimedia database management systems have been developed to deal with these multimedia features. However, they suffer some common problems: lack of generality in media modeling, lack of precise knowledge of spatial information, limited spatio-temporal functionality, ineffective query languages, and inefficient query processing. The main purpose of this work is to investigate these problems and provide solutions. The first step of the research is to design a multimedia data model which is built upon and extends existing models. The proposed model is based on a temporal interval algebra. The establishment of a multimedia data model provides a theoretical foundation that is used to investigate multimedia query languages. The result is a general-purpose, object-oriented multimedia query language which supports content-based information retrieval. A prototype has been built to validate the proposed concepts.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

In the past few years, there has been a growing trend in the computer industry to provide support for multimedia data. Many of the new computer applications involve multimedia data. In the most general setting, *multimedia data* could mean arbitrary data types and data from arbitrary sources [Kim95]. In addition to traditional data types like numeric data and character strings, multimedia data includes graphics, images, audio and video.

Why are database techniques necessary to manage multimedia data? Many multimedia systems or media servers are implemented using *multimedia file systems*. These systems leave to the user the responsibility of formatting the files for multimedia objects as well as the management of large amounts of data. The development of multimedia computing systems can benefit from traditional database management system (DBMS) services such as data independence (data abstraction), high-level access (query languages), application neutrality (openness), controlled multi-user access (transactions, concurrency control), fault tolerance (recovery), and authorization (access control).

Another important reason for using a DBMS is that the existence of temporal and spatial relationships, together with the related data output synchronization requirements complicates data management and delivery. These relationships need to be modeled explicitly as part of the stored data. Thus, even if the multimedia data are stored in files, their relationships need to be stored as part of the meta-information in a DBMS.

The most distinguishable differences between a Multimedia DBMS (MDBMS) and a traditional DBMS are the following:

- *Spatiality*: Some media have spatial properties which must be captured by MDBMSs;
- *Temporality*: Some media, such as audio and video, are heavily dependent on their temporal characteristics and temporal relationships among them;

- *Presentation*: The query result must be presented according to special presentation requirements, such as synchronization of different media.

As a result, most current research focuses on the above areas. While traditional DBMSs, such as relational DBMS, are not suitable for multimedia DBMS, an object-oriented approach is an elegant basis for addressing all data modeling requirements of multimedia applications [WK87, CK95]. Thus, almost all MDBMSs are directly or indirectly (by extending relational models into object-oriented models) based on object-oriented technology.

Many multimedia models, including query models, have been proposed [Mas91, DG92, LAF+93, LG93, OT93, GBT94, WDG94, HJW95, IB95, MS96b, ABL95, DDI+95, ACC+96, IKO+96]. Some common problems are the following:

- They lack generality to support a broad range of media. Many systems are designed either to handle just one medium, e.g., images, or target certain applications such as medical imaging or geographical information systems.

- They lack precise knowledge of spatial information. Many MDBMSs have only the Minimum Bounding Rectangle (MBR) representation which is not suitable for certain applications.

- They lack a sufficiently rich set of temporal and spatial operators to support a broad range of content-based information retrieval. Some systems may have support for only certain temporal or spatial relationships. For example, an MDBMS should arguably support both *directional* and *topological* relationships[1].

- They lack knowledge representation to support temporal and spatial reasoning. Such reasoning can greatly enhance system efficiency by reducing the space required, especially in a distributed environment because higher cost is expected in shipping data over a network.

- They lack uniform temporal and spatial operations in their query languages (if one exists) to support query result presentation. The uniform processing of spatio-temporal operators in both the data model and the query model is necessary in order to provide a uniform user interface and to achieve system efficiency.

The major goal of this research is to investigate these problems and to propose solutions. The research work is part of a distributed image database (DISIMA) project [OÖL+97]. Although the description of the proposed model is based on TIGUKAT [ÖPS+95], any Object DBMS can be used. TIGUKAT is an object database management system (ODBMS) that is under development at the Laboratory for Database Systems Research at the University of Alberta. TIGUKAT has a novel object model whose identifying characteristics include a purely behavioral semantics and

[1] *Directional relations* describe order in space (e.g., south, northwest); *topological relations* describe neighborhood and incidence (e.g., overlap, disjoint).

a uniform approach to objects. Everything in the system is a first-class object with well-defined behavior. The computational model is one of applying behaviors to objects. A query model has been developed for TIGUKAT that is complete with a formal object calculus, an equivalent object algebra and an object SQL (Structured Query Language [ANSI86]). Major reasons for choosing TIGUKAT are its behavioral uniformity, object-oriented features, ease of use, and the availability of its designers and developers.

1.2 Scope and Contributions

This thesis describes a model and query language for MDBMSs using object-oriented techniques. In the modeling part, the focus is on finding an efficient way to represent image spatial information, to segment video data, and to model *salient objects* in both videos and images. Salient objects are defined as interesting physical objects in media. Since object-oriented technology is generally accepted as a promising tool for modeling multimedia data [Kim95, KA95], all the work here assumes the existence of an ODBMS. In the querying part, the focus is on defining a general-purpose query language which is capable of exploring the features presented in the model. A functional and predicative approach is used to achieve this goal. In any case, the central focus is always around temporality and spatiality.

Since "multimedia database" is such a broad topic, many aspects related to multimedia databases are not addressed in this thesis. These include multimedia database population (development of automatic image and video analyzing tools), audio data, multimedia query optimization, multidimensional indexing, and distributed data management issues.

1.2.1 Spatial Aspects

Most reported multimedia database systems have used minimum bounding rectangles (MBR) as the underlying spatial representation. While this was acceptable in early MDBMS development, it is no longer sufficient as MDBMSs are applied to more sophisticated applications, such as satellite images and medical images. In these cases, precise spatial information must be maintained. Therefore, a two tier model is necessary to efficiently process both simple and sophisticated spatial queries.

The point-set technique has been used in spatial databases [EF91, PSV96], especially in defining *topological relations*. The description of topological spatial relations in terms of topologically invariant properties of point-sets is fairly simple. As a consequence, the topological spatial relation between two point-sets may be determined with little computational effort. However, point-set approaches are notorious for large storage space requirements. Directly extracting point-sets from raw images is too expensive and almost impossible in practice. On the other hand, according to some research [SC96, WJ97], the size of raw images can be reduced without losing their spatial relations. By using the techniques described in [LÖ98] over the reduced images, it is possible to store the precise geometrical shapes and locations of salient objects with much less space. However, even with the big space savings of the new approach [LÖ98], huge storage for salient objects and expensive computation for spatial relationships are still required. A dramatic approach is to use MBRs to approximate salient objects. The advantage of using MBRs is that only two points are necessary for a salient object regardless of its size. This is not only reduces the storage requirement, but also simplifies the computation of spatial relationships. At a higher level, a unique representation of spatial objects and a complete set of definitions for both topological and directional relations are introduced. Such a representation is based on Allen's temporal interval algebra [All83]. This work is further enhanced by a rich set of spatial inference rules in order to fully support complex spatial relationships between objects.

In general, the proposed system adopts a two-step spatial processing procedure which has been proven to be efficient and effective [BKSS94, PSV96]. Whenever a query requires objects satisfying precise spatial relations, the system first finds all the objects which satisfy the required relations based on the objects' MBRs. This is because the computation of spatial relations based on MBRs is very fast. However, the result of the first step may contain *false hits*, i.e., some objects are included in the result because of the MBR approximation. Certain restrictions have to be applied in this two-level processing, such as requiring objects to be convex, to avoid wrong answers. Therefore, non-convex objects have to be decomposed into a set of convex objects in the model. In the second step, the precise spatial information is used to compute the required relations. This step is computationally expensive, but it deals with far fewer objects than the first step (hopefully). The spatial model supports both quantitative and qualitative spatial queries.

1.2.2 Temporal Aspects

A formalization of an abstract multimedia model, called Common Video Object Tree (CVOT), is proposed. Unique features of the CVOT, compared with others, are its flexibility in organizing video data and its incorporation of temporal relationships among video objects. As a consequence, it allows native support for a rich set of temporal multimedia operations. Since TIGUKAT has a powerful temporal model [Gor98, GLÖS96], a comprehensive video type system has been developed based on this temporal model. A key of this type system is using TIGUKAT's type *history* to model videos, clips, and frames, as well as salient objects. Therefore, a uniform interface is established in modeling different aspects of video data. Furthermore, a greedy algorithm which can automatically construct a CVOT for a given video is developed. The time complexity of the algorithm is also analyzed.

1.2.3 Modeling and Querying Aspects of Moving Objects

The most striking difference between still images and videos stems from movements and variations. Retrieving moving objects, which requires both spatial and temporal knowledge of objects, is part of content-based querying. Modeling moving objects provides an effective way of exploring the combined features of the proposed spatial and temporal models. Numerous ways [DG94, IB95, SAG95, YYHI96] have been proposed in the literature for tracking the movement of a single object, i.e., the *trajectory* of an object over a period of time. To the best of my knowledge, no research has addressed the relationships between moving objects.

A new way of qualitatively representing the trajectory of a moving object is proposed. Pioneering work has been done in exploring the relative spatio-temporal relations between moving objects. Furthermore, algorithms for matching trajectories and spatio-temporal relations of moving objects are designed to facilitate query processing. These algorithms can handle both exact and similarity matching. The resulting system supports a broad range of user queries about moving objects.

1.2.4 Query Language Aspects

The definition of a formal multimedia model completes the initial step of this research and establishes a platform from which a systematic and consistent investigation of query languages is conducted. Powerful query languages significantly help simplify multimedia database access. These languages must provide constructs for querying multimedia data, based on the structure of multimedia data as well as their contents. Furthermore, *query presentation*, which refers to the way query results are presented, is more complex in multimedia systems than in traditional DBMSs. This is because multimedia presentations have to take into account the synchronization of various media.

Although structured documents are easier to manage and exchange, they are not easy to query for ordinary users. The reason is that a user must know the structure of the documents, i.e., the schema of the database. As a consequence, much research in querying structured documents is focused on helping users query the database without precise (or complete) knowledge of document structure. This is called *querying without precise (or complete) knowledge*. Several approaches [BRG88, CACS94] have been proposed to address this problem, but the procedural evaluation of queries allowing non-precise specifications is not clear. The extended query language proposed in this thesis allows users to query structured documents, without precise knowledge of their structure, by using a *functional approach*.

Although, graphical user interfaces have been recognized as an important way of querying MDBMSs, many multimedia queries are best expressed in the traditional text-based manner because it is simple and precise, e.g., "*find all the images in which* John *is to the left of* Paul". Furthermore, a programmable interface to MDBMSs requires an embedded text-based query language. Therefore, an object-oriented, general-purpose query language, called Multimedia Object Query Language (MOQL), is proposed based on Object Query Language (OQL) [Cat94]. Features of MOQL include general applications, content-based queries, a rich set of spatio-temporal operators, query presentations, and querying without complete knowledge.

1.2.5 Implementation Aspects

The final step is to build a prototype from the proposed model and language. It is very difficult, if not impossible, to establish the validity of a new system without an implementation. Such an implementation should provide some insightful experience and feedback for improving the model and the language. A proof-of-concept prototype has been built using ObjectStore — a commercial ODBMS [LLOW91].

The prototype focuses on spatial attributes and is currently loaded only with images. Since videos can be viewed as a set of representative frames (or still images) which are selected from a set of video shots (or clips), the extension to video data is straightforward. An important feature of this prototype is the introduction of *logical salient objects* and *physical salient objects*. *Logical salient objects* describe attributes independent of the underlying media while *physical salient objects* describe attributes dependent on the underlying media. Such a separation of logical and physical attributes directly benefit the implementation of the type system and the query processing.

The whole system consists of C++ code (the core system), Lex/Yacc code (parsing MOQL), Java code (the graphical user interface), and C code (generating MOQL query trees and setting the web communication between the Java user interface and the core system). Currently the system is functional and can be accessed over the World Wide Web (WWW).

1.3 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 presents the proposed spatial model which includes the two-level representations. The lower level uses a point-set approach to capture an object's precise spatial properties, while the higher level uses an MBR to approximate the object's spatial properties. The MBR representation is based on a temporal interval algebra which is supported in the temporal model. The higher level captures both directional and topological relations. The spatial model is further inhanced by supporting a set of spatial reasoning rules and supporting the spatial similarity.

Chapter 3 introduces the CVOT model which captures the temporality of videos. It is used to model video semantic features such as *salient objects* and *activities*. The integration of the CVOT model into TIGUKAT is also described.

Chapter 4 presents an innovative way of modeling moving objects. This chapter serves as an empirical example of the combined application of the proposed temporal and spatial models. Some exact-matching and similarity-matching algorithms for moving objects are designed.

Chapter 5 examines the open issues of multimedia query languages and presents MOQL whose complete syntax is specified in Appendix D. The temporal, spatial, presentation, and structured document extensions are discussed in detail. Query examples are presented to explain the usage of MOQL. Its formal semantics are also presented briefly, with details contained in Appendix F.

Chapter 6 outlines the current status of the system implementation. It provides the background and reasons for some of the design choices. Selected classes, methods, and relationships are discussed in detail.

Chapter 7 summarizes the research work and major contributions of this thesis. It also points out possible future work.

There are seven appendixes: Appendix A gives a detailed description of spatial relation definitions; Appendix B sketches a proof for a topological relation theorem; Appendix C explains how to compute different factor effects; Appendix D describes the formal specifications of MOQL; Appendix E lists all the algorithms for MOQL predicates and functions; Appendix F discusses the expressiveness of MOQL; and Appendix G proves the correctness of the spatial reasoning rules.

Chapter 2

The Spatial Model

Many applications depend on spatial relationships in multimedia information systems. There is significant research on spatial relationships in image databases and geographic information systems (GIS) [RFS88, EF91, AEG94, CSE94, PS94, SYH94, Ege94, NSN95, PTSE95]. A spatial model is an essential part of an abstract MDBMS model which can be used as the basis for declarative queries. The research of spatial modeling is conducted within the DISIMA project [OÖL+97]. The DISIMA project aims at building a complete image database system enabling content-based querying. It allows users to assign different semantics to an image component (semantic independence) and to change an image representation without any effect on applications (representation independence). Furthermore, DISIMA provides an interoperable architecture to allow querying multiple and possibly remote image repositories through special DISIMA wrappers.

Information about the spatial semantics of multimedia must be structured so that indexes can be built to efficiently retrieve data from an MDBMS. *Spatial properties*, pertaining to spatial-oriented objects in a database, including points, lines, polygons, and volumes. Spatial relations have been classified [PE88] into several types, including *topological relations* that describe neighborhood and incidence (e.g., overlap, disjoint); *directional relations* that describe order in space (e.g., south, northwest); and *distance relations* that describe ranges of distance between objects (e.g., far, near). These types of spatial relations have been studied independently as well as in association with each other. Attention is payed to the first two types, i.e., topological and directional relations, because distance relations are domain dependent.

One of the most important issues in modeling spatial relationships is how to handle spatial queries. The special requirements of multimedia query languages in supporting spatial relationships have been investigated within the context of specific applications such as image databases and GISs [RFS88]. From a user's perspective, the following requirements are necessary for supporting spatial queries in an MDBMS:

- Support should be provided for object domains which consist of *complex* (structured) spatial objects in addition to simple (unstructured) points and alphanumeric domains. References to these spatial objects through their spatial domains must be directed by pointing to or describing the space they occupy, and not by referencing their encodings.

- Support should exist for *direct spatial searches*, which locate the spatial objects in a given area of images. This can resolve queries of the form "*Find all the faces in a given area within an image*".

- It should be possible to perform *hybrid spatial search*, which locates objects based on some attributes and some associations between attributes and the spatial objects. This can resolve queries of the form "*Display the person's name, age, and an image in which he/she is riding on a horse if the person is wearing blue jeans*".

- Support should exist for *complex spatial searches*, which locate spatial objects across the database by using set-theoretic operations over spatial attributes. This can resolve queries of the form "*Find all the roads which pass through city X*" where one may need to get the location coordinates of city X and then check road maps to see which ones contain the coordinates.

- Support should be provided to perform *direct spatial computations*, which compute specialized simple and aggregate functions from the images. This can resolve queries of the form "*Tell me the area of this object and find another object whose area is closest to this one*".

- Finally, support should exist for *spatio-temporal queries* which involve not only spatial relations, but temporal relations as well. This can resolve queries of the form "*Find a clip in which a dog is approaching someone from the left*".

In the proposed spatial model, minimum bounding rectangle (MBR) is used to approximate salient objects and their spatial relationships are defined in terms of temporal intervals [All83]. This MBR approach is further improved by using a revised point-set approach that improves space savings over a previously proposed similar approach [EF91]. The point-set approach is necessary for applications where MBR approximation is not sufficiently precise. The features of the proposed spatial model are [LÖS96c, LÖS96d, LÖ98]:

- a new representation of point-set topological relations;

- introduction of a unified representation of spatial objects;

- a complete set of definitions of both topological and directional relations;

- a rich set of spatial inference rules;

- comprehensive support for all user spatial queries elaborated above.

- a two-level spatial processing model.

The integration of the proposed spatial model into an object-oriented model is postponed to Chapter 3 because some temporal concepts must be introduced.

2.1 Related Work

Egenhofer and Franzosa [EF91] have specified eight fundamental topological relations that can hold between two planar regions. These relations are computed using four intersections over the concepts of *boundary* and *interior* of point-sets between two regions embedded in a two-dimensional space. For example, let A° and B° be the interiors of objects A and B, respectively, and $A^\bullet$ and $B^\bullet$ be the boundaries of A and B, respectively. Then the combinations of intersections ($A^\circ \cap B^\circ$, $A^\circ \cap B^\bullet$, $A^\bullet \cap B^\circ$, $A^\bullet \cap B^\bullet$) between interiors and boundaries form a set of topological relations. Checking whether each intersection is *empty* or *not empty* produces 16 possible results. However, only eight meaningful topological relations are identified: *disjoint, contains, inside, touch, equal, covers, covered_by*, and *overlap* as shown in Figure 2.1. A spatial SQL [Ege94] based on this topological representation has been proposed. The spatial SQL supports direct spatial search, hybrid spatial search, complex spatial search, and direct spatial computation.

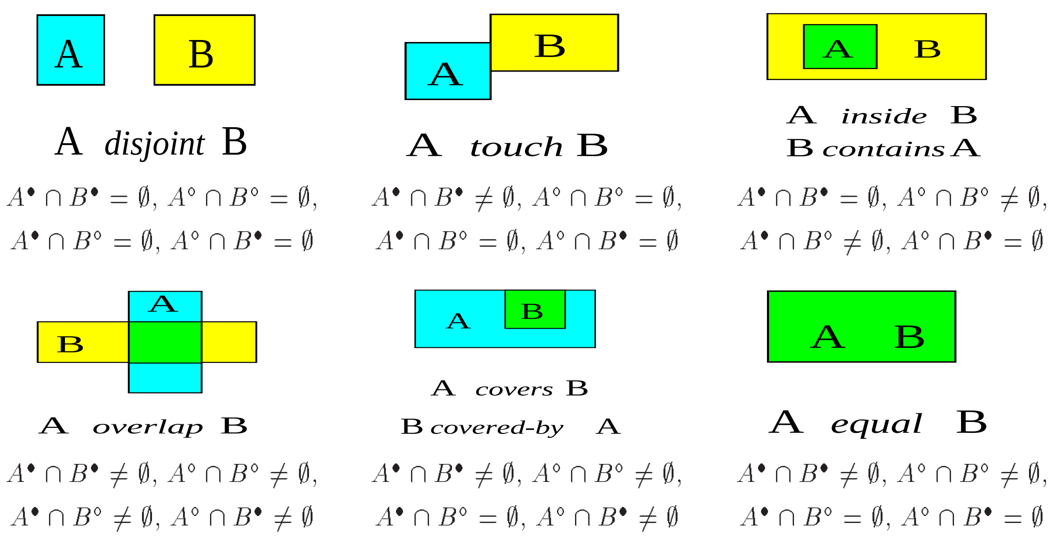


Figure 2.1: Definitions of Topological Relations

Papadias et al. [PS94, PTSE95] assume a construction process that detects a set of special points in an image, called *representative points*. Every spatial relation in the modeling space can be defined using only these points. Two kinds of representative points are considered: *directional* and *topological*. For example, some possible directional representative points are the centroid of an object or the lower-left and upper-right corners of an object's MBR. So in the case of using two representative points the directional relations between objects can be defined as intervals which may facilitate the retrieval of spatial objects from a database using an R-tree based indexing mechanism [PTSE95].

Nabil et al. [NSN95] propose a two dimensional projection interval relationship (2D-PIR) to represent spatial relationships based on Allen's interval algebra and Egenhofer's 4-intersection formalism. It enables a graph representation for pictures based on 2D-PIR to be constructed. In order to overcome some problems of using the MBR with boundaries not parallel to the horizontal and vertical axes, in the 2D-PIR representation two alternative solutions are proposed: slope projection and the introduction of topological relations. However, neither of these two solutions is complete in the sense that there still exist cases that the 2D-PIR representation cannot handle. For example, two spatial objects, which do not overlap, might overlap in the 2D-PIR representation.

An object-oriented image model, VIMSYS, is proposed in [GWJ91]. VIMSYS provides four layers to view image data. They are image representations and relations (IR), domain objects and relations (DO), image object and relations (IO), and domain events and relations (DE). Its prototype allows the user to incrementally express incomplete queries starting from semantic image objects with similarity predicates and it can retrieve a whole image or just a part of an image.

The Video Semantic Directed Graph (VSDG) model is a graph-based conceptual video model [DDI+95]. This model also suggests using Allen's temporal interval algebra to model spatial relations among objects. However, their definitions of such spatial relations are both incomplete and unsound. They are incomplete because only five topological relations are defined while there should be at least six topological relations [EF91]. They are unsound because the definition of *touch* relation includes *disjoint* relation while all the topological relations should be exclusive. Abdelmoty et al. [AEG94] extend the 4-intersection formalism [EF91] for topological relations to represent *orientational* relations. The orientational relations require a reference object called an *origin* to establish a spatial relation. Each object's MBR, together with four lines extending from the corners of the rectangle to the origin, are used to divide the space external to the object into four semi-infinite areas. The directional relations between two objects are defined using the intersections of these areas. Hernández [Her94] defines the composition of topological and directional relations, with the result being pairs of topological/directional relations. Composition is accomplished using *relative topological orientation nodes* as a store for intermediate results. This work is extended in [CSE94] to handle composition of distance and directional relations.

2.2 Spatial Properties of Salient Objects

2.2.1 Spatial Representations

It is a common strategy in spatial access methods to store object approximations and use these approximations to index the data space in order to efficiently retrieve the potential objects that satisfy the result of a query [PTSE95]. Depending on application domains, there are several options in choosing object approximations. Minimum bounding rectangle (MBR) has been used extensively to approximate objects because they need only two points for their representation and many efficient index techniques have been developed (such as R-tree [Gut84] and its variants [SRF87, BKSS90]). They are the most commonly used approximations in spatial applications. Hence, MBRs are chosen to approximate objects in the model proposed in this thesis.

Definition 1 The MBR of a salient object A is defined by (A_x, A_y) where $A_x = [x_s, x_f]$, $A_y = [y_s, y_f]$. x_s and x_f are A's projection on the X axis with $x_s \leq x_f$ and similarly for y_s and y_f.

Definition 2 The *spatial property* of a salient object A is defined by a triple (A_x, A_y, C_A) where C_A is the centroid of A and A_x and A_y are its MBR projections as defined above. The centroid is represented by a two dimensional point (x_A, y_B).

This can be naturally extended by considering the time dimension: $(A_x(t), A_y(t), C_A(t))$ which captures the spatial property of a salient object A at time t. Basically, the spatial property of an object is described by its MBR and a representative point. In video modeling the time dimension must be also considered, as the spatial property of an object may change over time. For example, suppose the spatial property of A is $(A_x(t_1), A_y(t_1), C_A(t_1))$ at time t_1 and it becomes $(A_x(t_2), A_y(t_2), C_A(t_2))$ at time t_2. The displacement of A over the time interval $I = [t_1, t_2]$ is $\mathit{DISP}(A, I) \equiv \sqrt{(x_A(t_1) - x_A(t_2))^2 + (y_A(t_1) - y_A(t_2))^2}$ which is the movement of the centroid of A. Also the Euclidean distance between two objects A and B at time t_k is $\mathit{DIST}(A, B, t_k) \equiv \sqrt{(x_A(t_k) - x_B(t_k))^2 + (y_A(t_k) - y_B(t_k))^2}$ which is also characterized by the centroid of A and B. The goal is to design a spatial representation that is sufficiently powerful to support both quantitative and qualitative spatial retrieval.

2.2.2 Spatial Relationships

Spatial relationships between objects are very important in MDBMSs because they implicitly support *fuzziness* (i.e., certain level of variance from user queries) by their qualitative property. Allen [All83] gives a temporal interval algebra (Table 2.1) for representing and reasoning about temporal relations between events represented as intervals. A *temporal interval* is identified as the basic anchored specification of time. These temporal relations have been cited by others [SF95, NSN95] for their simplicity and ease of implementation with constraint propagation algorithms. The elements of the algebra are sets of seven basic relations that can hold between two intervals and their inverse relations.

The temporal interval algebra essentially consists of the topological relations in one dimensional space, enhanced by the distinction of the order of the space. The order is used to capture the directional aspects in addition to the topological relations. 12 directional relations are considered in this model and they are classified into the following three categories: *strict directional relations*

Relation	Symbol	Inverse	Meaning
A before B	`b`	`bi`	AAA BBB
A meets B	`m`	`mi`	AAABBB
A overlaps B	`o`	`oi`	AAA BBB
A during B	`d`	`di`	AAA BBBBB
A starts B	`s`	`si`	AAA BBBBB
A finishes B	`f`	`fi`	AAA BBBBB
A equal B	`e`	`e`	AAA BBB

Table 2.1: Allen's 13 Temporal Interval Relations

(north, south, west, and east), *mixed directional relations* (northeast, southeast, northwest, and southwest), and *positional relations* (above, below, left, and right). The definitions of these relations in terms of Allen's temporal algebra are given in Table 2.2. The symbols $\wedge$ and $\vee$ are the standard logical *AND* and *OR* operators, respectively. A short notation $\{\}$ is used to distribute the $\vee$ operator over interval relations. For example $A_x\,\{\mathtt{b},\mathtt{m},\mathtt{o}\}\,B_x$ is equivalent to $A_x\,\mathtt{b}\,B_x \vee A_x\,\mathtt{m}\,B_x \vee A_x\,\mathtt{o}\,B_x$.

Among Egenhofer's eight topological relations there are two inverse relations: *covers* vs *covered_by* and *inside* vs *contains*. Hence, only six topological relations are defined here, as shown in the last part of Table 2.2. Note that the definitions of directional and topological relations are based on two dimensional (2D) space. In 3D space, the depth of an object has to be considered and the extension is straightforward.

Figure 2.2 shows all the cases of A *northwest* of B ($A\,\mathtt{NW}\,B$) while Figure 2.3 shows all the cases of A *north* of B ($A\,\mathtt{NT}\,B$). Since $A\,\mathtt{NW}\,B \equiv (A_x\,\{\mathtt{b},\mathtt{m}\}\,B_x \wedge A_y\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,A_{jy}) \vee (A_x\,\{\mathtt{o}\}\,B_x \wedge A_y\,\{\mathtt{bi},\mathtt{mi}\}\,B_y)$, the following three cases are possible:

- If A_x is before B_x ($A_x\,\{\mathtt{b}\}\,B_x$), A_y can be after, met by, or overlapped by B_y ($A_y\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,B_y$). These cases correspond to (a), (b), and (c) of Figure 2.2, respectively.

- If A_x meets B_x ($A_x\,\{\mathtt{m}\}\,B_x$), A_y can be after, met by, or overlapped by B_y ($A_y\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,B_y$). These cases correspond to (d), (e), and (f) of Figure 2.2, respectively.

- If A_x overlaps with B_x ($A_x\,\{\mathtt{o}\}\,B_x$), A_y can only be either after or met by B_y ($A_y\,\{\mathtt{bi},\mathtt{mi}\}\,B_y$). These cases correspond to (g), and (h) of Figure 2.2, respectively.

The definition of A *above* B ($A\,\mathtt{AB}\,B \equiv A_y\,\{\mathtt{bi},\mathtt{mi}\}\,B_y$) requires that A's projection on the y-axis be greater than or equal to B's projection on the y-axis. The *above* relation includes A *north*

Relation	Meaning	Definition
$A\ \texttt{ST}\ B$	South	$A_x\ \{\texttt{d},\texttt{di},\texttt{s},\texttt{si},\texttt{f},\texttt{fi},\texttt{e}\}\ B_x \wedge A_y\ \{\texttt{b},\texttt{m}\}\ B_y$
$A\ \texttt{NT}\ B$	North	$A_x\ \{\texttt{d},\texttt{di},\texttt{s},\texttt{si},\texttt{f},\texttt{fi},\texttt{e}\}\ B_x \wedge A_y\ \{\texttt{bi},\texttt{mi}\}\ B_y$
$A\ \texttt{WT}\ B$	West	$A_x\ \{\texttt{b},\texttt{m}\}\ B_x \wedge A_y\ \{\texttt{d},\texttt{di},\texttt{s},\texttt{si},\texttt{f},\texttt{fi},\texttt{e}\}\ B_y$
$A\ \texttt{ET}\ B$	East	$A_x\ \{\texttt{bi},\texttt{mi}\}\ B_x \wedge A_y\ \{\texttt{d},\texttt{di},\texttt{s},\texttt{si},\texttt{f},\texttt{fi},\texttt{e}\}\ B_y$
$A\ \texttt{NW}\ B$	Northwest	$(A_x\ \{\texttt{b},\texttt{m}\}\ B_x \wedge A_y\ \{\texttt{bi},\texttt{mi},\texttt{oi}\}\ B_y)\vee$ $(A_x\ \{\texttt{o}\}\ B_x \wedge A_y\ \{\texttt{bi},\texttt{mi}\}\ B_y)$
$A\ \texttt{NE}\ B$	Northeast	$(A_x\ \{\texttt{bi},\texttt{mi}\}\ B_x \wedge A_y\ \{\texttt{bi},\texttt{mi},\texttt{oi}\}\ B_y)\vee$ $(A_x\ \{\texttt{oi}\}\ B_x \wedge A_y\ \{\texttt{bi},\texttt{mi}\}\ B_y)$
$A\ \texttt{SW}\ B$	Southwest	$(A_x\ \{\texttt{b},\texttt{m}\}\ B_x \wedge A_y\ \{\texttt{b},\texttt{m},\texttt{o}\}\ B_y)\vee$ $(A_x\ \{\texttt{o}\}\ B_x \wedge A_y\ \{\texttt{b},\texttt{m}\}\ B_y)$
$A\ \texttt{SE}\ B$	Southeast	$(A_x\ \{\texttt{b},\texttt{m}\}\ B_x \wedge A_y\ \{\texttt{b},\texttt{m},\texttt{o}\}\ B_y)\vee$ $(A_x\ \{\texttt{oi}\}\ B_x \wedge A_y\ \{\texttt{b},\texttt{m}\}\ B_y)$
$A\ \texttt{LT}\ B$	Left	$A_x\ \{\texttt{b},\texttt{m}\}\ B_x$
$A\ \texttt{RT}\ B$	Right	$A_x\ \{\texttt{bi},\texttt{mi}\}\ B_x$
$A\ \texttt{BL}\ B$	Below	$A_y\ \{\texttt{b},\texttt{m}\}\ B_y$
$A\ \texttt{AB}\ B$	Above	$A_y\ \{\texttt{bi},\texttt{mi}\}\ B_y$
$A\ \texttt{EQ}\ B$	Equal	$A_x\ \{\texttt{e}\}\ B_x \wedge A_y\ \{\texttt{e}\}\ B_y$
$A\ \texttt{IN}\ B$	Inside	$A_x\ \{\texttt{d}\}\ B_x \wedge A_y\ \{\texttt{d}\}\ B_y$
$A\ \texttt{CV}\ B$	Cover	$(A_x\ \{\texttt{di}\}\ B_x \wedge A_y\ \{\texttt{fi},\texttt{si},\texttt{e}\}\ B_y)\vee$ $(A_x\ \{\texttt{e}\}\ B_x \wedge A_y\ \{\texttt{di},\texttt{fi},\texttt{si}\}\ B_y)\vee$ $(A_x\ \{\texttt{fi},\texttt{si}\}\ B_x \wedge A_y\ \{\texttt{di},\texttt{fi},\texttt{si},\texttt{e}\}\ B_y)$
$A\ \texttt{OL}\ B$	Overlap	$A_x\ \{\texttt{s},\texttt{d},\texttt{f}\}\ B_x \wedge A_y\ \{\texttt{o},\texttt{oi},\texttt{di},\texttt{fi},\texttt{si}\}\ B_y\vee$ $A_x\ \{\texttt{e},\texttt{di},\texttt{fi},\texttt{si}\}\ B_x \wedge A_y\ \{\texttt{o},\texttt{oi},\texttt{s},\texttt{d},\texttt{f}\}\ B_y\vee$ $A_x\ \{\texttt{o},\texttt{oi}\}\ B_x \wedge A_y\ \{\texttt{o},\texttt{oi},\texttt{f},\texttt{fi},\texttt{d},\texttt{di},\texttt{s},\texttt{si},\texttt{e}\}\ B_y$
$A\ \texttt{TC}\ B$	Touch	$(A_x\ \{\texttt{m},\texttt{mi}\}\ B_x\wedge$ $A_y\ \{\texttt{d},\texttt{di},\texttt{s},\texttt{si},\texttt{f},\texttt{fi},\texttt{o},\texttt{oi},\texttt{m},\texttt{mi},\texttt{e}\}\ B_y)\vee$ $(A_x\ \{\texttt{d},\texttt{di},\texttt{s},\texttt{si},\texttt{f},\texttt{fi},\texttt{o},\texttt{oi},\texttt{m},\texttt{mi},\texttt{e}\}\ B_x\wedge$ $A_y\ \{\texttt{m},\texttt{mi}\}\ B_y)$
$A\ \texttt{DJ}\ B$	Disjoint	$A_x\ \{\texttt{b},\texttt{bi}\}\ B_x \vee A_y\ \{\texttt{b},\texttt{bi}\}\ B_y$

Table 2.2: Directional and Topological Relation Definitions

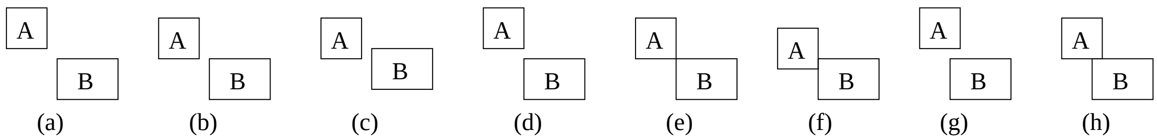


Figure 2.2: All the Cases of $A\,\mathtt{NW}\,B$

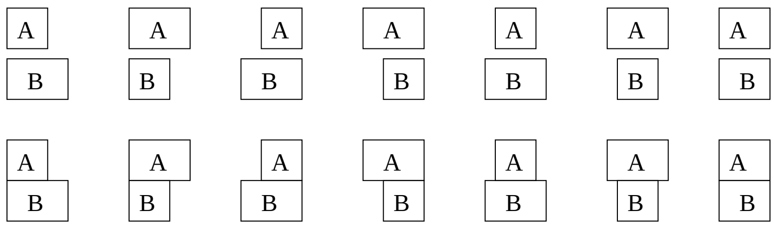


Figure 2.3: All the Cases of $A\,\mathtt{NT}\,B$

of B ($A\,\mathtt{NT}\,B$), part of A *northwest* of B ($A\,\mathtt{NW}\,B$), and part of A *northeast* of B ($A\,\mathtt{NE}\,B$) according to the definitions. The positional relations are more general than those defined in [SYH94] because only the top half in Figure 2.3 (A and B do not touch) satisfy the relation *above*. The definition of A *overlap* B ($A\,\mathtt{OL}\,B$) indicates that object A shares some region with object B. If this shared region becomes either just a line or a point, then A touches B ($A\,\mathtt{TC}\,B$). A is *disjoint* B ($A\,\mathtt{DJ}\,B$) indicates no shared region between A and B. All the definitions of spatial relations can be found in Appendix A.

In this definition, if two objects overlap, they do not have any directional relation. This is certainly an arguable definition. In Figure 2.4 it is natural to say A *overlaps* B in (a) and A *west* of B in (c). However, it may not be reasonable to claim the latter in cases (b) and (d). The problem comes from the representation of the temporal interval algebra which does not distinguish the degree of the overlap regions. Even worse, in Figure 2.4(e) A and B do not have a clear directional relation. This may not be satisfactory in some fine-grain multimedia applications. Nevertheless, using the interval relations (algebra) to capture both directional and topological relations of spatial objects can offer more information about spatial relations than other methods [NSN95], such as 2D-string [CJT96]. In other words, it has greater expressive power. Adopting such an interval algebra is even more attractive in MDBMSs than in GIS and image systems for those DBMSs which support Allen's temporal algebra in their temporal models. In these DBMSs no special treatment is required for spatial intervals from an implementation point of view.

Example 1 Consider a locomotive image as in Figure 2.5. Suppose the salient objects are the cab (cab), the window (window), the left big wheel (lbw), the right big wheel (rbw), the left small wheel (lsw), the right small wheel (rsw), the smokestack (smokestack), and the body (body). From the image, the following spatial relations can be detected: { window `IN` cab, cab `DJ` lbw, cab `LT` body, cab `AB` lbw, body `TC` rbw, body `NW` rsw, body `NT` rbw, smokestack `NE` body, lbw `WT` rbw, rbw `WT` lsw}.

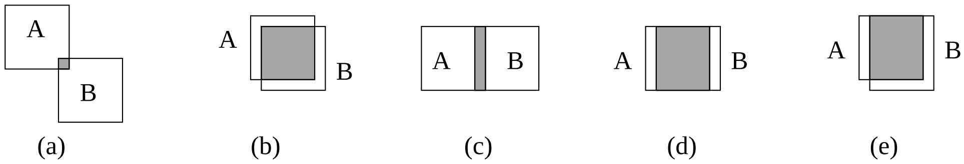


Figure 2.4: Some Non-directional Spatial Cases

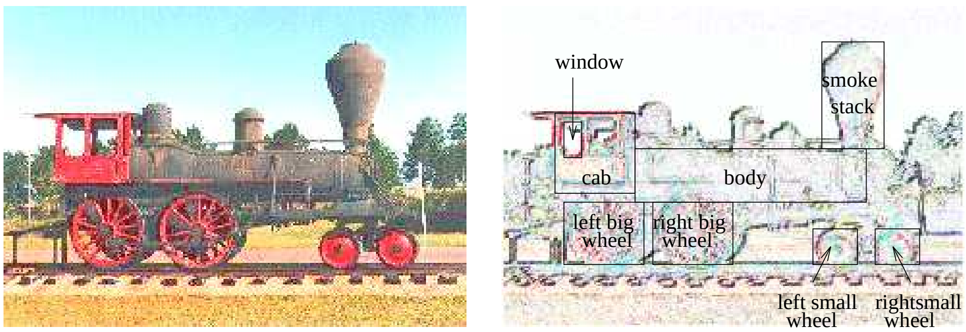


Figure 2.5: A Locomotive Image and Salient Objects

2.3 Reasoning about Spatial Relations

Logic-based representations, such as rules, are used in qualitative spatial reasoning since they provide a natural and flexible way to represent spatial knowledge [PS94]. Such representations usually have well defined semantics and simple inference rules that can be integrated into any deductive system. For example, if relations A *north* of B, and B *overlap* C, and C *north* of D are valid, then one can infer that A *above* D holds. This can be expressed as a rule

$$A \,\mathtt{NT}\, B \wedge B \,\mathtt{OL}\, C \wedge C \,\mathtt{NT}\, D \Rightarrow A \,\mathtt{AB}\, D.$$

A spatial inference rule can support spatial analysis without transforming any spatial knowledge into the domain of underlying coordinates and point-region representations. Reasoning with imprecise and incomplete information may be achieved in a purely qualitative matter or, when necessary and available, augmented by quantitative information. Another major advantage of using spatial inference rules is to save space within MDBMSs because it is not reasonable to explicitly store all the spatial relations.

Rule 1 (Reflexivity) The following topological relation is reflexive:

$A \,\mathtt{EQ}\, A$.

Rule 2 (Symmetry) The following topological relations are symmetric:

$A \,\mathtt{EQ}\, B \Leftrightarrow B \,\mathtt{EQ}\, A$ $\quad$ $A \,\mathtt{OL}\, B \Leftrightarrow B \,\mathtt{OL}\, A$ $\quad$ $A \,\mathtt{TC}\, B \Leftrightarrow B \,\mathtt{TC}\, A$

$A \,\mathtt{DJ}\, B \Leftrightarrow B \,\mathtt{DJ}\, A$.

Rule 3 (Inverse Property) The following directional relations are inverses of each other:

$A\,\mathtt{NT}\,B \Leftrightarrow B\,\mathtt{ST}\,A \qquad A\,\mathtt{NE}\,B \Leftrightarrow B\,\mathtt{SW}\,A \qquad A\,\mathtt{NW}\,B \Leftrightarrow B\,\mathtt{SE}\,A$

$A\,\mathtt{ET}\,B \Leftrightarrow B\,\mathtt{WT}\,A \qquad A\,\mathtt{LT}\,B \Leftrightarrow B\,\mathtt{RT}\,A \qquad A\,\mathtt{AB}\,B \Leftrightarrow B\,\mathtt{BL}\,A.$

Rule 4 (Transitivity) Let $\theta \in \{\mathtt{NW}, \mathtt{NE}, \mathtt{SW}, \mathtt{SE}, \mathtt{LT}, \mathtt{RT}, \mathtt{AB}, \mathtt{BL}, \mathtt{IN}, \mathtt{EQ}\}$ then

$A\theta B \wedge B\theta C \Rightarrow A\theta C.$

Rule 5 (Implication) Some relations imply other relations:

$A\,\mathtt{NT}\,B \Rightarrow A\,\mathtt{AB}\,B \qquad A\,\mathtt{ST}\,B \Rightarrow A\,\mathtt{BL}\,B$

$A\,\mathtt{WT}\,B \Rightarrow A\,\mathtt{LT}\,B \qquad A\,\mathtt{ET}\,B \Rightarrow A\,\mathtt{RT}\,B.$

Rule 6 The relationships between $\{\mathtt{IN}, \mathtt{CV}\}$ and $\{\mathtt{DJ}\}$ are:

$A\,\mathtt{IN}\,B \wedge B\,\mathtt{DJ}\,C \Rightarrow A\,\mathtt{DJ}\,C \qquad A\,\mathtt{CV}\,B \wedge A\,\mathtt{DJ}\,C \Rightarrow B\,\mathtt{DJ}\,C.$

Rule 7 This rule indicates relationships between topological relation $\{\mathtt{IN}, \mathtt{CV}\}$ and the positional directional relations $\{\mathtt{LT}, \mathtt{RT}, \mathtt{AB}, \mathtt{BL}\}$. Suppose $\theta \in \{\mathtt{LT}, \mathtt{RT}, \mathtt{AB}, \mathtt{BL}\}$ then

$A\,\mathtt{IN}\,B \wedge B\theta C \Rightarrow A\theta C \qquad A\,\mathtt{CV}\,B \wedge A\theta C \Rightarrow B\theta C.$

Rule 8 This rule indicates relationships between topological relations $\{\mathtt{IN}, \mathtt{CV}\}$ and the strict directional relations $\{\mathtt{ST}, \mathtt{WT}, \mathtt{NT}, \mathtt{ET}\}$:

$A\,\mathtt{IN}\,B \wedge B\,\mathtt{NT}\,C \Rightarrow A\,\mathtt{AB}\,C \qquad A\,\mathtt{IN}\,B \wedge B\,\mathtt{ST}\,C \Rightarrow A\,\mathtt{BL}\,C$

$A\,\mathtt{IN}\,B \wedge B\,\mathtt{WT}\,C \Rightarrow A\,\mathtt{LT}\,C \qquad A\,\mathtt{IN}\,B \wedge B\,\mathtt{ET}\,C \Rightarrow A\,\mathtt{RT}\,C$

$A\,\mathtt{CV}\,B \wedge A\,\mathtt{NT}\,C \Rightarrow B\,\mathtt{AB}\,C \qquad A\,\mathtt{CV}\,B \wedge A\,\mathtt{ST}\,C \Rightarrow B\,\mathtt{BL}\,C$

$A\,\mathtt{CV}\,B \wedge A\,\mathtt{WT}\,C \Rightarrow B\,\mathtt{LT}\,C \qquad A\,\mathtt{CV}\,B \wedge A\,\mathtt{ET}\,C \Rightarrow B\,\mathtt{RT}\,C.$

Rule 9 Suppose $\theta \in \{\mathtt{LT}, \mathtt{RT}, \mathtt{AB}, \mathtt{BL}\}$. The relationships between $\{\mathtt{OL}\}$ and $\{\mathtt{LT}, \mathtt{RT}, \mathtt{AB}, \mathtt{BL}\}$ are

$A\theta B \wedge B\,\mathtt{OL}\,C \wedge C\theta D \Rightarrow A\theta D.$

Rule 10 The relationships between $\{\mathtt{OL}\}$ and $\{\mathtt{ST}, \mathtt{WT}, \mathtt{NT}, \mathtt{ET}\}$ are

$A\,\mathtt{NT}\,B \wedge B\,\mathtt{OL}\,C \wedge C\,\mathtt{NT}\,D \Rightarrow A\,\mathtt{AB}\,D \qquad A\,\mathtt{WT}\,B \wedge B\,\mathtt{OL}\,C \wedge C\,\mathtt{WT}\,D \Rightarrow A\,\mathtt{LT}\,C$

$A\,\mathtt{ST}\,B \wedge B\,\mathtt{OL}\,C \wedge C\,\mathtt{ST}\,D \Rightarrow A\,\mathtt{BL}\,C \qquad A\,\mathtt{ET}\,B \wedge B\,\mathtt{OL}\,C \wedge C\,\mathtt{ET}\,D \Rightarrow A\,\mathtt{RT}\,C.$

Rule 11 The relationships between $\{\mathtt{ST}, \mathtt{WT}, \mathtt{NT}, \mathtt{ET}\}$ and $\{\mathtt{NW}, \mathtt{NE}, \mathtt{SW}, \mathtt{SE}\}$ are

$A\,\mathtt{NT}\,B \wedge B\,\mathtt{NW}\,C \Rightarrow A\,\mathtt{AB}\,C \qquad A\,\mathtt{NT}\,B \wedge B\,\mathtt{NE}\,C \Rightarrow A\,\mathtt{AB}\,C$

$A\,\mathtt{ST}\,B \wedge B\,\mathtt{SW}\,C \Rightarrow A\,\mathtt{BL}\,C \qquad A\,\mathtt{ST}\,B \wedge B\,\mathtt{SE}\,C \Rightarrow A\,\mathtt{BL}\,C$

$A\,\mathtt{WT}\,B \wedge B\,\mathtt{NW}\,C \Rightarrow A\,\mathtt{LT}\,C \qquad A\,\mathtt{WT}\,B \wedge B\,\mathtt{SW}\,C \Rightarrow A\,\mathtt{LT}\,C$

$A\,\mathtt{ET}\,B \wedge B\,\mathtt{NE}\,C \Rightarrow A\,\mathtt{RT}\,C \qquad A\,\mathtt{ET}\,B \wedge B\,\mathtt{SE}\,C \Rightarrow A\,\mathtt{RT}\,C.$

Rule 12 The relationships between $\{\mathtt{NW}, \mathtt{NE}, \mathtt{SW}, \mathtt{SE}\}$ and $\{\mathtt{ST}, \mathtt{WT}, \mathtt{NT}, \mathtt{ET}\}$ are

$A\,\mathtt{NW}\,B \wedge B\,\mathtt{NT}\,C \Rightarrow A\,\mathtt{AB}\,C \qquad A\,\mathtt{NE}\,B \wedge B\,\mathtt{NT}\,C \Rightarrow A\,\mathtt{AB}\,C$

$A\,\mathtt{SW}\,B \wedge B\,\mathtt{ST}\,C \Rightarrow A\,\mathtt{BL}\,C \qquad A\,\mathtt{SE}\,B \wedge B\,\mathtt{ST}\,C \Rightarrow A\,\mathtt{BL}\,C$

$A\,\mathtt{NW}\,B \wedge B\,\mathtt{WT}\,C \Rightarrow A\,\mathtt{LT}\,C \qquad A\,\mathtt{SW}\,B \wedge B\,\mathtt{WT}\,C \Rightarrow A\,\mathtt{LT}\,C$

$A\,\mathtt{NE}\,B \wedge B\,\mathtt{ET}\,C \Rightarrow A\,\mathtt{RT}\,C \qquad A\,\mathtt{SE}\,B \wedge B\,\mathtt{ET}\,C \Rightarrow A\,\mathtt{RT}\,C.$

Rule 13 The relationships between $\{\mathtt{LT}, \mathtt{RT}, \mathtt{AB}, \mathtt{BL}\}$ and $\{\mathtt{ST}, \mathtt{WT}, \mathtt{NT}, \mathtt{ET}, \mathtt{NW}, \mathtt{NE}, \mathtt{SW}, \mathtt{SE}\}$ are

$A\,\mathtt{LT}\,B \wedge B\,\{\mathtt{WT}, \mathtt{NW}, \mathtt{SW}\}\,C \Rightarrow A\,\mathtt{LT}\,C \qquad A\,\mathtt{RT}\,B \wedge B\,\{\mathtt{NE}, \mathtt{ET}, \mathtt{SE}\}\,C \Rightarrow A\,\mathtt{RT}\,C$

$A\,\mathtt{AB}\,B \wedge B\,\{\mathtt{NE}, \mathtt{NT}, \mathtt{NW}\}\,C \Rightarrow A\,\mathtt{AB}\,C \qquad A\,\mathtt{BL}\,B \wedge B\,\{\mathtt{SE}, \mathtt{ST}, \mathtt{SW}\}\,C \Rightarrow A\,\mathtt{BL}\,C.$

Rule 14 Suppose $\theta \in \{\mathtt{LT}, \mathtt{RT}, \mathtt{AB}, \mathtt{BL}\}$. The relationships between $\mathtt{TC}$ and $\{\mathtt{LT}, \mathtt{RT}, \mathtt{AB}, \mathtt{BL}\}$ are

$A \theta B \wedge B\,\mathtt{TC}\,C \wedge C \theta D \Rightarrow A \theta D$.

Rule 15 The relationships between $\mathtt{TC}$ and $\{\mathtt{ST}, \mathtt{WT}, \mathtt{NT}, \mathtt{ET}\}$ are

$A\,\mathtt{NT}\,B \wedge B\,\mathtt{TC}\,C \wedge C\,\mathtt{NT}\,D \Rightarrow A\,\mathtt{AB}\,D \qquad A\,\mathtt{ST}\,B \wedge B\,\mathtt{TC}\,C \wedge C\,\mathtt{ST}\,D \Rightarrow A\,\mathtt{BL}\,D$

$A\,\mathtt{WT}\,B \wedge B\,\mathtt{TC}\,C \wedge C\,\mathtt{WT}\,D \Rightarrow A\,\mathtt{LT}\,D \qquad A\,\mathtt{ET}\,B \wedge B\,\mathtt{TC}\,C \wedge C\,\mathtt{ET}\,D \Rightarrow A\,\mathtt{RT}\,D$.

A query subsystem is usually provided to support efficient image retrievals based on user queries. In reality, user queries usually contain spatial constraints or relations which must be satisfied when the results are returned to users. How can the spatial inference rules be used to retrieve data from an MDBMS? Since all the rules are propositional Horn clauses, they can be easily integrated into any MDBMS by either using a logic language with a simple inference engine (like DATALOG [Ull85] or LDL [TZ86]) or by using a lookup table. These rules serve as semantic transformation rules that can be employed during query optimization. The correctness proof of all the rules can be found in Appendix G.

Example 2 Look at the Example 2.1 and Figure 2.5. Besides those detected relations, the following new spatial relations can be derived:

- Since the cab is to the left of the body and the body is to the northwest of the right small wheel (cab LT body $\wedge$ body NW rsw), it is derivable that the cab is to the left of the right wheel (cab LT rsw) by **Rule 13**.
- Since the window is inside the cab and the cab is disjoint from the left big wheel (window IN cab $\wedge$ cab DJ lbw), it is derivable that the window is disjoint from the left big wheel (window DJ lbw) by **Rule 6**.
- Given the window is inside the cab and the cab is above the left big wheel (window IN cab $\wedge$ cab AB lbw), it is derivable that the window is above the left big wheel (window AB lbw) by **Rule 7**.
- Since the smokestack is to the northeast of the body and the body is to the north of the left big wheel (smokestack NE body $\wedge$ body NT rbw), it is derivable that the smokestack is above the right big wheel (smokestack AB rbw) by **Rule 12**.
- Given the left big wheel is to the west of the right big wheel and the right big wheel is to the west of the left small wheel (lbw WT rbw $\wedge$ rbw WT lsw), it is derivable that the left big wheel is to the left of the left small wheel (lbw LT lsw) by **Rule 5**.
- Since the cab is to the left of the body and the body is externally connected to the right big wheel, and the right big wheel is to the left of the left small wheel (cab LT body $\wedge$ body TC rbw $\wedge$ rbw LT lsw, rbw LT lsw is derived from rbw WT lsw from **Rule 5**), it is derivable that the left big

wheel is to the west of the left small wheel (lbw WT lsw) by **Rule 4**. Furthermore, the cab is to the left of the left small wheel (cab LT lsw) from **Rule 14**.

The above derived relations are not complete, i.e., there are many other relations which are derivable from the given relations.

It is assumed that some basic spatial relations are generated a priori either by image processing algorithms or manually, or by a hybrid mechanism. These relations are usually stored as metadata or object attributes and will be used by the query subsystem for efficient query processing. Together with content-based indexing, the availability of metadata or object attributes avoids the invocation of the expensive image processing algorithms each time a query is processed. Metadata or object attributes may not be *definite* in the sense that it is possible to have more than one relation between two objects. Furthermore they may not be *complete* or may not be necessary to be complete in the sense that not all the spatial relations are explicitly captured by the metadata or object attributes. There are two major reasons that cause such incompleteness[SYH94]. First, it could be impossible for existing image processing algorithms to recognize all the objects and their relations. Second, those implied relations may not be stored explicitly in order to save space. Saving space is particularly attractive [YGBC89] in a distributed environment where the metadata or object attributes are stored at the user sites with limited storage facilities, while the actual images may be stored in remote image archives. The query subsystem executed at the user sites uses the metadata or object attributes to determine the images that need to be retrieved from the remote sites.

2.4 Topological Spatial Relations

The proposed MBR approach works fine for many applications. However, it may encounter difficulties in handling topological relations if precise spatial relations are required. Consider an image example as shown in Figure 2.6. Suppose the salient objects are {treeTop, trunk, balloon, littleGirl, and bigGirl}. The MBRs of balloon and bigGirl indicate that they overlap although they do not. Suppose a user posts a query "*find all the objects overlapping* treeTop". Then objects trunk and balloon will be returned if the computation is based on MBRs. However, it is obvious that balloon should not be included. In this case balloon is called a *false hit*. To deal with false hit problem, the precise spatial information of salient objects is important. The point-set approach is considered to be the most general model for the representation of spatial information [EF91, SYH94].

Egenhofer and Franzosa's proposal for eight fundamental topological relations for spatial regions has been widely adopted as the topological model in spatial databases [Par95, PSV96]. Their work has shown that the description of topological spatial relations in terms of topologically invariant properties of point-sets is fairly simple. As a consequence, the topological spatial relation between two point-sets may be determined with little computational effort. However, point-set approaches

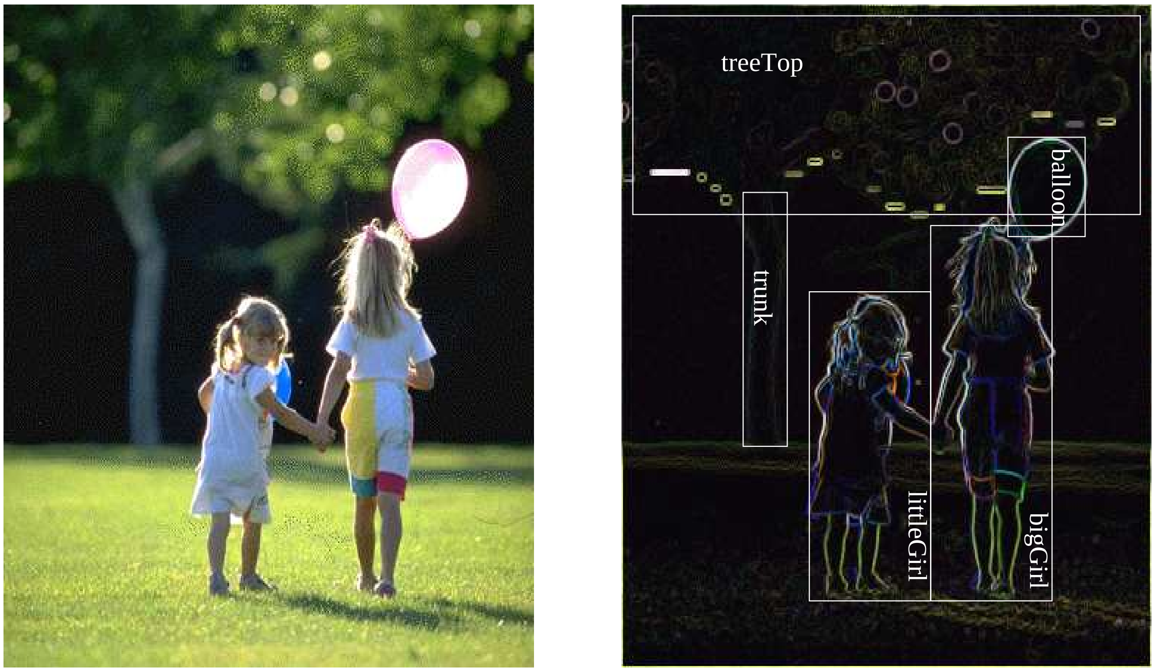


Figure 2.6: Minimum Bounding Box

are notorious for large storage space requirements. Saving space with a small degradation in computation efficiency is the major motivation of this research.

A new way of computing point-set topological spatial relations is proposed by eliminating explicit storage of the *interior* point-set within a database. Experiments show that not only does this dramatically reduce the database size, but it also speeds up query evaluation in some cases.

2.4.1 Formal Approach

The model of topological spatial relations is based on the point-set topological notions of *interior* and *boundary*. In this subsection the appropriate definitions and results from point-set topology are presented. Most of the results can be found in many algebraic topology text books, e.g. [Sch68].

Definition 3 Let X be a set. Suppose that associated with X there is a system Ψ of subsets, called the *open sets*, that has the following properties:

- The union of an arbitrary collection of sets from Ψ belongs to Ψ. The empty subset of X belongs to Ψ; and
- the intersection of a finite number of sets from Ψ belongs to Ψ. X belongs to Ψ.

Then it is said that Ψ defines a *topology* on X or that X is provided with a topology. The sets of Ψ are called the *open sets* of this topology. A set X that is provided with a topology is called a *topology space*. Let x be a point of the topology space X. Every open set of X that contains x is called an *open neighborhood* of x. A subset of X is called a *neighborhood* of x if it contains an open neighborhood of x.

Let A be a subset of the topological space X. The set of all those points for which A is a neighborhood is denoted by A°. A° is called the *interior* of A and the points of A° are called the *interior points* of A. From the definition, it follows that A° is open and is the union of all those open sets of X which are contained in A. Hence, A° is the largest open set of X contained in A. The complements of the open sets in X are called *closed sets* and have two properties: (1) the intersection of an arbitrary collection of closed sets is closed, i.e., X is closed; (2) the union of a finite number of closed sets is closed, i.e., the empty subset is closed.

Definition 4 The *closure* of $A \subseteq X$, denoted by $\overline{A}$, is defined to be the intersection of all closed sets that contain A, i.e., the closure of A is the smallest closed set containing A. It follows that a is in the closure of A iff every neighborhood of a intersects A. Hence, $a \in \overline{A}$ iff $A \cap B \neq \emptyset$ for every open set B containing a. The empty set is the only set with empty closure. The *boundary* of $A \subseteq X$, denoted by $A^\bullet$, is the intersection of the closure of A and the closure of the complement of A, i.e., $A^\bullet = \overline{A} \cap \overline{X - A}$. The points in the boundary of A are called *boundary points*. They form a closed set.

Until now two most important concepts, interior set and boundary set, are introduced. They are crucial in defining topological relations later on. Note that a boundary point of A may not belong to A. From the definition, a is in the boundary of A iff every neighborhood of a intersects both A and its complement, i.e., $a \in \overline{A}$ iff $A \cap B \neq \emptyset$ and $B \cap (X - A) \neq \emptyset$ for every open set B containing a. The relationship between interior, closure, and boundary can be summarized as $A^\circ \cap A^\bullet = \emptyset$ and $A^\circ \cup A^\bullet = \overline{A}$. The following example clarifies these concepts.

Example 3 Suppose $X = \{a, b, c, d\}$ and $A = \{a, b\}$. It is easy to verify that $\Psi = \{\emptyset, \{a, b\}, \{c, d\}, \{a, b, c, d\}\}$ is a topology on X. Therefore, $A^\circ = \{a, b\}$, $A^\bullet = \emptyset$, and $\overline{A} = \{a, b\}$. If there is another topology $\Psi\prime = \{\emptyset, \{a, c\}, \{b, d\}, \{a, b, c, d\}\}$, then $A^\circ = \emptyset$, $A^\bullet = \{a, b, c, d\}$, and $\overline{A} = \{a, b, c, d\}$.

The model described in [EF91] defines the topological spatial relations between two subsets, A and B, of a topological space X based on a consideration of these four intersections of the boundaries and interiors of the two sets A and B, i.e., $A^\bullet \cap B^\bullet$, $A^\circ \cap B^\circ$, $A^\bullet \cap B^\circ$, and $A^\circ \cap B^\bullet$. By assigning the appropriate value of *empty* and *non-empty* to the four intersections, there are 16 different possibilities. A set is either empty or non-empty; therefore, these 16 topological spatial relations provide complete coverage, that is, given pair of sets A and B in X, there is always a topological relation associated with A and B. Furthermore, a set cannot simultaneously be empty and non-empty, from which follows that the 16 topological relations are mutually exclusive. The framework for the spatial relations between point-sets carries over to spatial regions (see Appendix B for the formal definition of spatial regions), however, not all of the 16 relations between arbitrary point-sets exist between two spatial regions.

Since the model is designed for polygonal areas in the plane, the topological space X and the sets under consideration in X are restricted, i.e., the topological space X is connected. This guarantees that the boundary of each set of interest is not empty. The sets of interest are *spatial regions* defined as $A \subset X$, a non-empty proper subset of X, such that A° is connected and $A = \overline{A^{\circ}}$. Such a restriction makes some combinations of the four intersections impossible. Egenhofer and Franzosa have proven that there exist only nine possible topological relations and this number is decreased to eight if the boundaries are assumed to be connected as shown in Figure 2.1.

A proposal to replace interior point-sets of spatial regions by functions, to save space, is presented next. It is known that the method of representing topological relations by calculating the four intersection sets of the interior and the boundary of two regions is both simple and effective. A general topological spatial relation system can be easily implemented based on this method. Although the point-set approach is simple and general, it has a huge storage requirement. Therefore, the goal here is to reduce the storage requirement. Before giving the main theorem, more notation is necessary.

Definition 5 A point p in 2D space has two orthogonal values: x from domain D_1 and y from domain D_2, which can be represented by $p = (x, y)$. p's values on D_1 and D_2 are denoted by p_x and p_y respectively. Let A be a point-set; two functions are defined:

$\mathcal{F}_x$: $A \longrightarrow D_1$ maps a point into domain D_1, e.g., $\mathcal{F}_x(p) = p_x$;
$\mathcal{F}_y$: $A \longrightarrow D_2$ maps a point into domain D_2, e.g., $\mathcal{F}_y(p) = p_y$.

A spatial region is *convex* if for any two points in the interior there exists a line joining the two points such that all the points on the line are also in the interior.

Theorem 1 Let A be a convex region in X. In a two dimensional space, for all $p \in A^{\circ}$ there always exist four points p_1, p_2, p_3, p_4 in $A^{\bullet}$, i.e., $p_i \in A^{\bullet}$ $(i = 1, 2, 3, 4)$, such that
$\mathcal{F}_x(p_1) = \mathcal{F}_x(p)$ and $\mathcal{F}_y(p_1) < \mathcal{F}_y(p)$,
$\mathcal{F}_x(p_2) = \mathcal{F}_x(p)$ and $\mathcal{F}_y(p_2) > \mathcal{F}_y(p)$,
$\mathcal{F}_y(p_3) = \mathcal{F}_y(p)$ and $\mathcal{F}_x(p_3) < \mathcal{F}_x(p)$, and
$\mathcal{F}_y(p_4) = \mathcal{F}_y(p)$ and $\mathcal{F}_x(p_4) > \mathcal{F}_x(p)$.

The proof of this theorem is deferred to Appendix B as more topology definitions are necessary. From Theorem 2.1 it is possible to determine whether or not $p \in A^{\circ}$ is true. The next step is to compute the four point-set intersections, i.e.:

- $A^{\circ} \cap B^{\bullet} \neq \emptyset$ if $\exists p \in B^{\bullet}$ such that $p \in A^{\circ}$;

- $A^\circ \cap B^\bullet = \emptyset$ if $\forall p \in B^\bullet$ such that $p \in A^\circ$;
- $B^\circ \cap A^\bullet \neq \emptyset$ if $\exists p \in A^\bullet$ such that $p \in B^\circ$;
- $B^\circ \cap A^\bullet = \emptyset$ if $\forall p \in A^\bullet$ such that $p \in B^\circ$.

This is important because all the topological relations are determined by the results of this four intersections (see Figure 2.1). The cases of computing $A^\bullet \cap B^\circ \neq \emptyset$ and $A^\bullet \cap B^\circ = \emptyset$ are similar to the above while the case of $A^\bullet \cap B^\bullet$ is trivial. The most difficult part is in computing $A^\circ \cap B^\circ$ since it is impossible to fetch an element from A° or B° although its membership can be tested. Fortunately, according to [CSE94] there are only two topological relations which need to compute $A^\circ \cap B^\circ$. The two relations are *disjoint* and *touch* and both need to test $A^\circ \cap B^\circ = \emptyset$. However, the relation *disjoint* can be expressed by following three conditions: (1) $A^\bullet \cap B^\bullet = \emptyset$; (2) $A^\bullet \cap B^\circ = \emptyset$; and (3) $A^\circ \cap B^\bullet = \emptyset$; while the relation *touch* can be expressed by following three conditions: (1) $A^\bullet \cap B^\bullet \neq \emptyset$; (2) $A^\bullet \cap B^\circ = \emptyset$; and (3) $A^\circ \cap B^\bullet = \emptyset$. The proof of their equivalence is trivial.

2.4.2 Experiment

In last subsection, a new approach, replacing region interior point-sets by functions, is proposed. To evaluate the performance of this approach in computing topological relations, a series of experiments are conducted on a commercial ODBMS, ObjectStore [LLOW91], as a spatial object repository. Currently, no efficiency values for point-set based topological computation are available in the literature, making these experiments unique. Consequently, the new approach is only compared with the original approach of Egenhofer and Franzosa. The experiment is divided into two phases. In the first phase, major parameters or factors are examined to see which may play more important role than others. This allows further attention to focus on those important factors. In the second phase, the performance of the original approach, where the interior of a spatial region is explicitly stored, is compared with the proposed functional approach of replacing such an interior with a function. The performance metrics in the second phase are CPU time required in computing topological relations.

Instead of writing specialized code to have faster access, The ObjectStore query facility is used to access spatial data. Although this makes the program much slower, it is more realistic and general. The total amount of space saved is application dependent. For example, if the average size of region interiors is the same as the average size of region boundaries, then 50% space savings may be achieved. In most cases, the size of region interiors is bigger than the size of region boundaries so more than 50% space savings can be expected. For the rest of this section, performance comparisons are in terms of CPU time.

2.4.2.1 Experimental Design

The experiment design technique being used in the first phase is the 2^k factorial design [Jai91]. 2^k factorial design is an effective experimental method for evaluating effects of some factors and their combinations. k represents the number of factors and 2 refers to the levels of each factor. Three factors are chosen in the experiment; therefore, 2^3 factorial design is used. The three factors are:

1. Average sizes of interiors of spatial regions;

2. Average sizes of boundaries of spatial regions;

3. Point-sets with or without indexes.

Since the argument is to eliminate spatial region's interior explicitly, it is natural to see how different sizes between interiors and boundaries of spatial regions influence the overall performance. It is a common technique to use indexes in databases to handle large amount of data. Therefore, how indexes affect the overall performance is also tested. An index for a point-set sorts the points of the set in ascending order. One thing to note is that in the case of using indexes the time to create and maintain the indexes is not considered. This is because few or no updates are assumed in an MDBMS. However, the cost of maintaining indexes can be very high if there are many region updates. The two levels for average sizes of region interiors and region boundaries are both 100 points and 200 points respectively. In the case of 100 interior points, 800 bytes space is saved for a region if each point is represented by two integers and each integer is represented by four bytes. Similarly, 1600 bytes are saved for a region in the case of 200 points. A two-dimensional hash index over a point is used. The maximum testing size is 200 spatial regions which are randomly generated. The final database size is at around 8M bytes and all the experiments are conducted on a SUN SPARC SS20 with one 70MHZ CPU and 64MB main memory.

2.4.2.2 Performance Analysis

Table 2.3 and Table 2.4 show the results of the experiments over the three factors: interior sizes (100 points Interior-100 and 200 points Interior-200), boundary sizes (100 points Boundary-100 and 200 points Boundary-200), and indexable (Hash or no Hash). For each spatial region, a pair-wise comparison over all other spatial regions is computed for each of the eight topological relations. The average number of topological relation comparisons are 135,000. In general, the original approach has a better performance than the functional approach except when both interior and boundary sizes are 100 and there is no hash index. Although the difference of this exception is very little, it does reveal that when region sizes are small the functional approach is also competitive.

However, the purpose of this 2^3 factorial experiment, i.e., phase one, is not to compare the performance between these two approaches; it is to show the impact of each factor and their combinations over each approach. So more attention can be paid to those factors in the subsequent

Index	Interior-100		Interior-200	
	Boundary-100	Boundary-200	Boundary-100	Boundary-200
No Hash	489	257	205	194
Hash	201	163	107	108

Table 2.3: CPU Times (seconds) of the Original Approach

Index	Interior-100		Interior-200	
	Boundary-100	Boundary-200	Boundary-100	Boundary-200
No Hash	471	288	242	218
Hash	266	240	134	194

Table 2.4: CPU Times (seconds) of the Functional Approach

performance experiments. Using the method, discussed in [Jai91], of computing the effect of each factor, Table 2.5 is derived from Table 2.3 and Table 2.4. The process of this transformation is described in Appendix C. In Table 2.5 *Interior* is the effect of interior sizes of spatial regions; *Boundary* is the effect of boundary sizes of spatial regions; and *Hash* is the effect of hash index. "I-B" is the effect of the combination of *Interior* and *Boundary*; "I-H" is the effect of the combination of *Interior* and *Hash*; "B-H" is the effect of the combination of *Boundary* and *Hash*; and "I-B-H" is the effect of the combination of *Interior*, *Boundary*, *Hash*.

In the original approach, indexing plays an important role with 39% contribution to the total CPU time while the sizes of region boundaries contribute 30%, which is also significant. This reveals that region boundaries are more important than region interiors because it seems they consume more CPU time in the computation of topological relations. The combination of *Interior* and *Hash* contributes 8% and other combinations have little impact on the overall performance.

Approach	*Interior*	*Boundary*	*Hash*	I-B	I-H	B-H	I-B-H
Original	10%	30%	39%	8%	5%	5%	3%
Functional	5%	42%	28%	11%	11%	2%	1%

Table 2.5: Percentage of Effects of 3 Factors over Different Approaches

In the functional approach, indexing plays a less important role (28% of total CPU time) compared with the original approach (39%). This makes sense because in the functional approach there are no interiors for spatial regions so indexes are only for the boundaries. On the other hand, boundaries of spatial regions (42%) have bigger impact compared with the case in the original approach (30%). The reason is that region boundaries are used more frequently than before. The interiors contribute only 5% of the performance. The combination of region interiors and boundaries have 11% effect which is quite significant. So does the combination of region interiors and hash index. The contribution of the combination of all the three factors is negligible.

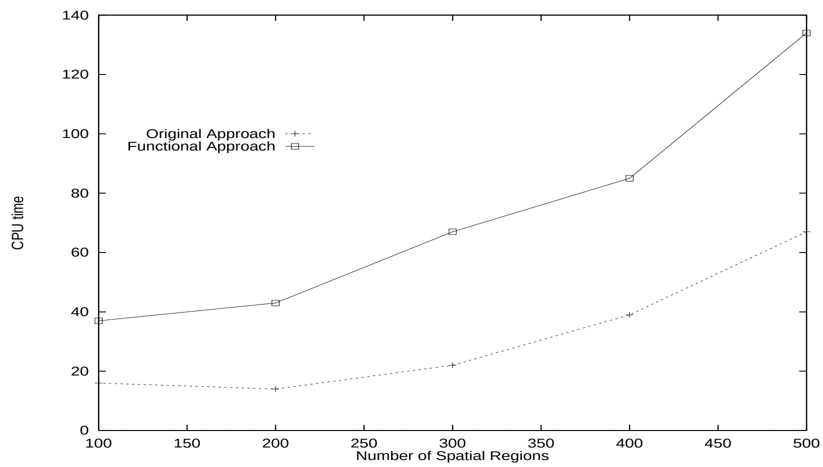


Figure 2.7: *overlap* (CPU time in seconds)

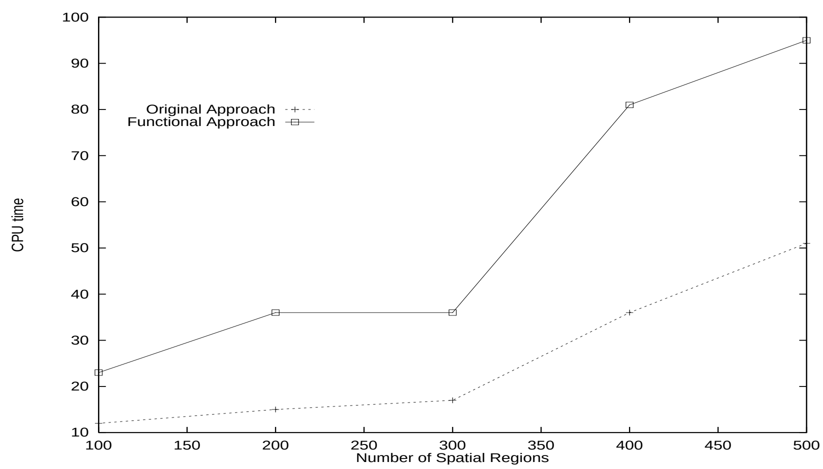


Figure 2.8: *cover* (CPU time in seconds)

Figure 2.7 to Figure 2.12 show different CPU times in computing different topological relations. The horizontal scales are the number of spatial regions and the vertical scales are the CPU time in terms of seconds. For each spatial region, a pair-wise computation of a spatial relation over all other spatial regions, is conducted. In Figure 2.7 (*overlap*) and Figure 2.8 (*cover*), the original approach beats the functional approach with quite a bit of margin while in Figure 2.10 (*disjoint*) the two approaches are alternately taking the lead. In between, Figure 2.9 (*touch*) shows the original approach is slightly better than the new approach. This reveals that when there are

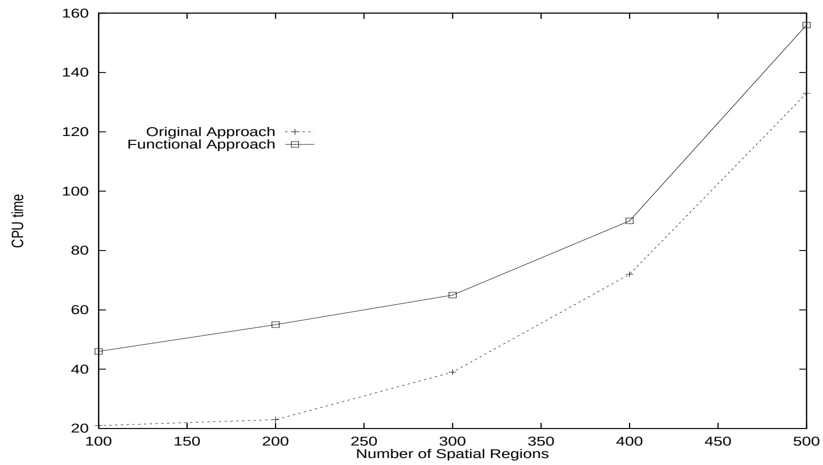


Figure 2.9: *touch* (CPU time in seconds)

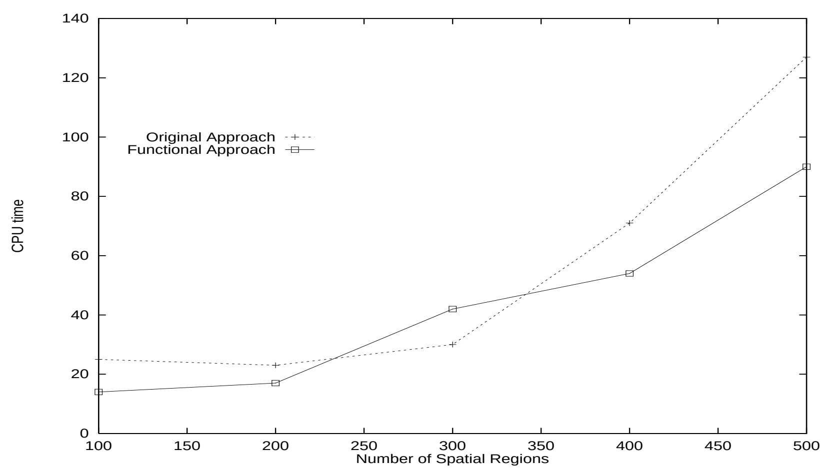


Figure 2.10: *disjoint* (CPU time in seconds)

many comparisons to region interiors (as the cases in computing *overlap* and *cover* relations) the original approach performs better. This is because region interiors have been indexed in the original approach while there is no index at all in the functional approach.

In Figure 2.11 (*equal*) and Figure 2.12 (*inside*), the functional approach performs consistently better than the original approach. The reason is that the computation is mainly dependent on region boundaries so the computation workload is roughly the same with both two approaches. For the functional approach the database is much smaller so it requires less CPU and I/O time to do the processing. That is why the functional approach out-performs the original approach in these two cases. In summary the functional approach has better performance than the original approach in a few cases. These graphs also reveal that the most expensive topological relations in terms of CPU time are *touch*, *overlap*, and *disjoint*.

In order to compare the overall performance of the two approaches, a series of experiments are

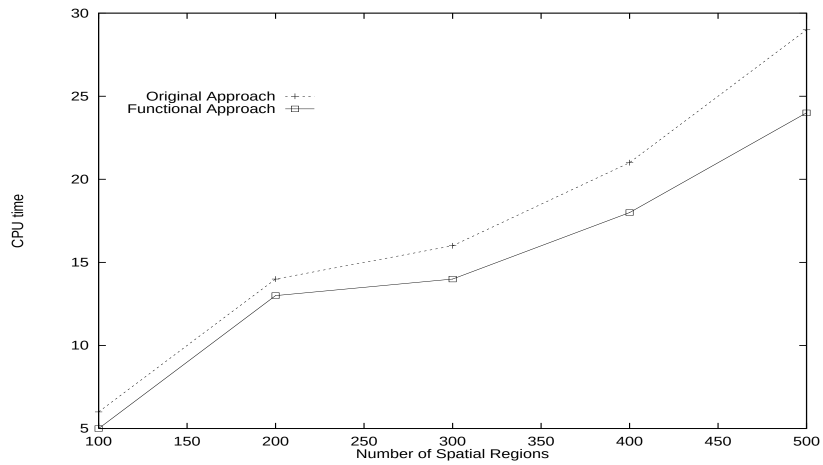


Figure 2.11: *equal* (CPU time in seconds)

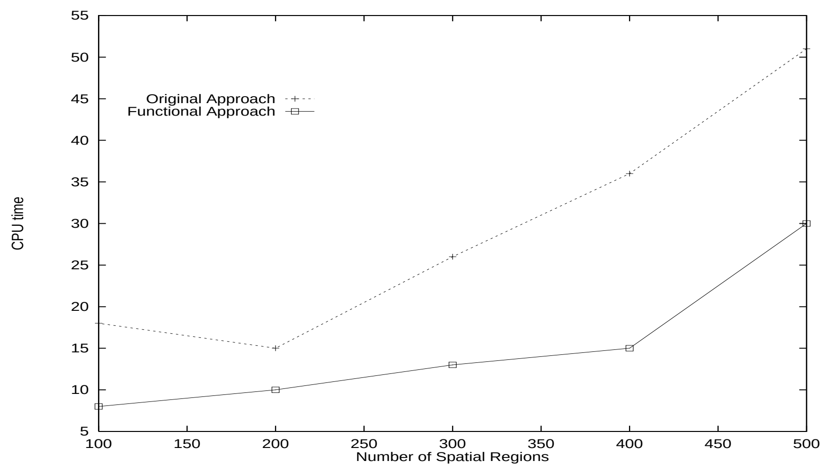


Figure 2.12: *inside* (CPU time in seconds)

conducted over a spatial database with 800 spatial regions. The average sizes of region interiors and boundaries are both 200 points so the space savings is about 50%. Considering the volume of the data, hash indexes are used in all the runs, which is certainly favorable to the original approach. The total size of the database is around 15MB. Again, each region is compared with all other regions with all eight topological relations. The experiments are conducted by testing different number of spatial regions and the results are given in Figure 2.13. Although the original approach performs consistently better than the functional approach, the relative gap is decreasing as the number of comparing regions increase. For example, when the number of regions is less than 500 the original approach is 100% faster than the functional approach. However, in the case of 800 regions, this number is reduced to 25%. This trend seems to continue from the experiments. No further experiment for over 800 regions is conducted because 800 regions in an image is an extraordinary number for all the known applications.

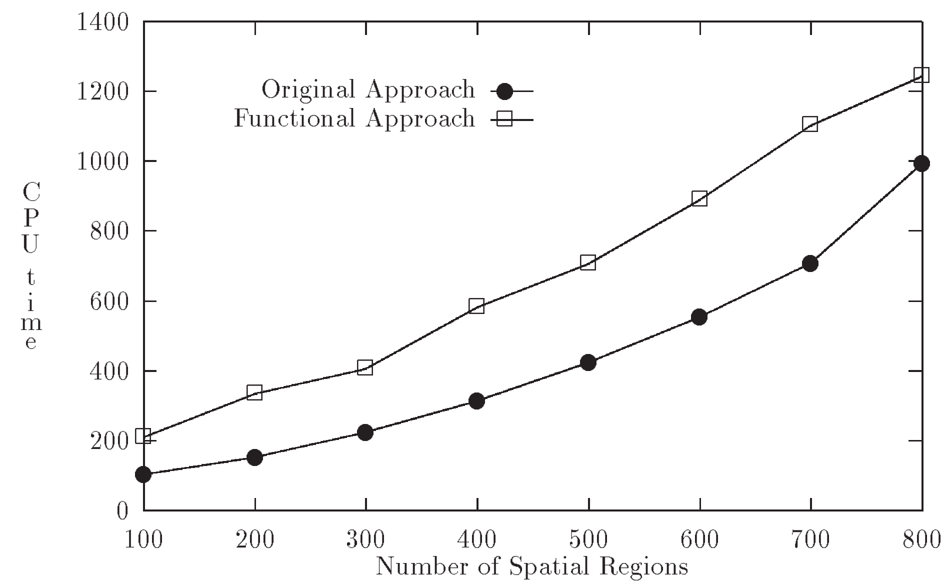


Figure 2.13: CPU Time Comparison Between Original and Functional Approaches

2.5 Spatial Similarity

Retrieval by *spatial similarity* deals with a class of queries that is based on spatial relationships among the domain objects. Spatial queries require selecting target images that satisfy, to varying degrees, the spatial relationships specified in the query. This degree of conformance is used to rank order target images with respect to the query. The robustness of an algorithm lies in that it recognizes translation, scale, rotation, and arbitrary variants of images.

Informally, two images are *spatially identical* if they contain the same number of salient objects and the spatial relationships among the objects are the same between the two images. An algorithm for spatial similarity can be defined as a function that takes two images (or the corresponding physical salient objects and their spatial relations) as input and returns a value in the range of $[0, 1]$ as output. Value 1 is expected when the input images are spatially identical[1]. The other values must be proportional to the degree of agreement in the spatial relationships between the corresponding physical salient objects in the input images. In other words, spatial similarity algorithms should possess *monotonicity* property. Let t_1 and t_2 be two target images and q be the query image. If t_1 satisfies more spatial relationships in q than t_2, t_1 should be assigned a higher similarity value.

Intuitively, each physical salient object has a centroid which describes its mass location. An edge can be drawn from any two centroids in a 2D space. By comparing the angle of two edges (one is from query image and the other one is from target image), the spatial similarity of the two

[1]This is a very popular definition in the image similarity research community. However, the common deinition in other areas is the opposite. I.e., if the two input images are identical the value should be 0 because 0 means no difference.

Spatial_Similarity(q_image, t_image)
$q\_image = \{o_{11}, o_{12}, \ldots, o_{1m}\}$;
$t\_image = \{o_{21}, o_{22}, \ldots, o_{2n}\}$;
{
(1) **sort** salient objects in q_image and t_image such that
(2) same objects are paired in front of the object list and
(3) suppose there are k common pairs:
(4) $q\_image = \{o_{q_1}, o_{q_2}, \ldots, o_{q_m}\}$;
(5) $t\_image = \{o_{t_1}, o_{t_2}, \ldots, o_{t_n}\}$; and
(6) $o_{q_i} = o_{t_i}\ (i = 1, 2, \ldots, k)$;
(7) **if** $(k < 2)$
(8) **return** 0.0;
(9) **if** $(k == 2)$
(10) **return** $\theta = \ll \overrightarrow{o_{q_1} o_{t_1}}, \overrightarrow{o_{q_2} o_{t_2}} \gg$;
(11) **if** q_image and t_image are perfect variant
(12) **return** 1.0;
(13) $SIM = 0.0$;
(14) **for** $(i = 1;\ i < k;\ i++)$ {
(15) Compute θ_i between two lines:
(16) one made by two objects from $\{o_{q_1}, o_{q_2}, \ldots, o_{q_k}\}$
another made by two objects from $\{o_{t_1}, o_{t_2}, \ldots, o_{t_k}\}$;
(17) $SIM_i = \frac{1}{k}(\frac{1+cos\theta}{2})$;
(18) $SIM = SIM + SIM_i$;
(19) **return** SIM;
(20) } /* end of **for** */
}

Figure 2.14: Spatial Similarity Algorithm

images can be determined. Formally, let $p(x_p, y_p)$ and $q(x_q, y_q)$ be two points in the 2D Cartesian coordinate space. The notation $\overrightarrow{pq}$ denotes the vector from point p to point q. The magnitude of $\overrightarrow{pq}$, denoted by $||\overrightarrow{pq}||$, is given by $||\overrightarrow{pq}|| = \sqrt{(x_q - x_p)^2 + (y_q - y_p)^2}$. Given two vectors $\overrightarrow{pq}$ and $\overrightarrow{p\prime q\prime}$, the acute or obtuse angle, θ, between $\overrightarrow{pq}$ and $\overrightarrow{p\prime q\prime}$ is computed by the expression

$$\cos\theta = \frac{\overrightarrow{pq} \bullet \overrightarrow{p\prime q\prime}}{||\overrightarrow{pq}||\ ||\overrightarrow{p\prime q\prime}||}, \quad 0 \le \theta \le \pi,$$

where the operator $\bullet$ denotes vector dot product. The angle is denoted by $\ll \overrightarrow{pq}, \overrightarrow{p\prime q\prime} \gg$. This approach has been proven to be both efficient and effective [GJ96].

The spatial similarity algorithm, shown in Figure 2.14, is revised from [GJ96]. The algorithm takes two input images q_image (query image) and t_image (target image) and returns a value between 0 and 1. Statements (1) — (6) compute the salient objects which are common to both q_image and t_image. For example, suppose $q\_image = \{person, sea, sand\}$ and

$t\_image = \{yatch, sea, sand, person\}$. Then, the result of the sorting process is: $q\_image = \{person, sea, sand\}$, $t\_image = \{person, sea, sand, yatch\}$, and $k = 3$. If the number of common salient objects is less than 2 (0 or 1), their spatial similarity is set to be 0 (statements (7) and (8)). If this number is 2, only one angle determines the similarity value (statements (9) and (10)). Statements (11) and (12) determine if images q_image and t_image are perfect variants. Two images are *perfect variants* if they spatially identical. Test for perfect variants is based on the equality of the corresponding acute or obtuse angles in q_iamge and t_image. For perfect variant images, a value of 1 is returned. Otherwise, the spatial similarity is computed based on the number of objects common to q_image and t_image, and the difference in the orientations of the corresponding vectors formed by joining the centroids of successive salient objects (statements (13) to (20)).

Chapter 3

The Temporal Model

Some media, such as video and audio, possess temporal properties. This research does not consider audio; therefore, video is the focus of discussion throughout this chapter. A tree-based model for specifying the temporal semantics of video data is proposed to achieve efficient object accessing. Furthermore, a unique way of integrating this video model into an object DBMS (TIGUKAT) is presented. The result of such an integration allows users to explore the video data using object-oriented techniques through a uniform accessing interface. A broad range of temporal queries are supported by this integrated video ODBMS.

Video content analysis is an active research area [GBT94, IKO$^+$96, YYL96]. A major goal of most works is to achieve automatic extraction of feature semantics from a video and to support content-based video retrieval. Feature extraction includes object recognition, spatio-temporal encoding, scene analyzing, etc. However, feature extraction is outside the scope of this thesis, although a few prototype systems [SZ94, LPE96, SK96, YYL96, HJ97] have been reported that can be used in conjunction with the work of this thesis.

3.1 Video Modeling

Video modeling is the process of translating raw video data into an efficient internal representation to capture video semantics. A video model is an essential part of an abstract MDBMS which can be used as a basis for declarative querying. The procedural process of extracting video semantics from a video is called *video segmentation*. There are two approaches to video segmentation in an object-oriented context: *stream-based* and *structured* [Gha96]. In the *stream-based* approach, as shown in Figure 3.1 a video consists of a sequence of clips and a clip in turn consists of a sequence of *frames* that are displayed at a specified rate, e.g., 30 frames a second for TV shows, 24 frames a second for most movies. In the *structured approach*, a clip is considered as a sequence of scenes and each scene consists of multiple objects and background objects, actors (e.g., three dimensional representation of Mickey Mouse, dinosaurs, lions), light sources that define shading, and the audience's view point. Spatial constructs are used to place objects that constitute a scene in a rendering space while temporal constructs describe how the objects and their relationship evolve as a function of time. The rendering of a structured presentation must satisfy the temporal constraints imposed on the display of each object.

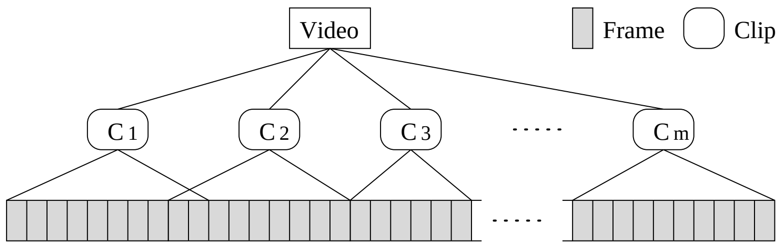


Figure 3.1: Stream-based Video Clips and Frames

Each approach has its advantages and disadvantages [Gha96]. The stream-based approach benefits from decades of research and development on analog devices that generate high resolution frames. This is because the output of these devices is digitized to generate a stream-based video clip. However, it is difficult to extract the contents from stream-based clips. Moreover, if an object is displayed at a playing rate lower than its prespecified frame rate, its display will suffer from frequent disruptions and delays unless elaborated service adaptation policies are employed. On the other hand, very little work has been done on the structured approach because there are no devices equivalent to a camcorder that can analyze a scene to compute either its individual objects or the temporal and spatial relationships that exist among these object. Therefore, the stream-based approach is technically feasible and is the focus of this thesis.

The information about the semantics of a video must be structured so that indexes can be built for efficient data retrieval from a video ODBMS. The functionality of such a database depends on its model of time. A new video model, called Common Video Object Tree (CVOT), is proposed. The CVOT model has the capability of supporting both spatial and temporal relationships among video objects. It also allows native support for a rich set of temporal multimedia operations. The abstract CVOT model is seamlessly integrated with the powerful temporal object model of TIGUKAT to provide concrete ODBMS support for video data. Some unique features of the proposed model are the following:

- A flexible way of organizing video clips based on a common object tree;
- A uniform approach of modeling video using temporal histories;
- A rich set of qualitative temporal operations on video data in an integrated video ODBMS.

3.2 Related Work

There is significant interest in modeling video systems. Gibbs et. al. [GBT94] investigate timed streams as the basic abstraction for modeling temporal media using object-oriented technology. The *media element* in their model corresponds to video frames in the proposed model. A *timed stream* is modeled by a finite sequence of tuples $< e_i, s_i, d_i >$, $i = 1, \ldots, n$, where e_i is a media element, s_i is the start time of e_i and d_i is its duration. Three general structuring mechanisms (interpretation, derivation, and composition) are used to model time-based media. Videos are modeled as time-based streams. However, the temporal operations are not well supported in their model. For example, there is no support for qualitative temporal relationships.

AVIS (Advanced Video Information System) [ACC+96], which uses *association maps* to group salient objects and activities, is quite close to the CVOT model. A video stream is segmented into a set of frame-sequences $[x, y)$, where x is the start frame and y is the end frame. Based on the association maps, a *frame segment tree* is built to capture objects and activities occurring in the frame-sequences. Then two arrays are created: the objectArray and the activityArray. Each element of either array is an ordered linked list of pointers to nodes in the frame segment tree. It is shown that such a data structure results in efficient query retrieval. The CVOT model adopted part of this efficient indexing scheme. Although the AVIS model is similar to CVOT, there are some fundamental differences:

- In AVIS two neighboring clips must be consecutive with no overlaps. In CVOT, the segmentation of a video can be arbitrary in the sense that two neighboring clips could overlap. This extension in the CVOT model is important because special video editing techniques, such as fade-in and fade-out, can be modeled.

- The frame segment tree in AVIS is a binary tree and, in practice, it is made up of many *empty nodes* (nodes without any common objects or activities from its child nodes). This problem could result in deep binary trees. In CVOT, the tree is an arbitrary tree and only the root node's common object set is allowed to be empty. The major advantage of such a shallow tree is its small number of nodes which can result in significant space saving and possibly efficient searching. The tradeoff could be the higher cost of building an arbitrary tree and more complex searching algorithms.

- Activities are not explicitly modeled in CVOT because an activity may span multiple clips. Therefore, one part of an activity may be in one clip and the other part may be in another clip. In AVIS, all clips have to be equal length while CVOT permits variable lengths. This is important if clips are determined according to scenes or activities since the duration of these scenes or activities may be different.

Video Semantic Directed Graph (VSDG) is a graph-based conceptual video model [DDI+95].

The most important feature of the VSDG model is an unbiased representation of the information while providing a reference framework for constructing a semantically heterogeneous user's view of the video data. Using this model along with an object-oriented hierarchy, a new video system architecture is proposed which can automatically handle video data. The video semantic directed graphs are more complicated than CVOT trees without introducing any more capability. Furthermore, the VSDG model does not directly support range queries such as "*Find all the clips in which John appears.*"

OVID (Object-oriented Video Information Database) [OT93] is an object-oriented video model. It introduces the notion of a *video object* which can identify an arbitrary video frame sequence (a meaningful scene) as an independent object and describe its contents in a dynamic and incremental way. However, the OVID model has no schema and the traditional class hierarchy of ODBMSs is not assumed. An inheritance based on an *interval inclusion relationship* is introduced to share descriptional data among video objects. A major problem with the OVID model is its heavy dependence on the *video description* which has to be done manually. The idea of modeling salient objects and activities using the object-oriented technology in CVOT is borrowed from the OVID model. Later on, multimedia temporal operations are integrated into this object-oriented model.

Little and Ghafoor [LG93] have described a temporal model to capture the timing relationships between objects in composite multimedia objects, and mapped it to a relational database. This model forms a basis for a hierarchical data model and for temporal access control algorithms to allow VCR-like capabilities. An architecture, called ViMod, for a video object database based on video features is proposed in [JH94]. The design of this model is the result of studying the metadata characteristics of queries and video features. The *algebraic video* data model [WDG94] allows users to model nested video structures such as shots, scenes and sequences and to define the output characteristics of video segments. A quite comprehensive set of temporal operators for video editing has been defined within the algebraic video system. Object-oriented Multimedia Database Environment for General Application (OMEGA) [Mas91] is an object-oriented database system for managing multimedia data. In this model an *acceptor* is defined to allow a user to communicate with the system. Such an acceptor can recognize and describe a real world entity by using any symbol system that is allowed in specified media. Some extensions are made in OMEGA for representing object hierarchies and temporal and spatial information about multimedia data. A video-on-demand (VOD) system [LAF+93] has been developed to model a real-world video rental store. The VOD system allows the viewing preferences of each individual to be tailored and adapted to the available programs (for example, one viewer's interest in classic movies, another's restriction to children's programs). It supports some temporal access operations (such as fast-forward, reverse playback, loop, middle-point suspension, middle-point resumption etc.) by using a domain-specific model for motion pictures. However, The VOD explores no spatial and little temporal information of video contents.

3.3 The Common Video Object Tree Model

There are several different ways to segment a video into clips, e.g., by *fixed time intervals* or by *shots*. A *fixed time interval* segmentation approach divides a video into equal length clips using a predefined time interval (e.g. 2 seconds) while a *shot* is a set of continuous frames captured by a single camera action [HJW95]. A key aspect in video modeling is how to organize video clips to efficiently handle user queries. An interval-based model [ACC+96, DDI+95] seems a promising approach because of its simplicity and efficiency. It is even more attractive here because the proposed spatial model for images is also interval-based. Therefore, both the spatial and temporal models can be based on a common idea. In the CVOT model there is no restriction on how videos are segmented. Without loss of generality, it is assumed that any given video stream has a finite number of clips and any clip has a finite number of frames as shown in Figure 3.1. The main idea behind the CVOT model is to find all the common objects among clips and to group clips. A tree structure is used to represent such a clip group. In this section a formal definition of the model and an algorithm for constructing the tree are given.

3.3.1 Video Clip Sets

A clip is associated with a time interval $[t_s, t_f]$. More specifically, a clip is a set of consecutive frames between a start time t_s and a finish time t_f: $[t_s, t_f] = \{t | t_s \leq t \leq t_f\}$ where t_s and t_f are the relative (discrete) time instants in a given video and $t_s \leq t_f$. Since all clips have a start and finish time, a partial order could be defined over clips. To simplify notation, $C_i = [t_{s_i}, t_{f_i}]$ is used to mean that clip C_i is associated with a time interval $[t_{s_i}, t_{f_i}]$. Semantically, C_i is the set of all the frames within this interval.

Definition 6 Let $C_i = [t_{s_i}, t_{f_i}]$ and $C_j = [t_{s_j}, t_{f_j}]$ be two clips. Then $\preceq$ is defined as the partial order over clips with $C_i \preceq C_j$ iff $t_{s_i} \leq t_{s_j}$ and $t_{f_i} \leq t_{f_j}$. Also, $C_i \prec C_j$ iff $t_{f_i} < t_{s_j}$.

Definition 7 A set of clips C is said to be *ordered* iff C is finite, i.e., $C = \{C_1, \ldots, C_m\}$ and there exists a partial order such that $C_1 \preceq C_2 \preceq \ldots \preceq C_m$. A set of clips $C = \{C_1, C_2, \ldots, C_m\}$ is said to be *strongly ordered* iff C is ordered and $C_1 \prec C_2 \prec \ldots \prec C_m$. A set of clips $C = \{C_1 \ldots, C_m\}$ is said to be *perfectly ordered* iff C is ordered and for any two neighboring clips $C_i = [t_{s_i}, t_{f_i}]$ and $C_{i+1} = [t_{s_{i+1}}, t_{f_{i+1}}]$, $t_{s_{i+1}} = t_{f_i} + 1$ $(\forall i = 1, 2, \ldots, m-1)$. A set of clips $C = \{C_1 \ldots, C_m\}$ is said to be *weakly ordered* iff C is ordered and C is neither strongly ordered nor perfectly ordered.

Example 4 (a) $C = \{[1, 10], [13, 17], [28, 100]\}$ is an ordered clip set because $[1, 10] \preceq [13, 17] \preceq [28, 100]$, and a strongly ordered set because $[1, 10] \prec [13, 17] \prec [28, 100]$. However, it is not perfectly ordered since $t_{s_2} = 13 \neq 11 = t_{f_1} + 1$.
(b) $C = \{[1, 10], [2, 8], [13, 17]\}$ is not ordered since $[1, 10] \not\preceq [2, 8]$ and $[2, 8] \not\preceq [1, 10]$.
(c) $C = \{[1, 10], [11, 17], [18, 100]\}$ is a perfectly ordered clip set because C is ordered and $t_{s_2} =$

$t_{f_1}+1, t_{s_3}=t_{f_2}+1$. It is easy to verify that C is also strongly ordered.
(d) $C=\{[1,10],[8,17],[15,30]\}$ is a weakly ordered clip set because C is ordered and C is neither strongly ordered ($t_{f_1}=10 \geq t_{s_2}=8$) nor perfectly ordered ($t_{s_2} \neq t_{f_1}+1$).

The above examples indicate that ordered clip sets disallow subintervals (set C in Example 3.1(b)). In the CVOT model only weakly ordered and perfectly ordered clip sets are considered because the associated intervals of their clips can be merged into larger intervals. This is important in the case of a stream-based representation. The following theorem states the relationship between strongly ordered clip sets and perfectly ordered clip sets.

Theorem 2 All perfectly ordered clip sets are also strongly ordered.

Proof: Let $C=\{[t_{s_1},t_{f_1}],\ldots,[t_{s_m},t_{f_m}]\}$ be a perfectly ordered clip set. This means $[t_{s_1},t_{f_1}] \preceq \ldots \preceq [t_{s_m},t_{f_m}]$ with $t_{s_{i+1}}=t_{f_i}+1$ $(i=1,\ldots,m-1)$ and $t_{s_i} \leq t_{s_{i+1}}$ and $t_{f_i} \leq t_{f_{i+1}}$ $(i=1,\ldots,m-1)$. Hence, from $t_{s_i} \leq t_{f_i} < t_{s_{i+1}} \leq t_{s_{i+1}}$ $(i=1,\ldots,m-1)$ we have $[t_{s_i},t_{f_i}] \prec [t_{s_{i+1}},t_{f_{i+1}}]$ $(i=1,\ldots,m-1)$. Therefore, $[t_{s_1},t_{f_1}] \prec \ldots \prec [t_{s_m},t_{f_m}]$. From the definition of strongly ordered clip sets it follows that C is strongly ordered. ■

Note that the reverse of this theorem is not necessarily true, i.e. strongly ordered clip sets may not be perfectly ordered. This is shown in Example 3.1(a).

3.3.2 Salient Object Set

Each video frame has many salient objects, e.g. persons, houses, cars, etc. Assuming there is always a finite set of salient objects $SO=\{SO_1,SO_2,\ldots,SO_n\}$ for a given video, the spatial property of an SO_i is captured by a *spatial object*, which is defined by an MBR (X_i,Y_i) where $X_i=[x_{s_i},x_{f_i}], Y_i=[y_{s_i},y_{f_i}]$ in a given frame.

Let $\mathcal{SO}$ be the collection of all salient object sets and $\mathcal{C}$ be the collection of all clip sets. Two new functions are introduced. One is the function $\mathcal{F}$: $SO \to \mathcal{C}$ which maps a salient object from $SO \in \mathcal{SO}$ into an ordered clip set $C \in \mathcal{C}$. The other is the function $\mathcal{F}^*$: $C \to \mathcal{SO}$ which maps a clip from $C \in \mathcal{C}$ into a salient object set $SO \in \mathcal{SO}$. Intuitively, function $\mathcal{F}$ returns a set of clips which contains a particular salient object while the reverse function $\mathcal{F}^*$ returns a set of salient objects which belong to a particular clip. The *common salient objects* for a given clip set is defined as those salient objects which appear in every clip within the set. Some salient objects may appear in many different clips, but others may not. Hence, the number of common salient objects between clips are different. In order to quantify such a difference, clip *affinity* is introduced.

Definition 8 The *affinity* of m clips $\{C_1,\ldots,C_m\}$ is defined as

$$aff(C_1,\ldots,C_m)=|\mathcal{F}^*(C_1)\cap\mathcal{F}^*(C_2)\cap\ldots\cap\mathcal{F}^*(C_m)|$$

where $\{C_1, \ldots, C_m\}$ is an ordered clip set, $m \geq 2$, $|X|$ is the cardinality of set X, and $\cap$ is the standard set intersection.

Example 5 Figure 3.2 shows a video in which John is using bat $\mathsf{bat1}$ and Ken is using bat $\mathsf{bat2}$ to play table tennis while Mary watches. After the game, John drives home in his car. Assume that the salient objects are $SO = \{\mathsf{john}, \mathsf{ken}, \mathsf{mary}, \mathsf{ball}, \mathsf{bat1}, \mathsf{bat2}, \mathsf{car}\}$. If the video is segmented as in Figure 3.2, then $C = \{C_1, C_2, C_3, C_4, C_5\}$ is a perfectly ordered clip set. Furthermore, john, ball, and $\mathsf{bat1}$ are in C_1; john, mary, ball, $\mathsf{bat1}$ are in C_2; ken, ball, $\mathsf{bat2}$ are in C_3; ken, ball, $\mathsf{bat2}$ are in C_4; and john, car are in C_5. Then,

$$\begin{array}{ll}
\mathcal{F}(\mathsf{john}) = \{C_1, C_2, C_5\} & \mathcal{F}^*(C_1) = \{\mathsf{john}, \mathsf{ball}, \mathsf{bat1}\} \\
\mathcal{F}(\mathsf{ken}) = \{C_3, C_4\} & \mathcal{F}^*(C_2) = \{\mathsf{john}, \mathsf{ball}, \mathsf{bat1}, \mathsf{mary}\} \\
\mathcal{F}(\mathsf{mary}) = \{C_2\} & \mathcal{F}^*(C_3) = \{\mathsf{ken}, \mathsf{ball}, \mathsf{bat2}\} \\
\mathcal{F}(\mathsf{ball}) = \{C_1, C_2, C_3, C_4\} & \mathcal{F}^*(C_4) = \{\mathsf{ken}, \mathsf{ball}, \mathsf{bat2}\} \\
\mathcal{F}(\mathsf{bat1}) = \{C_1, C_2\} & \mathcal{F}^*(C_5) = \{\mathsf{john}, \mathsf{car}\} \\
\mathcal{F}(\mathsf{bat2}) = \{C_3, C_4\} & \\
\mathcal{F}(\mathsf{car}) = \{C_5\}. &
\end{array}$$

Now, the affinity of C_1 and C_2 is

$$\begin{aligned}
\mathit{aff}(C_1, C_2) &= |\mathcal{F}^*(C_1) \cap \mathcal{F}^*(C_2)| = |\{\mathsf{john}, \mathsf{ball}, \mathsf{bat1}\} \cap \{\mathsf{john}, \mathsf{ball}, \mathsf{bat1}, \mathsf{mary}\}| \\
&= |\{\mathsf{john}, \mathsf{ball}, \mathsf{bat1}\}| = 3.
\end{aligned}$$

Similarly, $\mathit{aff}(C_2, C_3) = 1$, $\mathit{aff}(C_1, C_2, C_3) = 1$, etc.

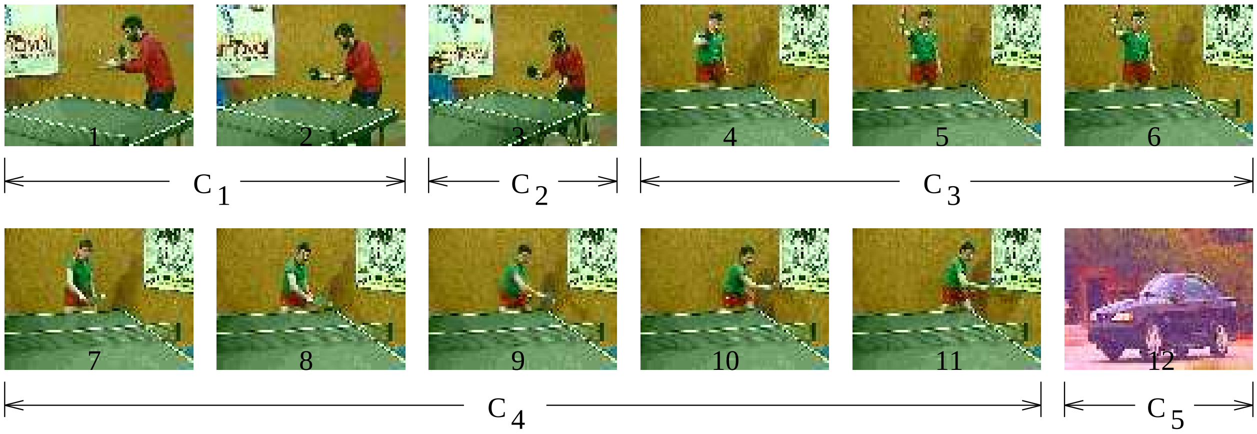


Figure 3.2: Salient Objects and Clips

Theorem 3 The affinity function is *monotonically non-increasing*. That is, if $\{C_1, \ldots, C_m\}$ is an ordered clip set, then $\mathit{aff}(C_1, \ldots, C_k) \geq \mathit{aff}(C_1, \ldots, C_k, C_{k+1})$ where $k = 2, 3, \ldots, m - 1$.

Proof: The proof follows from the set intersection property, $A \cap B \subseteq A$, where A and B are any two sets. Let set A be $\mathcal{F}^*(C_1) \cap \ldots \cap \mathcal{F}^*(C_k)$ and set B be $\mathcal{F}^*(C_{k+1})$. Then, $\mathcal{F}^*(C_1) \cap \ldots \cap \mathcal{F}^*(C_k) \cap \mathcal{F}^*(C_{k+1}) \subseteq \mathcal{F}^*(C_1) \cap \ldots \cap \mathcal{F}^*(C_k)$. It follows that $|\mathcal{F}^*(C_1) \cap \ldots \cap \mathcal{F}^*(C_k) \cap \mathcal{F}^*(C_{k+1})| \leq |\mathcal{F}^*(C_1) \cap \ldots \cap \mathcal{F}^*(C_k)|$. Therefore, $\mathit{aff}(C1, \ldots, C_k, C_{k+1}) \leq \mathit{aff}(C_1, \ldots, C_k)$. ∎

3.3.3 The Common Video Object Tree

Clustering clips is an important issue as it affects both the effectiveness and efficiency of query retrievals. A clustering scheme should also maintain any existing temporal relationships among frames. CVOT meets these requirements. For a given set of clips a tree structure is built upon the common salient objects. Trees provide an easy and efficient way of clustering clips with less complexity than graphs. For any given weakly or perfectly ordered clip set C, each leaf node in a CVOT tree is an element of C. All the leaf nodes are ordered from left to right by their time intervals while an internal node represents a set of common salient objects, which appear in all its child nodes. The only node that can have an empty common salient object set is the root node or a node of the clip with an empty salient object set. Every node (including internal, leaf, and root node) has a time interval and a set of salient objects which appear during this time interval. The time interval of an internal node has a start time which is equal to the start time of its immediate leftmost child node and a finish time which is equal to the finish time of the immediate rightmost child node. Figure 3.3 shows an example of a CVOT tree which is built from Example 3.2. As seen in Figure 3.3, the cardinality of the common object set shrinks as the tree is traversed from the leaf nodes to the root. This is in conformance with the monotonically non-increasing nature of clip affinity stated in Theorem 3.2. The figure also shows how the time intervals are propagated up from the leaf nodes. For example, the internal node N_2 has the interval $[1, 3]$ which is composed of its two child leaf nodes C_1 and C_2. The root always spans all of the time intervals in the whole clip set.

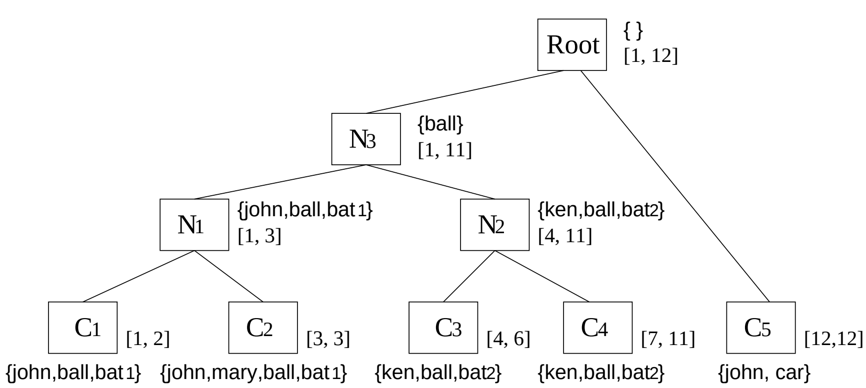


Figure 3.3: A Common Video Object Tree Built by GMCO Algorithm

```
GMCO(C, SO, R)
C = {C_1, ..., C_m}: a weakly or perfectly ordered clip set;
SO = {SO_1, ..., SO_n}: a salient object set;
R: root node of a CVOT tree;
{
    NewC := ∅; /* a new ordered clip set */
    while (C ≠ ∅) {
        if (C == {C_1}) {
            Attach C_1 to R; /* set C_1 to be a child node of R */
            C = C - {C_1};
        } else {
            if (aff(C_1, C_2) == 0) {
                Attach C_1 to R;
                C = C - {C_1};
            } else {
                Temp = {C_1, C_2};
                Expand(C - Temp, Temp, aff(C_1, C_2));
                Create a new node N and Attach C_1, C_2 to N;
                NewC = NewC ∪ {N};
                w = |Temp|; /* (Temp = {C_1, C_2, ..., C_w}) */
                Assign F*(C_1) ∩ F*(C_2) ∩ ... ∩ F*(C_w) into N;
                Assign [t_{s_{C_1}}, t_{f_{C_w}}] into N;
                C = C - Temp;
            } /* end of if */
        } /* end of if */
    } /* end of while */
    if (NewC ≠ ∅)
        GMCO(NewC, SO, R);
    return R;
}
```

Figure 3.4: Greedy Maximum Common Objects Algorithm

$\mathbf{Expand}(C,\ T,\ AFF)$
$C = \{C_{r+1}, \ldots, C_m\}$: an ordered clip set;
$T = \{C_1, \ldots, C_r\}$: a new ordered clip set and $r \geq 2$;
AFF: affinity of T (integer constant);
{
    if ($C == \emptyset$) **return**;
    if ($\mathit{aff}(C_1, \ldots, C_r, C_{r+1}) \geq AFF$)
        $\mathbf{Expand}(C - \{C_{r+1}\},\ T \cup \{C_{r+1}\},\ AFF)$;
    else
        return;
}

Figure 3.5: Expand Subroutine

A greedy algorithm, called *Greedy Maximum Common Objects* (GMCO), is developed to build a CVOT (Figure 3.4). Given a set of clips C, the GMCO algorithm builds the CVOT in a bottom-up fashion. The idea is to find the largest set of neighboring clips of the given set without reducing its affinity. An internal node is then created for this new set of clips. This process continues until either the common object set is empty or all nodes are merged into one node (root). If the common object set is empty, this node is directly attached to the root of the CVOT.

The algorithm first checks if the clip set is a singleton. If so, the element is attached directly to the root. If the clip set has more than one element, each of these makes a leaf node. The next step is to compute the largest clip set without reducing the affinity of this set. This is done by checking whether the affinity of two neighboring clips is zero. If so, the clip set is not expanded because a larger clip set will only decrease its affinity. If the affinity of two neighboring clips is not zero, this value is set to be the initial affinity of the clip set. Then, a subroutine **Expand** is called to compute the largest clip set. **Expand**(C, T, *AFF*) shown in Figure 3.5 recursively expands a common object set T by selecting more elements from the common object set C as long as the affinity is not smaller than an integer value *AFF*. Here, sets C and T are disjoint while the initial value of *AFF* is the affinity of the first two clips. The correctness of the subroutine **Expand** is guaranteed by Theorem 3, i.e., the monotonically non-increasing nature of the clip affinity. Finally, algorithm **GMCO** recursively builds the tree by adding another level.

Example 6 The CVOT instance in Figure 3.3 is built from Example 3.2 by the **GMCO** algorithm. In Figure 3.3, node C_1 has time interval $[1, 2]$ and a set of salient objects {john, ball, bat1}; node C_2 has time interval $[3, 3]$ and a set of salient objects {john,mary, ball, bat1}; node C_3 has time interval $[4, 6]$ and a set of salient objects {ken, ball, bat2}; node C_4 has time interval $[7, 11]$ and a set of salient objects {ken, ball, bat2}; node C_5 has time interval $[12, 12]$ and a set of salient objects {john, car}. The affinity of C_1 and C_2 is 3 while the affinity will be 1 if C_3 is added. Therefore,

C_1 and C_2 should have a parent node N_1 with a time interval $[1, 3]$ and a salient object set {john, ball, bat1}. Similarly, node N_2 has time interval $[4, 11]$ and a set of salient objects {ken, ball, bat2}. Node C_5 has to be attached directly to the root node because it is the only one left in the clip set. Since the affinity of N_1 and N_2 is 1, a new internal node N_3 is created as shown in Figure 3.3. Then N_3 is directly attached to the root node. Hence, the *Root* node has time interval $[1, 12]$ and an empty salient object set.

CVOT effectively clusters frames, but further optimizations are possible. In the above example, if a query asks for clips in which *John* appears, the result is $\{C_1, C_2, C_5\}$. CVOT clusters C_1 and C_2 under N_1, but does not cluster C_5. To address this problem, an inverted list can be used to further index CVOT nodes as shown in Figure 3.6. Such an array is called a CVOTArray. Each element of a CVOTArray is a salient object and consists of a linked list which links all the nodes where the salient objects appear. However, in any linked list, if there is a CVOT internal node, all its children will not be in this list so redundancy is eliminated. The cost of accessing such an array is constant time, if a hash function is used. Hence the cost of searching for a salient object in clips is a constant time. The cost of searching for all the clips where the salient object appears is less than the number of its appearances.

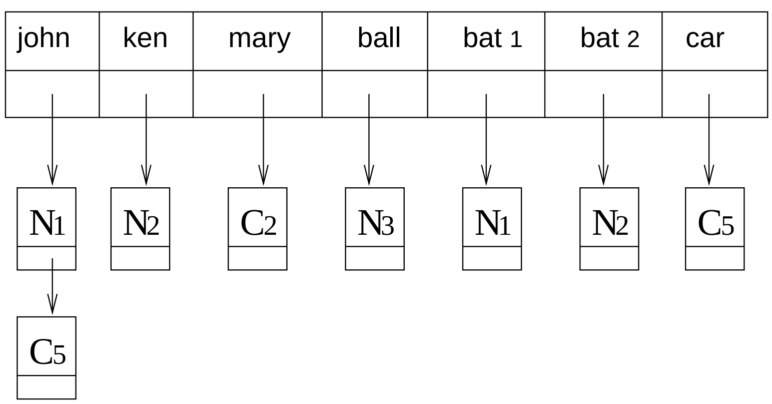


Figure 3.6: A CVOTArray

3.4 The ODBMS Support

CVOT is an abstract model. To have proper database management support for continuous media, this model is integrated into the TIGUKAT data model. The TIGUKAT model [ÖPS+95, Gor98] is a *behavioral* and *uniform* object model. The model is *behavioral* in the sense that all accesses and manipulations of objects are based on the application of behaviors to objects. The model is *uniform* in that every component of information, including its semantics, is modeled as a *first-class* *object* with well-defined behaviors. The typical object-oriented features such as abstract types, strong typing, complex objects, multiple inheritance, and parametric types are also supported.

The primitive objects of the model include: *atomic entities* (reals, integers, strings, etc.); *types* for defining common features of objects; *behaviors* for specifying the semantics of operations that may be performed on objects; *functions* for specifying implementations of behaviors over types; *classes* for automatic classification of objects based on types; and *collections* for supporting general heterogeneous groupings of objects.

In this thesis, a reference prefixed by "`T_`" refers to a type, "*C_*" to a class, "*B_*" to a behavior, and "`T_X`< `T_Y` >" to the type `T_X` parameterized by the type `T_Y`. For example, `T_person` refers to a type, *C_person* to its class, *B_age* to one of its behaviors and `T_collection`< `T_person` > to the type of collections of persons. A reference such as David, without a prefix, denotes an object instance for some other application specific reference. Consequently, the model separates the definition of object characteristics (a *type*) from the mechanism for maintaining instances of a particular type (a *class*).

3.4.1 The Temporal Object Model

Temporality has been added to the TIGUKAT model [GÖS97, Gor98] as type and behavior extensions of its type system. The following is a brief overview of the temporal ontology and temporal history features of this model. These features are relevant to the integration of the CVOT model into this particular temporal object model. Figure 3.7 gives part of the time type hierarchy that includes the temporal ontology and temporal history features of the temporal model.

A wide range of operations on temporal intervals is provided. Unary operators which return the lower bound, upper bound and length of the time interval are defined. The model supports a rich set of ordering operations among intervals [All83] (they are depicted in Table 2.1), e.g., *before*, *overlaps*, *during*, etc., as well as set-theoretic operations, e.g., *union*, *intersection* and *difference*[1]. A time interval can be expanded or shrunk by a specified time duration. Different kinds of open, closed, half open and half closed intervals are modeled.

A *time instant* (*moment*, *chronon*, etc.) is a specific anchored moment in time. A time instant is modeled as a special case of a (closed) time interval which has the same lower and upper bound, e.g., $24Jan1996 = [24Jan1996, 24Jan1996]$. A time instant can be compared with another time instant with the transitive comparison operators "<" and ">". A time duration can be added to or subtracted from a time instant to return another time instant. A time instant can be compared with a time interval to check if it falls before, within or after the time interval.

A *time span* is an unanchored relative duration of time. A time span is basically an atomic

[1]Note that the union of two disjoint intervals is not an interval. Similarly, for the difference operation, if the second interval is contained in the first, the result is not an interval. In the temporal model, these cases are handled by returning an object of the *null* type (`T_null`).

T_interval	B_lb:	T_instant
	B_ub:	T_instant
	B_length:	T_span
	B_before:	T_interval → T_boolean
	B_equal:	T_interval → T_boolean
	B_during:	T_interval → T_boolean
	B_meets:	T_interval → T_boolean
	B_overlaps:	T_interval → T_boolean
	B_starts:	T_interval → T_boolean
	B_finishes:	T_interval → T_boolean
	B_union:	T_interval → T_interval
	B_intersection:	T_interval → T_interval
	B_difference:	T_interval → T_interval
	B_subtract:	T_span → T_interval
	B_add:	T_span → T_interval
T_instant	B_lessthaneqto:	T_instant → T_boolean
	B_greaterthaneqto:	T_instant → T_boolean
	B_elapsed:	T_instant → T_span
	B_subtract:	T_span → T_instant
	B_add:	T_span → T_instant
	B_intersection:	T_interval → T_instant
	B_difference:	T_interval → T_instant
	B_shrink:	T_span → T_instant
	B_succ:	T_instant
	B_pred:	T_instant
T_span	B_lessthan:	T_span → T_boolean
	B_greaterthan:	T_span → T_boolean
	B_lessthaneqto:	T_span → T_boolean
	B_greaterthaneqto:	T_span → T_boolean
	B_add:	T_span → T_span
	B_subtract:	T_span → T_span
	B_succ:	T_span
	B_pred:	T_span
T_history<T_X>	B_history:	T_collection<T_timeStampedObject<T_X>>
	B_insert:	T_X,T_interval → T_boolean
	B_validObjects:	T_interval → T_collection<T_timeStampedObject<T_X>>
T_timeStampedObject<T_X>	B_value:	T_X
	B_timeStamp:	T_interval

Table 3.1: Behaviors on Time Intervals, Time Instants, and History

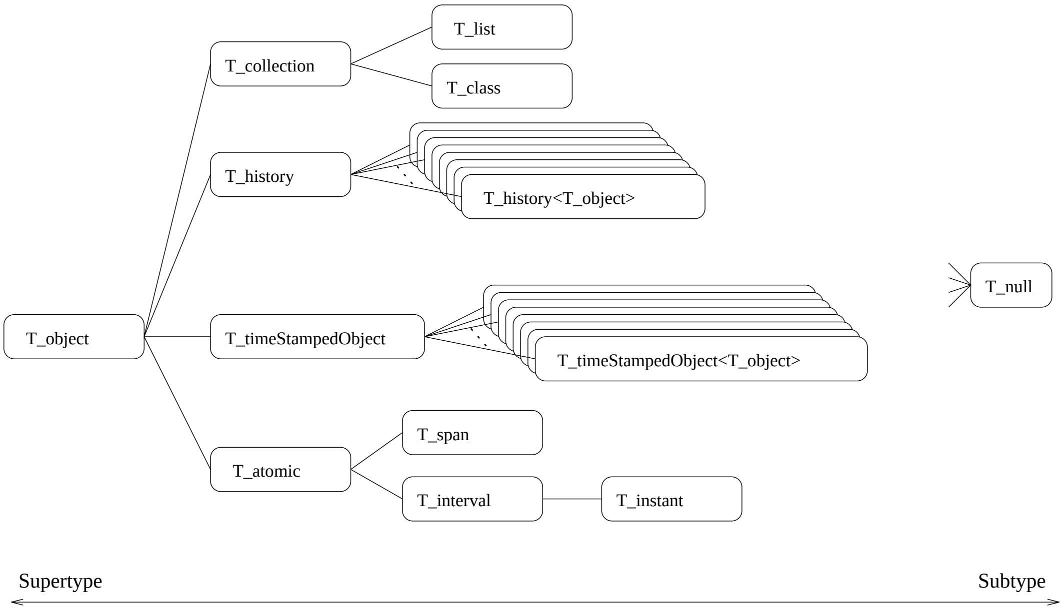


Figure 3.7: The Basic Time Type Hierarchy

cardinal quantity, independent of any time instant or time interval. Time spans have a number of operations defined on them. A time span can be compared with another time span using the transitive comparison operators "<" and ">". A time span can be subtracted from or added to another time span to return a third time span. The detailed behavior signatures corresponding to the operations on time intervals, time instants, and time spans are given in the Table 3.1.

3.4.2 Temporal Histories

One requirement of a temporal model is an ability to adequately represent and manage histories of objects and real-world activities. The model represents the temporal histories of objects whose type is `T_X` as objects of the `T_history<T_X>` type as shown in Figure 3.7. A temporal history consists of objects and their associated timestamps (time intervals or time instants). A *timestamped object* knows its timestamp and its associated object (value) at (during) the timestamp. A temporal history is made up of such objects. Table 3.1 gives the behaviors defined on histories and timestamped objects. Behavior *B_history* defined on `T_history<T_X>` returns the set (collection) of all timestamped objects that comprise the history. Another behavior defined on history objects, *B_insert*, timestamps and inserts an object in the history. The *B_validObjects* behavior allows the user to get the objects in the history that are valid at (during) the given time.

Each timestamped object is an instance of the `T_timeStampedObject<T_X>` type. This type represents objects and their corresponding timestamps. Behaviors *B_value* and *B_timeStamp* defined in `T_timeStampedObject` return the value and the timestamp of a timestamped object, respectively.

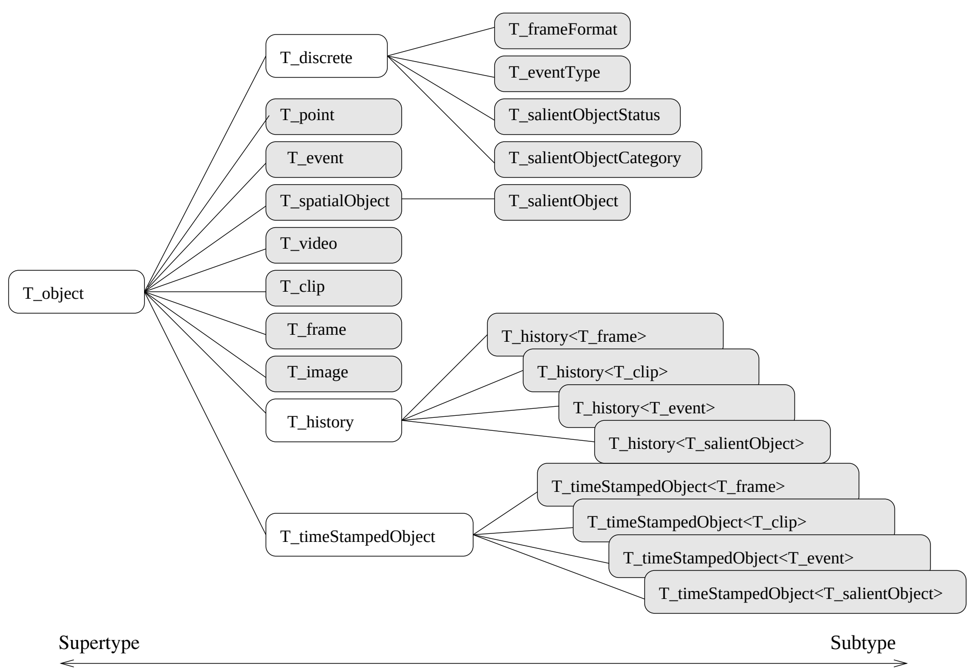


Figure 3.8: The Video Type System

3.4.3 System Integration

Integrated multimedia systems that use a uniform object model have simplified system support and possibly better performance. In such a system, the multimedia component can directly use many functions provided by the ODBMS, such as concurrency control, data recovery, access control etc. In this section, the integration of the CVOT model into TIGUKAT is discussed. Temporal histories are used to model the various features of the CVOT model and the contents of a video. Figure 3.8 shows the proposed video type system. The types that are in a grey shade are directly related to this video model and they will be discussed in detail throughout this section.

3.4.3.1 Integrating the Spatial Model

Table 3.2 shows the primitive behavior signatures of spatial objects. Any object occupying some space is an instance of `T_spatialObject`. `T_spatialObject` defines an object's 2D intervals that are computed from the projections of the object's MBR over x and y axes. The behavior *B_centroid* returns the centroid of the object while the behavior *B_area* returns the area of the region occupied by the object. The distance between objects at a certain time and the displacement of an object over time intervals are captured by *B_distance* and *B_displacement*, respectively. The rest of the behaviors are related to the directional and topological relations that have been discussed in Chapter 2. For example, if `p` is a spatial object, then the centroid of `p` is `p`.*B_centroid*. If `q` is another spatial object. Then `p`.*B_overlap*(`q`) returns *true* (a boolean value) if `p` overlaps `q`. Otherwise, it

`T_spatialObject`	*B_xinterval*:	`T_interval`
	B_yinterval:	`T_interval`
	B_centroid:	`T_point`
	B_area:	`T_real`
	B_displacement	`T_interval`, `T_interval` $\rightarrow$ `T_real`
	B_distance	`T_spatialObject`, `T_interval` $\rightarrow$ `T_real`
	B_south:	`T_spatialObject` $\rightarrow$ `T_boolean`
	B_north:	`T_spatialObject` $\rightarrow$ `T_boolean`
	B_west:	`T_spatialObject` $\rightarrow$ `T_boolean`
	B_east:	`T_spatialObject` $\rightarrow$ `T_boolean`
	B_northwest:	`T_spatialObject` $\rightarrow$ `T_boolean`
	B_northeast:	`T_spatialObject` $\rightarrow$ `T_boolean`
	B_southwest:	`T_spatialObject` $\rightarrow$ `T_boolean`
	B_southeast:	`T_spatialObject` $\rightarrow$ `T_boolean`
	B_left:	`T_spatialObject` $\rightarrow$ `T_boolean`
	B_right:	`T_spatialObject` $\rightarrow$ `T_boolean`
	B_below:	`T_spatialObject` $\rightarrow$ `T_boolean`
	B_above:	`T_spatialObject` $\rightarrow$ `T_boolean`
	B_equal:	`T_spatialObject` $\rightarrow$ `T_boolean`
	B_inside:	`T_spatialObject` $\rightarrow$ `T_boolean`
	B_overlap:	`T_spatialObject` $\rightarrow$ `T_boolean`
	B_cover:	`T_spatialObject` $\rightarrow$ `T_boolean`
	B_touch:	`T_spatialObject` $\rightarrow$ `T_boolean`
	B_disjoint:	`T_spatialObject` $\rightarrow$ `T_boolean`
`T_point`	*B_xvalue*:	`T_real`
	B_yvalue:	`T_real`

Table 3.2: Primitive Behavior Signatures of Spatial Objects

returns *false* (another boolean value). An instance of `T_point` defines two real values *B_xvalue* and *B_yvalue*.

3.4.3.2 Integrating the CVOT Model

In Figure 3.8 the `T_video` type is defined to model videos. An instance of `T_video` has all the semantics of a video. As discussed in Section 3.3, a video is segmented into a set of clips. Since a clip set is ordered and each clip has an associated time interval, it is natural to model this set as a history. A clip set is modeled by defining the behavior *B_clips* in `T_video`. *B_clips* returns a history object of type `T_history`$<$ `T_clip` $>$, the elements of which are timestamped objects of type `T_clip`.

Example 7 Suppose myVideo is an instance (object) of `T_video`. Then,

- myVideo.*B_clips* returns an instance (object) of type `T_history`$<$ `T_clip` $>$. Let this object be myVideoClipHistory.
- myVideoClipHistory.*B_history* returns a collection (clip set) which contains all the timestamped clip objects of type `T_timeStampedObject`$<$ `T_clip` $>$ in myVideo. Let one of these

clip history objects be myVideoCHOneClip.

- myVideoCHOneClip.*B_value* returns the clip object of myVideoCHOneClip, while myVideoCHOneClip.*B_timeStamp* returns the time interval of myVideoCHOneClip.

Table 3.3 gives the behavior signatures of videos.

`T_video`	*B_clips*:	`T_history<T_clip>`
	B_cvotTree:	`T_cvotTree`
	B_weaklyOrdered:	`T_boolean`
	B_perfectlyOrdered:	`T_boolean`
	B_stronglyOrdered:	`T_boolean`
	B_length:	`T_span`
	B_publisher:	`T_collection<T_company>`
	B_producer:	`T_collection<T_person>`
	B_date:	`T_instant`
	B_play:	`T_boolean`
`T_clip`	*B_frames*:	`T_history< T_frame >`
	B_salientObjects:	`T_collection<T_timeStampedObject<T_salientObject>>`
	B_activities:	`T_collection<T_timeStampedObject<T_activity>>`
	B_affinity:	`T_list<T_clip>` $\rightarrow$ `T_integer`
	B_type:	`T_videoType`
	B_play:	`T_boolean`
`T_frame`	*B_type*:	`T_frameType`
	B_content:	`T_image`
`T_cvotTree`	*B_search*:	`T_salientObject`, `T_cvotTree` $\rightarrow$ `T_cvotTree`

Table 3.3: Behavior Signatures of Videos, Clips, and Frames

The behavior *B_cvotTree* returns the common video object tree of a video. For example, myVideo.*B_cvotTree* creates a CVOT tree from the clip set of myVideo. The implementation of *B_cvotTree* is the GMCO algorithm discussed in Subsection 3.3.3.

In the CVOT model, a video knows the ordering of its clips. This ordering is defined by several video behaviors: *B_weaklyOrdered*, *B_perfectlyOrdered* and *B_stronglyOrdered* which simply iterate over the time intervals in the clip set history and determine whether a clip history is weakly ordered, perfectly ordered, or strongly ordered respectively.

A common question to myVideo would be its length (duration). This is modeled by the *B_length* behavior and it returns an object of type `T_span`. If a video is segmented into a perfectly ordered clip set, its length is equal to the total length of all the clips in this set. However, if the clip set is weakly ordered or strongly ordered, the video length is not equal to the total length of all the clips because in such clip sets, clips may overlap or be disjoint.

Video information should also include metadata, such as the publishers, producers, publishing date, etc. A video can also be played by using *B_play*.

Each clip has a set of consecutive frames, which is modeled by `T_history<T_frame>`. Since a clip must be associated with a time interval, clips are treated as timestamped objects. Suppose myClip is a particular clip, then myClip is an instance (object) of type `T_timeStampedObject< T_clip >`. The content of myClip is myClip.*B_value* while the interval of myClip is myClip.*B_timeStamp*. Some behavior signatures of clips are shown in Table 3.3.

All the salient objects within a clip are grouped by the behavior *B_salientObjects* which returns a collection of timestamped salient objects `T_collection < T_timeStampedObject < T_salientObjects >>`. Since a salient object can appear several times within a clip, such distinct appearances must be captured using the timestamps.

It is legitimate to ask a clip's affinity (*B_affinity*) with other clips. *B_play* of `T_clip` is able to play a clip on an appropriate output device. Other related operations, such as *stop, pause, play backward*, etc., are omitted from the table because they are not important to the discussion. Such an omission is also applicable to the behaviors of `T_video`. The existence of a video enumerated type `T_videoType` is assumed.

The basic building unit of a clip is the frame which is modeled by `T_frame` in Table 3.3. Frames are modeled within a clip as a history which is the same way to model clips within a video. Many different types of frames may exist, e.g. predicted frames, intracoded frames, and bidirectional frames in MPEG videos [Gal91]. This is defined by the behavior *B_type* of `T_frame`. The content of a frame, *B_content*, is an image which defines many image properties such as width, height and color.

3.4.3.3 Modeling Video Features

The semantics or contents of a video are usually expressed by its *features* which include video attributes and the relationships between these attributes. Typical video features are salient objects and *activities*[2]. An *activity* may involve many different salient objects over a time period, like holding a party, playing table tennis, and chatting with someone.

An activity can occur in different places either within a clip or across multiple clips. For example, the activity johnDrive may occur in multiple clips. Additionally, this activity may occur several times within a clip. Therefore, an appropriate representation is necessary to capture the temporal semantics of general activities. A simple and natural way to model the temporal behavior of activities is to use a historical structure. Thus, the history of activities is modeled as objects of type `T_history < T_activity >`. Instances, such as johnDrive, of `T_history< T_activity >` consist of timestamped activities. This allows us to keep track of all the activities occurring within a video, although an activity may occur in multiple clips or just occur within one clip. In the interest of tracking all the activities occurring within a clip, the behavior *B_activities* is included in `T_clip`.

Similarly, since salient objects can also appear multiple times in a clip or a video, the history

[2]Some authors refer to the activities as *events*. However, the term *event* is reserved to refer to an activity that occurs at an instant for consistency with standard literature in physics and mathematics.

of a salient object is modeled as objects of type T_history< T_salientObject >. The behavior *B_salientObjects* of T_clip returns all the salient objects within a clip. Using histories to model salient objects and activities results in a higher level abstraction in object-oriented design. Furthermore, it enables us to uniformly capture the temporal semantics of video data because a video is modeled as a history of clips and a clip is modeled as a history of frames.

T_activity	*B_activityType*:	T_activityType
	B_track:	T_histry<T_activity>
	B_inClips:	T_video → T_collection< T_timeStampedObject < T_clip >>
	B_activityObjects:	T_collection<T_salientObject>
T_salientObject	*B_boundingBox*:	T_boundingBox
	B_centroid:	T_history<T_point>
	B_inClips:	T_video → T_collection< T_timeStampedObject < T_clip >>
	B_track:	T_history<T_salientObject>
	B_status:	T_status

Table 3.4: Some Behavior Signatures of Activities and Salient Objects

The behavior *B_activityType* of T_activity, shown in Table 3.4, identifies the type of activities T_activityType and the behavior *B_roles* indicates all the persons involved in an activity. *B_activityObjects* returns all the salient objects within an activity. *B_inClips* indicates all the clips in which this activity occurs. *B_track* results in all the occurrences of an activity. For example, johnDrive.*B_track* returns all the activities of johnDrive that happen at different times. It is certainly reasonable to include other information, such as the location and time of an activity, into type T_activity, but they are not important to this discussion.

The behavior *B_boundingBox* of type T_salientObject defines a bounding box on an object so its spatial information can be captured. *B_centroid* returns the centroid of a salient object and the behavior *B_inClips* returns all the clips in which the salient object appears. This corresponds to the reverse function $\mathcal{F}^*$ defined in Section 3.3. *B_track* of T_salientObject returns all the occurrences of a salient object and the CVOT model provides an excellent way of implementing such a behavior. *B_status* may be used to define some other attributes of a salient object. For example, it is very useful to know whether an object is *rigid* or not if we want to track the motion of the object. Here T_status is assumed to be such an enumerated type.

Chapter 4

The Moving Object Model

The most striking difference between still images and videos stems from movements and variations. Retrieving moving objects, which requires both spatial and temporal knowledge of objects, is part of content-based querying. Typical applications are automated surveillance systems, industrial monitoring, road traffic monitoring, video databases etc. Modeling moving objects has received some research attention recently [DG94, IB95, SAG95, ABL95, YYHI96] but it is certainly in its infancy. Most research focuses on tracking the movement of a single object, i.e., the *trajectory* of an object over a period of time, which is certainly very important. For example, object trajectories are needed for many useful video annotation tasks such as describing city street intersections, sporting events, pedestrian mall traffic, cell movements from quantitative fluorescence microscopy, groups of animals and meteorological objects [IB95]. However, another important aspect of moving objects is their relative spatial relationships over a timeline, which has not been studied. For example, a coach may have particular interest in the relative movement of his players during a game so that they can achieve the best cooperation. Alternatively, interests may exist in finding objects whose movements match user drawings in cases where it is difficult to verbally describe complex movement relations. Typical queries about moving objects can be: "*Find all the objects which have a similar trajectory to this one*", "*Find a video clip where a dog approaches a person from the left*", "*Find a video clip which matches a scene I sketched*", etc. These queries are so common that any video database should be able to model them.

In this chapter, a new way of qualitatively representing the trajectory of a moving object and the relative spatio-temporal relations between moving objects is introduced. This chapter also presents new algorithms for matching trajectories and spatio-temporal relations of moving objects to facilitate query processing. The algorithms can handle both exact and similarity matches. The resulting system supports a broad range of user queries about moving objects, especially for systems with graphical user interfaces.

4.1 Related Work

Egenhofer studies gradual changes of the topological relations over time within the context of GISs [EAT92]. It has been recognized that a qualitative change occurs if the deformation of an object affects its topological relation with respect to another object. A computational model is presented to describe the changes. Most importantly it reveals the internal relationships which are useful in describing the closeness of topological relations. The work represented in this chapter is, in a sense, an extension of [EAT92] by considering directional relations as well as topological relations and time.

Dimitrova and Golshani [DG94] describe a method of computing the trajectories of objects. Their objective is to discover motion using a dual hierarchy consisting of spatial and temporal parts for *video sequence* representation. Video sequences are identified by objects present in the scene and their respective motion. Motion vectors extracted during the motion compensation phase of video encoding are used. The motion information extraction is then used in the intermediate level by motion tracing and in the high level by associating an object and a set of trajectories with recognizable activities and events. They focus on trajectories of objects at a high level and do not study relations between moving objects.

Intille and Bobick propose an interesting model that uses a *closed-world assumption* to track object motions [IB95]. A closed-world is a region of space and time in which the specific context is adequate to determine all possible objects present in that region. Besides using closed-worlds to circumscribe the knowledge relevant to tracking, they also exploit them to reduce complexity. Two types of entities exist in a closed-world, *objects* and *image regions*. An important feature of this model is that the knowledge of the domain dictates how objects can interact and is independent of how the scene is captured for vision analysis. After an image region within a video frame is selected, each pixel within this region is assigned to one of the objects within its closed-world. Context-specific features are used to construct templates for tracking each moving object in the closed-world. Then, objects are tracked to the next frame using the templates. They describe a prototype for tracking football players. However, it is not clear how to decide a closed-world region. For specialized images, such as superimposed images or fade-in fade-out clips, this method will fail.

The Video Query By Example (V-QBE) system [YYHI96] only deals with the trajectory of a single moving object. A unified model for spatial and temporal information is proposed in [Wor94]. A *bitemporal relation*, including both event time and database time, is applied to objects in the system. This model is designed for GISs. There is also some research on moving objects that result from camera motions [ABL95, SAG95], such as *booming, tilting, panning, zooming*, etc. These types of movement are not considered in this thesis. More work on motion detection can be found in [BH94, GD95].

4.2 Modeling Moving Objects

A *moving object* is a salient object which moves its position over time. It is assumed that moving objects are rigid or consist of rigid parts connected together and these rigid parts are never disintegrated. For any moving object only eight possible directions are considered as shown in Figure 4.1(a).

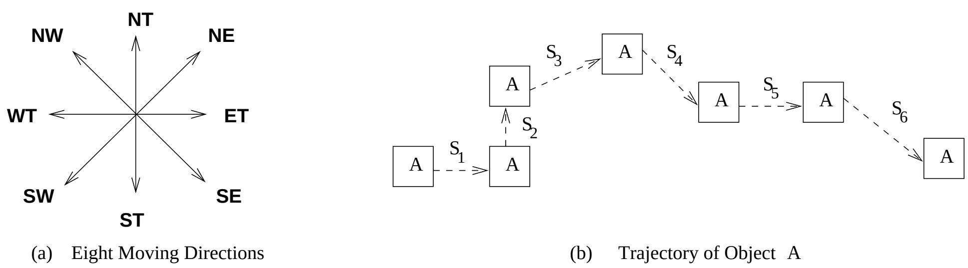


Figure 4.1: Moving Direction and Example

Definition 9 Let A be a moving object and the *motion* of A over time interval I_i is (S_i, d_i, I_i) where $S_i = DISP(A, I_i)$ is the displacement of A and d_i is a direction whose domain is the union of strict and mixed directional relations. For a given ordered list of time intervals $\langle I_1, I_2, \ldots, I_n\rangle$, the *trajectory* of A can be described by a list of motions

$$\langle (S_1, d_1, I_1), (S_2, d_2, I_2), \ldots, (S_n, d_n, I_n)\rangle.$$

Example 8 Figure 4.1(b) shows a trajectory of object A. The trajectory consists of a sequence of motions which can be expressed by

$$\langle (S_1, \texttt{ET}, I_1), (S_2, \texttt{NT}, I_2), (S_3, \texttt{NE}, I_3), (S_4, \texttt{SE}, I_4), (S_5, \texttt{ET}, I_5), (S_6, \texttt{SE}, I_6)\rangle.$$

Being able to model a moving object is quite useful from a querying perspective. However, this model is at a very low level in the sense that the computation of a displacement could be very expensive for each object in every frame of a video. Even if only sample representative frames are used, the processing is still time-consuming. Moreover, it is very difficult, if not impossible, to capture the relationships of a moving object with another salient object (either moving or non-moving). For example, to answer the query how close is the movement of A and B to the movement of C and D is not easy from the trajectories of A, B, C, and D. A model that can describe how an object moves and how an object relates to other objects is clearly more expressive than the above model. Such a model is introduced in this section.

Definition 10 Let A and B be two moving objects. Their *moving spatio-temporal relationship* (abbreviated as *mst-relation*) during time interval I_k is $A\ (\alpha, \beta, I_k)\ B$, such that $A\alpha B$ and $A\beta B$ are true at time interval I_k where α is any topological relation and β is any one of the 12 directional relations or `NULL`. `NULL` means no directional relation. For a given ordered list of time intervals $\langle I_1, I_2, \ldots, I_n\rangle$, all the mst-relations of A and B are defined by an *mst-list*:

$$\langle(\alpha_1, \beta_1, I_1), (\alpha_2, \beta_2, I_2), \ldots, (\alpha_n, \beta_n, I_n)\rangle.$$

From the definition, the spatial relationships over a time period between moving objects are captured by both their topological and directional relations. `NULL` is necessary because two objects may have no directional relation, such as when A is *inside* of B. The following example further explains the concept of an mst-relation and an mst-list. It also indicates that the mst-relations of two objects are neither unique nor symmetric.

Example 9 Figure 4.2 shows two moving objects approaching each other, overlapping, and then going away from each other. This might represent two friends meeting on a street, shaking hands, hugging, and then leaving each other. During time interval I_1, the mst-relation between A and B is $(\texttt{DJ}, \texttt{WT}, I_1)$. This relation changes to $A\ (\texttt{TC}, \texttt{WT}, I_3)\ B$ at interval I_3 and becomes $A\ (\texttt{TC}, \texttt{ET}, I_5)\ B$ at interval I_5. The mst-relation between B and A at interval I_3 is $(\texttt{TC}, \texttt{ET}, I_3)$ which is different from $A\ (\texttt{TC}, \texttt{WT}, I_3)\ B$. Hence, generally $A\theta B \neq B\theta A$ where θ is an mst-relation. Such an asymmetric property is caused by the directional relations. However, from the inverse property of all the directional relations, the mst-relation between B and A can be derived from the mst-relation between A and B. The mst-list of A and B over ordered time interval $\langle I_1, I_2, \ldots, I_6\rangle$ is $\langle(\texttt{DJ}, \texttt{WT}, I_1), (\texttt{DJ}, \texttt{WT}, I_2), (\texttt{TC}, \texttt{WT}, I_3), (\texttt{OL}, \texttt{NULL}, I_4), (\texttt{TC}, \texttt{ET}, I_5), (\texttt{DJ}, \texttt{ET}, I_6)\rangle$. Since, $A\,\texttt{WT}\,B$ can deduce $A\,\texttt{LT}\,B$ and $A\,\texttt{ET}\,B$ can deduce $A\,\texttt{RT}\,B$ according to the definitions of spatial directional relations, another mst-list for A and B can be:
$\langle(\texttt{DJ}, \texttt{LT}, I_1), (\texttt{DJ}, \texttt{LT}, I_2), (\texttt{TC}, \texttt{LT}, I_3), (\texttt{OL}, \texttt{NULL}, I_4), (\texttt{TC}, \texttt{RT}, I_5), (\texttt{DJ}, \texttt{RT}, I_6)\rangle$.

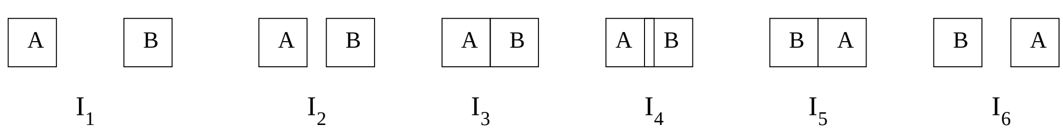


Figure 4.2: Two Moving Objects

4.3 Matching Moving Objects

Matching trajectories and mst-lists is useful in handling user queries. In this subsection two algorithms are designed to accomplish such a task. When a video DBMS is queried, a user may ask "*Is there any object whose trajectory matches the trajectory of object A?*". A's trajectory, denoted by $\langle M_1, M_2, \ldots, M_m\rangle$, can be given through a graphical user interface. Therefore, a systematic way is needed to find in the database any matching object which satisfies the above condition. The problem can be restated slightly differently: Does the trajectory of A match the trajectory $\langle N_1, N_2, \ldots, N_n \rangle$ of object B? To facilitate the retrieval of a particular motion in a trajectory, a set of linked lists are used to represent the trajectory of B. Each list head corresponds to a directional relation and each entry consists of an integer and a pointer to the next element. The integer represents the relative order of a particular displacement in the trajectory. For example, the first entry in Figure 4.3 indicates that the object is displaced in `NT` direction as the second move in its trajectory. Figure 4.3 shows a linked list representation for the trajectory of Figure 4.1(b).

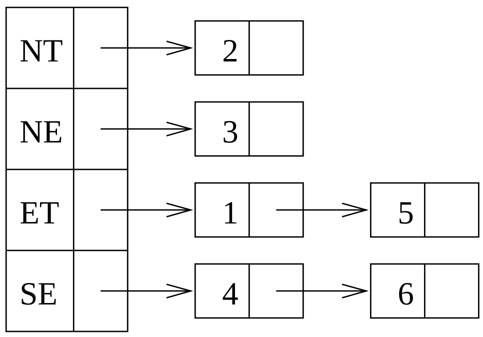
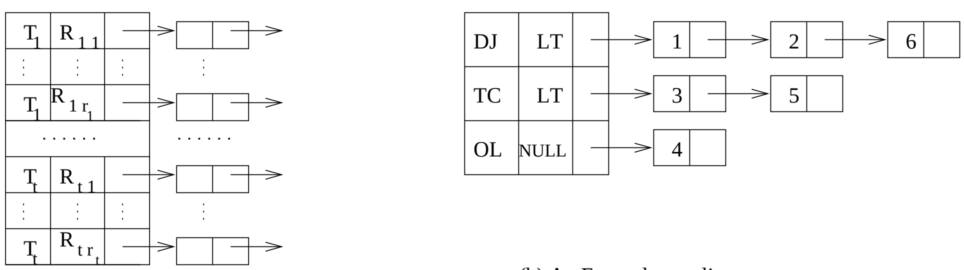


Figure 4.3: Data Structures for Trajectory



(a) Link List for an mst-set

(b) An Example mst-list

Figure 4.4: Data Structures for mst-list

Similarly a linked list representation can have an mst-list of two moving objects. The only change is to accommodate directional relations. The data structure of the linked list for mst-lists is shown in Figure 4.4(a). The first column, T_t, is a topological relation with $1 \leq t \leq 8$ and the second column R_{tr_i} is a directional relation or `NULL` with $1 \leq r_i \leq 13$ $(1 \leq i \leq t)$. The example mst-list of Figure 4.2 is shown in Figure 4.4(b).

Now it is possible to introduce the algorithms for matching objects' trajectories. Such algorithms are useful in some applications. For example, in hockey for a goal to count, no offensive team player can enter the small crease in front of the goalie when the shot is taken. However, in many cases, the referee cannot see exactly what had happened during a fraction of a second. So he has to rely on the result of two video specialists whose job is to analyze some recorded videos and tell the referee exactly what had happened. According to National Hockey League (NHL) statistics (see www.nhl.com), there are 2 to 3 such calls in each NHL game. Using the algorithms described here, this can be done automatically and with no possible human judgment error. Figure 4.5 describes an algorithm to test whether object A's trajectory matches object B's trajectory. The structure *linklist* is exactly the same structure as in Figure 4.3. Statement (5) indicates that if the number of motions of A is bigger than the number of motions of B, then A does not match B. Although in this case B may match A, this is considered to be a different case. Statements (6) and (7) look for B's first motion which is exactly the same as A's first motion. If there is no such match, FALSE is returned. Otherwise, the proper head pointer of B's linked list is located and from there a sequential comparison is conducted within the trajectories of A and B.

A similar algorithm for mst-lists is described in Figure 4.6. The only change is the data structure of the linked list. Here, the topological relations are considered. The data values are captured by TOPID (topological) and DIRID (directional) in the algorithm. This change results in some related changes in mst-list comparisons. The time complexities of the trajectory match algorithm and mst-relation match algorithms are easy to compute. The worst case of both algorithms is within the nested **while** statements. However, since the outer **while** statement is actually a constant (9 for the trajectory match algorithm and 104 for the mst-relation match algorithm in worst case), it can be ignored. As the inner **while** statement depends on n which is the number of object's moves, the time complexities of both algorithms are $O(n)$.

These two algorithms are exact match algorithms. It is well-known that exact match queries *normally* generate few results in multimedia systems. Therefore, fuzzy (similarity) queries must be supported. A query is *fuzzy* if the properties of objects being queried are not precisely defined (like *a big region*) or the comparison operators in the query cannot provide exact matches. Again consider object trajectories first. In the case of object displacement, the similarity of two displacements can be measured by a predefined *tolerance*. However, in the case of directional relations a new measurement metric has to be introduced to describe the similarity. The solution to this problem is to manually assign a *distance value* between any two directional relations as shown in Table 4.1. For example, $distance(\mathtt{NT}, \mathtt{NW}) = 1$. The way these values are assigned is completely determined by their *closeness* to each other. For example, *northwest* and *northeast* should have the same closeness value to direction *north*. They should be *closer* to the *north* than *west*, *east*, and *south* are. The smallest value of a distance is zero which is an exact match and the biggest value of a distance is four which is the opposite direction. For example, the distance of north (NT) versus south (ST) is 4.

Definition 11 Let $\{M_1, M_2, \ldots, M_m\}$ $(m \geq 1)$ be the trajectory of A, $\{N_1, N_2, \ldots, N_n\}$ be the trajectory of B, and $m \leq n$. $minDiff(A, B)$ is the smallest distance between A and B calculated as

```
TrajectoryMatch(A, B)
INPUT: A = {M_1, M_2, ..., M_m}: object A's trajectory
         B = {N_1, N_2, ..., N_n}: object B's trajectory
OUTPUT: TRUE /* A and B's trajectories match */
          FALSE /* A and B's trajectories do not match */
    int i, k = 1;
    struct linklist { int ID; struct linklist *next; }
          DIR[8]; /* The linked list of B's trajectory */
    struct linklist x; /* temporal variable */
    if (m > n) return FALSE;
    while (DIR[k] ≠ empty AND M_1 ≠ DIR[k].ID) k++;
    if (DIR[k] == empty) return FALSE;
    x = DIR[k].next;
    while (x ≠ empty) {
        i = 1;
        while (i ≤ m AND (i + x.ID) < n) {
            if (M_i ≠ x.ID) break;
                i++;
        } /* end of inner while */
        if (i > m) return TRUE;
        x = x.next;
    } /* end of outer while */
    return FALSE;
```

Figure 4.5: Trajectory Match Algorithm

follows:

$$minDiff(A, B) = MIN \{\sum_{i=1}^{m} distance(M_i, N_{i+j})\} \quad (\forall j \ 0 \leq j \leq n - m).$$

The biggest difference between A and B happens only if A's moving direction is always opposite to B's moving directions in all the comparisons. Such a case can be quantified by $maxDiff(A, B) = 4 * m$ since the maximum number of comparing motions is m. In order to present a uniform standard for measuring similarity to end users, a normalized similarity function for the trajectory of A and B is defined as

$$TrajSim(A, B) = \frac{maxDiff(A, B) - minDiff(A, B)}{maxDiff(A, B)}.$$

Function $TrajSim(A, B)$ defines a similarity degree of the trajectories of A and B and its domain is $[0, 1]$. For example, in the case $minDiff(A, B) = 0$, $TrajSim(A, B) = 1$ which indicates an exact match. In the case $minDiff(A, B) = maxDiff(A, B)$, it must be $TrajSim(A, B) = 0$ which indicates that A always moves in the opposite direction of B, and that similarity between A and B's trajectories is minimum.

```
MstMatch(A, B)
INPUT: A = {M1, M2, ..., Mm}: object A's trajectory
       B = {N1, N2, ..., Nn}: object B's trajectory
OUTPUT: TRUE /* A and B's trajectories match */
        FALSE /* A and B's trajectories do not match */
    int i, k = 1;
    struct linklist { int TOPID, DIRID; struct linklist *next; }
        TOPDIR[8 * 13]; /* The linked list of B's trajectory */
    struct linklist x; /* temporal variable */
    if (m > n) return FALSE;
    while (TOPDIR[k] ≠ empty∧(M1 ≠ TOPDIR[k].TOPID∨
            M1 ≠ TOPDIR[k].DIRID)) k++;
    if (TOPDIR[k] == empty) return FALSE;
    x = TOPDIR[k].next;
    while (x ≠ empty) {
        i = 1;
        while (i ≤ m ∧ (i + x.TOPID)< n) {
            if (Mi ≠ x.TOPID∨Mi ≠ x.DIRID) break;
                i++;
        } /* end of inner while */
        if (i > m) return TRUE;
        x = x.next;
    } /* end of outer while */
    return FALSE;
```

Figure 4.6: MST-Relation Match Algorithm

The similarity function of two mst-lists can be defined in a similar manner. The directional relations must be extended to include topological relations and `NULL`, as presented in Table 4.2. The distance values of positional relations are assigned in the same manner as it was done for other directional relations. The distance values between `NULL` and directional relations are assigned as an average distance among others, which is 2. In the case of assigning distance values for topological relations the problem is more complicated. Fortunately, a distance scheme of topological relations is presented in [EAT92] which is adopted as Table 4.3. The rationale of this scheme is based on the *closeness* of these relations when they evolve from one to another. For example, from relation *disjoint* to relation *overlap* there must exist the state *touch* sometime in between. Hence, the distance from *touch* to *disjoint* or *overlap* is considered closer than the distance from *disjoint* to *overlap*. Consequently, the similarity function for mst-lists can be computed using a function analogous to $TrajSim$.

Definition 12 Let $\{M_1, M_2, \ldots, M_m\}$ $(m \geq 1)$ be the mst-list of A, $\{N_1, N_2, \ldots, N_n\}$ be the

	NT	NW	NE	WT	SW	ET	SE	ST
NT	0	1	1	2	3	2	3	4
NW	1	0	2	1	2	3	4	3
NE	1	2	0	3	4	1	2	3
WT	2	1	3	0	1	4	3	2
SW	3	2	4	1	0	3	2	1
ET	2	3	1	4	3	0	1	2
SE	3	4	2	3	2	1	0	1
ST	4	3	3	2	1	2	1	0

Table 4.1: Distances of Moving Directions

	NT	NW	NE	WT	SW	ET	SE	ST	LT	RT	AB	BL	NULL
NT	0	1	1	2	3	2	3	4	3	3	1	4	2
NW	1	0	2	1	2	3	4	3	2	4	2	4	2
NE	1	2	0	3	4	1	2	3	4	2	2	4	2
WT	2	1	3	0	1	4	3	2	1	4	3	3	2
SW	3	2	4	1	0	3	2	1	2	4	4	2	2
ET	2	3	1	4	3	0	1	2	4	1	3	3	2
SE	3	4	2	3	2	1	0	1	4	2	4	2	2
ST	4	3	3	2	1	2	1	0	3	3	4	1	2
LT	3	2	4	1	2	4	4	3	0	4	2	2	2
RT	3	4	2	4	4	1	2	3	4	0	2	2	2
AB	1	2	2	3	4	3	4	4	2	2	0	4	2
BL	4	4	4	3	2	3	2	1	2	2	4	0	2
NULL	2	2	2	2	2	2	2	2	2	2	2	2	0

Table 4.2: Distances of Directional Relations

	DJ	TC	EQ	IN	CB	CT	CV	OL
DJ	0	1	6	4	5	4	5	4
TC	1	0	5	5	4	5	4	3
EQ	6	5	0	4	3	4	3	6
IN	4	5	4	0	1	6	7	4
CB	5	4	3	1	0	7	6	3
CT	4	5	4	6	7	0	1	4
CV	5	4	3	7	6	1	0	3
OL	4	3	6	4	3	4	3	0

Table 4.3: Distances of Topological Relations (Table 1 in [EAT92])

mst-list of B, and $m \leq n$. *minDiff* (A, B) is the smallest distance between A and B:

$$\mathit{minDiff}(A,B) \;=\; MIN\ \{\sum_{i=1}^{m} distance(M_i, N_{i+j})\}\quad (\forall j\ 0 \leq j \leq n-m)$$

where $distance(M_i, N_{i+j}) = distance(\alpha(M_i, N_{i+j})) + distance(\beta(M_i, N_{i+j}))$ where α and β extract the topological relation and the directional relation of an mst-relation respectively. $\mathit{maxDiff}(A, B) = 4*m+7*m$ since the maximum distance value of a topological relations is 7. An mst-list similarity function is defined as

$$MstSim(A,B) = \frac{\mathit{maxDiff}(A,B) - \mathit{minDiff}(A,B)}{\mathit{maxDiff}(A,B)}.$$

Example 10 Let $A = \{(\texttt{DJ}, \texttt{LT}, I_1), (\texttt{TC}, \texttt{LT}, I_2), (\texttt{TC}, \texttt{LT}, I_3), (\texttt{OL}, \texttt{NULL}, I_4), (\texttt{DJ}, \texttt{RT}, I_5)\}$ be object A's mst-list and $B = \{(\texttt{DJ}, \texttt{LT}, I_1), (\texttt{DJ LT}, I_2), (\texttt{TC}, \texttt{LT}, I_3), (\texttt{OL}, \texttt{NULL}, I_4), (\texttt{TC}, \texttt{RT}, I_5), (\texttt{DJ}, \texttt{RT}, I_6)\}$ be object B's mst-list. B's mst-relation to A is the same as shown in Figure 4.2. Therefore, $m = 5$, $n = 6$, $n - m = 1$, and $\mathit{maxDiff}(A, B) = 4*m+7*m = 55$. For $j = 0$,

$$\sum_{i=1}^{5} distance(A_i, B_{i+0}) = (0+0)+(1+0)+(0+0)+(0+0)+(1+4) = 6$$

and for $j = 1$,

$$\sum_{i=1}^{5} distance(A_i, B_{i+1}) = (0+0)+(1+0)+(1+2)+(3+2)+(0+4) = 13.$$

Hence $\mathit{minDiff}(A, B) \;=\; MIN\ \{6, 13\} \;=\; 6$. The mst-relation similarity of A and B is:

$$MstSim(A,B) = \frac{\mathit{maxDiff}(A,B) - \mathit{minDiff}(A,B)}{\mathit{maxDiff}(A,B)} = \frac{55-6}{55} = 0.89.$$

`T_salientObject`	*B_trajectory*:	`T_trajectory`
	B_mstSet:	`T_salientObject` → `T_mstSet`
`T_trajectory`	*B_exactMatch*:	`T_trajectory` → `T_boolean`
	B_simMatch:	`T_trajectory` → `T_real`
	B_subtrajectory:	`T_interval` → `T_trajectory`
`T_mstSet`	*B_exactMatch*:	`T_mstSet` → `T_boolean`
	B_simMatch:	`T_mstSet` → `T_real`
	B_submstSet:	`T_interval` → `T_mstSet`
`T_motion`	*B_displacement*:	`T_interval` → `T_real`
	B_moveDirection:	`T_moveDirection`
`T_mstRelation`	*B_topology*:	`T_topologicalRelation`
	B_direction:	`T_directionalRelation`
	B_interval:	`T_interval`

Table 4.4: Additional Behavior Signatures for Moving Objects

4.4 Integrating Moving Object Model

Table 4.4 shows the additional behavior signatures to accommodate moving objects. In type `T_salientObject`, *B_trajectory* of `T_salientObject` returns type `T_trajectory` which is a list of moving object's motions (`T_list` < `T_motion` >). A *list* is an ordered collection. Similarly, the behavior *B_mstSet* returns type `T_mstSet` which is a list of two moving objects' mst-relations (`T_list<T_mstRelation>`). *B_exactMatch* is the exact match algorithm (for either trajectories or mst-lists) described in Figure 4.5 and Figure 4.6. *B_simMatch* of `T_trajectory` returns the similarity degree of two trajectories, which is captured by the similarity function $trajSim(A, B)$. The returned value is a real number between 0 and 1. The behavior *B_simMatch* of `T_mstSets` is the similarity function $mstSim(A, B)$ for two moving objects' mst-list. *B_subtrajectory* and *B_submstSet* returns part of a trajectory and part of an mst-list, respectively, for a given time interval. `T_motion` describes one motion of a moving object while `T_mstRelation` describes one mst-relation of two moving objects. `T_moveDirection`, `T_topologicalRelation`, and `T_directionalRelation` are enumerated types and they represent the eight moving directions, the eight topological relations, and the twelve directional relations plus `NULL` respectively.

Example 11 Figure 4.7 shows a video in which John and Mary walk toward their house. Later, Mary rides a horse on a ranch with her colt and dog. Let mary and dog be two timestamped salient objects. Their spatial relations at time t (or frame t) can be decided by first binding mary and dog to a common time interval. That is, assuming t is a time interval t (whose starting time and ending time are t) and both t.*B_during*(mary.*B_timeStamp*) and t.*B_during*(dog.*B_timeStamp*) are true. Then the spatial intervals of mary and dog are compared according to the definitions given in Table 2.2 to check what topological and directional relations exist. These spatial intervals of mary can be extracted by mary.*B_value*.*B_xinterval* and mary.*B_value*.*B_yinterval*. Similarly, dog.*B_value*.*B_xinterval* and dog.*B_value*.*B_yinterval* are the spatial intervals of dog. The trajectory of dog is expressed by dog.*B_trajectory*. The mst-list of dog and mary is captured by dog.B_mstSet(mary).

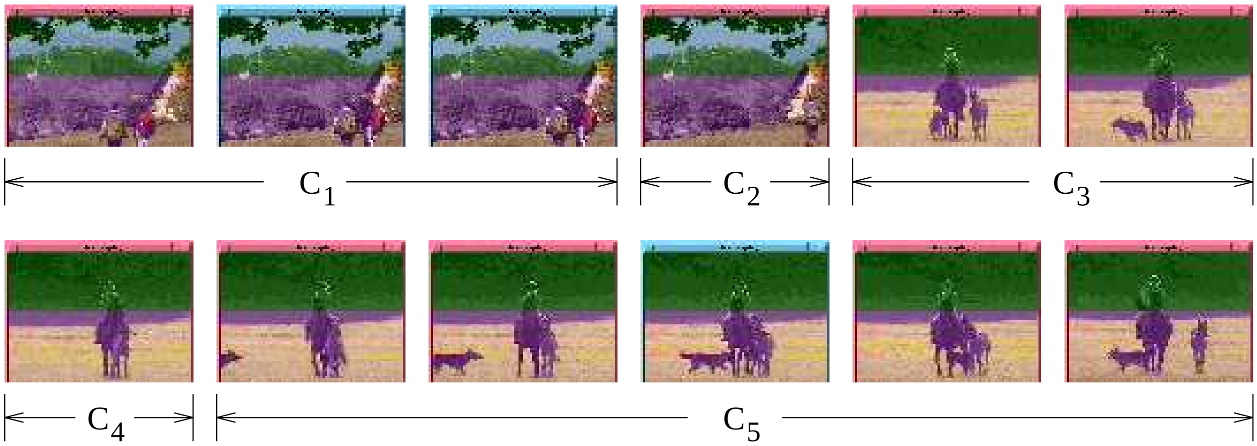


Figure 4.7: Video Clips

Chapter 5

The Multimedia Query Language

5.1 Introduction

One of the basic functionalities of a DBMS is to be able to efficiently process declarative user queries. The penetration of DBMS technology into multimedia information systems necessitates the development of query languages appropriate for this domain. The complex spatial and temporal relationships inherent in the wide range of multimedia data types make a multimedia query language quite different from its counterpart in traditional DBMSs. For example, the query languages of traditional DBMSs only deal with exact-match queries on conventional data types. Although this might be sufficient to deal with queries posed against metadata or annotations of multimedia data, content-based information retrieval requires non-exact-match queries which go beyond the traditional approaches.

5.1.1 Querying Multimedia Data

Powerful query languages significantly help simplify multimedia database access. These languages must provide constructs for querying, over a variety of multimedia attributes. Furthermore, *query presentation*, which refers to the way query results are presented, is more complex in multimedia systems than in traditional DBMSs. This is because multimedia presentations have to take into account the synchronization of various media. There have been a number of multimedia query language proposals [BRG88, OM88, RFS88, AB91, DG92, CIT$^+$93, Ege94, Güt94, CIT94, HK95, KC96, ATS96, MS96a, GR96, MHM96], which can be classified into three categories:

- Entirely new and specialized languages: [CIT$^+$93, HK95, KC96, ATS96, GR96].
- Languages that are based on a logical or functional programming approach: [DG92, MS96a].
- Languages that are extensions of SQL: [BRG88, OM88, RFS88, AB91, OT93, Ege94, Güt94, CIT94, MHM96].

The majority of existing approaches to designing multimedia query languages are extensions of SQL. This is generally due to the popularity of SQL for traditional database applications. A common problem with all the existing SQL-based multimedia query languages is that they are designed either for a particular medium or for a specific application domain, not for general use. For example, VideoSQL [OT93] is used only for video databases, SEQL [CIT94] is designed mainly for medical sequence image databases, ESQL [AB91] is good only for image databases, and PSQL [RFS88] and SpatialSQL [Ege94] are suitable only for spatial databases.

In this chapter a general multimedia query language is described, which is independent of particular media and specific applications. The language, called MOQL, is based on Object Query Language (OQL) [Cat94] that has been proposed by Object Database Management Group (ODMG) for ODBMSs. New features that MOQL introduces include support for temporal queries, spatial queries, structured document queries, and query presentations.

5.1.2 Querying Structured Documents

A related MDBMS research topic is the development of applications to store and retrieve multimedia documents effectively. Multimedia objects are frequently encapsulated in structured documents and the ability to query these multimedia documents are important. The topic has received significant research attention [BRG88, Mac90, YA94, CACS94, GMP95, ÖSEMV95, YIU96, BA96, ACC+97, ÖEMI+97] due to the importance of the issue for a wide class of applications such as software engineering, digital libraries, technical documentation, Internet communication (e.g., WWW) etc.

The Standard General Markup Language (SGML) [ISO86] has become the *de facto* standard for structured document creation and exchange. This is because of its suitability for a broad range of applications, its relative power, its widespread use (for example, the Hypertext Markup Language, HTML, that is the basis of WWW) and its role as the basis of another international standard HyTime [ISO92] for hypermedia documents. An SGML document has a hierarchical structure satisfying a document type definition (DTD).

The domain of structured documents has been developed in a manner that preserves the document hierarchical structure: a higher-level document object is formed from lower-level objects. Documents are represented by the description of their schema often mixed with the description of their contents. Data models and retrieval operations have been proposed to support these representations and to manipulate documents. Mostly, data models employ a tree model to represent the hierarchical document structure and the retrieval method is implemented by the sophisticated full text searching and the possibility of accessing a document component without knowing the full path to this component.

Although structured documents are easier to manage and exchange than unstructured documents, they are not easy to query for general users. The reason is that a user must know the structures of documents, i.e., the schema of the underlying database. In fact, much research in modeling and querying structured documents focuses on how to help users query the database without precise (or complete) knowledge of document structure. Several approaches [BRG88, CACS94] are proposed to attack this problem, but none of them gives details of how the approach may work internally for a structured document database, i.e., the procedural evaluation of queries allowing imprecise specifications.

In this chapter, extensions to OQL for querying structured document databases are constructed. Consequently, MOQL allows users to query structured documents without precise knowledge of the structure using a *functional approach*. A unique feature of MOQL is the provision of combined support for both document content retrieval and media content retrieval, which is missing from all the previous approaches.

5.2 Related Work

5.2.1 Multimedia Querying Languages

PSQL (Pictorial SQL) [RFS88] is designed for pictorial databases which require efficient and direct spatial search, based on the geometric form of spatial objects and relationships. It allows a user to directly manipulate spatial objects. An important feature of PSQL is the introduction of many spatial operators, such as *nearest* and *furthest* for point objects, *intersect* and *not-intersect* for segment objects, and *cover*, and *overlap* for region objects. Syntactically, PSQL is not much difference from the standard SQL.

EVA [DG92] is an object-oriented language, based on functional language features with roots in conventional set theory. It is formally defined using the mathematical framework of a many sorted algebra. Although EVA has defined a set of spatio-temporal operators to support query presentation, it lacks some useful presentation features, such as changing display speeds and time constraints (e.g., presenting some object for 20 minutes). Furthermore, EVA does not support spatial queries or video data.

A knowledge-based object-oriented query language, called PICQUERY$^+$, is proposed in [CIT$^+$93]. It is a high-level domain-independent query language designed for image and alphanumeric database management. It allows users to specify conventional arithmetic queries as well as evolutionary and temporal queries. The main PICQUERY$^+$ operations include panning, rotating, zooming, superimposing, color transforming, edge detecting, similarity retrieving, segmenting, and geometric operations. A template technique is introduced in PICQUERY$^+$ to facilitate user queries. Such *query templates* are used to specify predicates to constrain the database view.

SEQL (Spatial Evolutionary Query Language) [CIT94], a direct extension of SQL, is proposed to operate on the spatial evolutionary domains of medical images. In addition to alphanumeric predicates, SEQL contains constructs to specify spatial, temporal, and evolutionary conditions. A **when** clause is added to the language, which selects the appropriate snapshot of the data of interest at a particular point in time. It supports temporal functions which manipulate time points (such as *start time*, *end time* etc.), temporal ordering of an object history (such as *first*, *last*, *next*, etc.), and temporal intervals (such as *before*, *after*, *during*, etc.). Another extension is the addition of a **which** clause which describes various evolutionary processes on a set of evolving objects. Unfortunately, SEQL supports only image databases.

Marcus and Subrahmanian [MS96a] have proposed a formal theoretical framework for characterizing MDBMSs. The framework includes a logical query language that integrates diverse media. This is a first attempt at formally characterizing MDBMS. The model is independent of any specific application domain and provides the possibility of uniformly incorporating both query languages and access methods, based on multimedia index structures. This model defines a special data structure, called a *frame*, which is used for data access. The query language is based on logic programming and it makes extensive use of predicates and functions. Such a query language is suitable as an intermediate query language between a higher level language (such as OQL) and a lower level language (such as an object algebra).

Two new query languages, MMQL (Multimedia Query Language) and CVQL (Content-based Video Query Language), for video databases are described in [KC96] and [ATS96] respectively. A major problem with MMQL is that it does not support spatial queries which are fundamental to a multimedia query language. CVQL is defined based on video frame-sequences. Therefore, to query a video database using CVQL, a user must have good knowledge about the video (or frame sequence) being queried. ESQL [AB91] is an image domain query language for the relational model. Some research [HK95, PS95] has also been done in supporting multimedia content specification and retrieval in the design of a multimedia query language.

5.2.2 Querying Structured Documents

Multimedia Office Server (MULTOS) project [BRG88] is one of the earliest projects investigating issues of managing and querying structured documents with multimedia content. A MULTOS document server contains a number of specialized components, each performing certain document processing functions. Among these, the *storage subsystem* provides access methods for document retrieval and allows the storage of large database values, while the *operation/structure translator* maps document level operations onto the data structures of the storage subsystem. The query processing can be described as follows: first, the system exploits the complex structures of the data objects described by a conceptual model; then the system exploits the combination of different access methods, defined on different types of components of the data objects, for query optimization in order to determine the optimal query execution strategy. The problem of querying without precise knowledge is solved by introducing a specialized *dot notation* (or path expression), called a *star notation* *. The meaning of the star notation is that there is something unknown, but don't care. The advantage of this approach is its simplicity and intuitiveness. The disadvantage is that it does not support querying document structures. Furthermore, there is no discussion of how the star notation might be evaluated in the system.

Maestro (Management Environment for Structured Text Retrieval and Organization) [Mac90] is another database which supports structured documents and provides a query language to retrieve and link information contained in these structures. The system is implemented based on a commercial text retrieval system. The proposed query language allows users to query sequences of structures. For example, a user may ask for the first sentence of the first paragraph of an article. However, the query language does not support querying without precise knowledge.

The VERSO system [CACS94] is an ODBMS that is extended to handle SGML documents. It is built on top of O_2 [Deu90] to exploit its sophisticated type system and extensible query language O_2SQL. A key element of the VERSO system is the introduction of paths as first class citizens, which allows queries that address both the content and the document structures. A unique feature of VERSO is that it models SGML constraints in the data model by introducing a *union type*. The data model is extended to accommodate this new union type. Unfortunately, no existing commercial database, including the latest version of O_2, supports union types. The strategy of how to evaluate a path variable is elaborated in a recent paper [ACC+97].

A different approach, *virtual element markup*, in modeling and querying SGML documents is proposed in [YIU96]. A virtual element markup is an ordinary SGML element markup with special starting and ending tags and can be viewed as a logical layer on top of the SGML framework. A preprocess is necessary to handle an SGML document with virtual element markup. Virtual element markup is introduced to allow different references to the same objects. By introducing new tags, this approach achieves the goal of managing and querying documents from the DTD part instead of from the database part as VERSO does. Virtual element markups provide a uniform and flexible mechanism to make reference links from elements of SGML documents to database objects. By using this mechanism, SGML attributes and their values of marked-up words can be transparently stored as database attributes, and hyperlinks between keywords in documents can be established. However, this system does not address the problem of querying without complete knowledge.

There are several reported ODBMSs that support structured documents. HyperStorM (Hypermedia Document Storage and Modeling) [BA96, BAK97] is a structured document database which uses various object-oriented technologies to manage structured documents. For example, query optimization rules specialized for multimedia documents are investigated [Gay97]. Another structured document database project [ÖSEMV95, ÖEMI+97] is built on top of ObjectStore and is capable of storing, within one database, different types of documents by accommodating multiple document type definitions. This is accomplished by dynamically creating object types according to element definitions in each DTD. These systems are focusing on modeling and storing structured documents, as well as developing tools for automatic document insertion.

5.3 Object Query Language and Its Extensions

OQL defines an orthogonal expression language, in the sense that all operators can be composed with each other as long as the types of the operands are correct. It deals with complex objects without changing the set construct and the select-from-where clause. It is close to SQL92 [ANSI92] with object-oriented extensions such as complex objects, object identity, path expressions, polymorphism, operation invocation, and late binding. It includes high-level primitives to deal with bulk objects like structures, lists, and arrays. As a stand-alone language, OQL allows users to query objects by using their names as entry points into a database. As an embedded language, OQL allows applications to query objects that are supported by the native programming language, using expressions that yield atoms, structures, collections, and literals. An OQL query is a function which returns an object whose type may be inferred from the operators contributing to the query expression. OQL has one basic statement for retrieving information:

select [**distinct**] projection_attributes
from query [[**as**] identifier] {, query [[**as**] identifier] }
[**where** query]
[**group by** partition_attributes]
[**having** query]
[**order by** sort_criterion {, sort_criterion }]

where projection_attributes is a list of attribute names whose values are to be retrieved by the query. In the **from** clause, a variable has to be bound to a set of objects, an extent, or a query. In the **where** clause, the query is a conditional search expression that identifies the objects to be retrieved by the query. However, the conditional search expression can be any OQL query. In OQL, *query* is a very general expression, as described in Appendix D, and only the **select** form of a query corresponds to an actual user query. Clauses **group by**, **order by**, and **having** have the same semantics as their counterparts in SQL.

Query examples are introduced to describe different features of OQL and MOQL. ODMG's conventions are followed: a class name has its first character capitalized, a class extent is represented by the class name with its plural form (E.g. Employee is the name of class Employee and Employees is its extent), an object is identified by a special font (E.g. object x is denoted by x).

Query 1 Retrieve the birth date of the employee whose name is *John*:

select e.birthDate
from Employees e
where e.name="John"

This query involves only one extent, *Employees*, listed in the **from** clause. The query selects all the employees from class *Employee* that satisfy the condition of the **where** clause, then projects the result on the *birthDate* attribute. The result is a set of birth dates.

Query 2 Return a set of structures consisting of the ages and salaries of employees, whose names are "John" and seniorities are greater than 20:

select **struct**(a:*f*.age, s:*f*.salary)
from *f* **in** (**select** *e* **from** Employees *e* **where** *e*.seniority > 20) **as** *e*
where *f*.name = "John"

This query shows that the **from** clause does not have to be a class extent; it may contain queries too. The result of this query is a literal of the type set<struct>, namely, set<struct(a: int, s: float)>.

Query 3 Retrieve the street addresses of the spouses of employees who live in Paris:

select *e*.spouse.address.street
from Employees *e*
where *e*.lives_in("Paris")

The **select** statement includes a *path expression* and a method invocation. A method can return a complex object or a collection that can be embedded in a complex path expression. If *spouse* is a method defined on the class *Employee* which returns an object of class *Person*, the result of the above query is the set of their spouses' street names for those employees who live in Paris. Although *spouse* is a method, it is traversed as if it were a relationship. Moreover, the **where** clause contains a method, *lives_in*, which has one parameter.

Although OQL may be used to query MDBMSs, but it is very limited. Application developers have define their own multimedia functions and predicates, which is not acceptable for general users. More query examples are presented here to illustrate possible queries over moving objects which have been presented in the previous chapter. It is assumed that all the queries are posted to a particular video instance *myVideo* and salient objects and activities are timestamped objects as discussed in the previous section.

5.3.1 Querying Moving Objects

Query 4 In clip c find all the objects which have a similar trajectory as shown in Figure 4.1(b) denoted by *myTraj*.

select *x.B_value*
from *x* **in** *c.B_SalientObjects*
where *x.B_value.B_trajectory.B_simMatch(myTraj)* > *r*

where c is a clip object instance provided by users. For each object x in clip c, the trajectory of x is checked against *myTraj*. Such a comparison is determined by the similarity matching function between this two trajectories. r is a predefined (or user-provided) threshold, valued between $[0, 1]$, to qualify a match.

Query 5 Find clips in which object a is to the left of object b and later the two exchange their positions.

select c
from x_1,x_2,y_1,y_2 **in** $c.B\_salientObjects$, c **in** C_clip
where $x_1.B\_value{=}a$ **and** $y_1.B\_value{=}b$ **and** $x_2.B\_value{=}a$ **and** $y_2.B\_value{=}b$ **and**
$x_1.B\_timeStamp = y_1.B\_timeStamp$ **and** $x_1.B\_value.B\_left(y_1.B\_value)$ **and**
$x_2.B\_timeStamp = y_2.B\_timeStamp$ **and** $y_2.B\_value.B\_left(x_2.B\_value)$ **and**
$x_2.B\_timeStamp.B\_after(x_1.B\_timeStamp)$

Suppose clip c is the one that is looking for. Then there must be two timestamped objects, denoted by x_1 and y_1 respectively, in c's salient object set so that x_1 is a and y_1 is b. Similarly, two other timestamped objects, denoted by x_2 and y_2 respectively, must exist in c's salient object set so that x_2 is a and y_2 is b. The major difference between x_1 and x_2 is their time stamps. Here it is required that x_2 appear later than x_1, i.e., $x_2.B\_timeStamp.B\_after(x_1.B\_timeStamp)$. Therefore, if x_1 is to the left of y_1 at the time $x_1.B\_timeStamp$ and y_2 is to the left of x_2 at time $x_2.B\_timeStamp$, it is certain that a and b have exchanged their directional positions. Such a query might be expressed by Figure 4.2 in Example 4.2 using a graphical user interface. Let *myMstSet* represent the specified mst-list, the query could be simplified as

select c
from x,y **in** $c.B\_salientObjects$, c **in** C_clip
where $x.B\_value{=}a$ **and** $y.B\_value{=}b$ **and**
$x.B\_value.B\_mstSet(y.B\_value).B\_exactMatch(myMstSet)$

Query 6 Find clips in which a dog approaches Mary from the left (see Example 4.4 at the end of Chapter 4).

select c
from x_1,x_2,y_1,y_2 **in** $c.B\_salientObjects$, c **in** C_clip
where $x_1.B\_value{=}dog$ **and** $y_1.B\_value{=}mary$ **and**
$x_2.B\_value{=}dog$ **and** $y_2.B\_value{=}mary$ **and**
$x_1.B\_timeStamp = y_1.B\_timeStamp$ **and** $x_1.B\_value.B\_left(y_1.B\_value)$ **and**
$x_2.B\_timeStamp = y_2.B\_timeStamp$ **and** $x_2.B\_value.B\_left(y_2.B\_value)$ **and**
$x_2.B\_timeStamp.B\_after(x_1.B\_timeStamp)$ **and**
$x_1.B\_value.B\_displacement(x_1.B\_timeStamp,\ x_2.B\_timeStamp) >= h_1$ **and**
$xy.B\_value.B\_displacement(x_1.B\_timeStamp,\ x_2.B\_timeStamp) <= h_2$ **and**

where *dog* and *mary* are the references of two instances of `T_salientObject`. Suppose clip c is the clip looked for and two salient objects, denoted by x_1 and x_2, are introduced to represent *dog* and to reflect different timestamps. The same strategy is used for the object *mary*. Then, the *dog*'s displacement is computed over the time period and enforced to be greater than some predefined value h_1 to ensure enough movement is achieved. Furthermore, the displacement of *mary* is also computed and is required to be less than a predefined value h_2. This particular requirement of *mary* is to guarantee that it is the dog approaching Mary from the left, instead of Mary approaching the dog from the right.

This query can also be expressed using the mst-list described in Figure 5.1 where such an mst-list is denoted as *dogMaryMstSet*. The query is

select c
from x,y **in** $c.B\_salientObjects$, c **in** C_clip
where $x.B\_value{=}dog$ **and** $y.B\_value{=}mary$ **and**
$x.B\_value.B\_mstSet(y.B\_value).B\_simMatch(dogMaryMstSet) >= r$

However, this mst-list does not distinguish whether it is the *dog* approaching *mary* from the left or it is *mary* approaching the *dog* from the right. An *after* constraint must be put into this expression. One way to solve this problem is to make sure that *mary*'s displacement changes very little over the time interval as was done before.

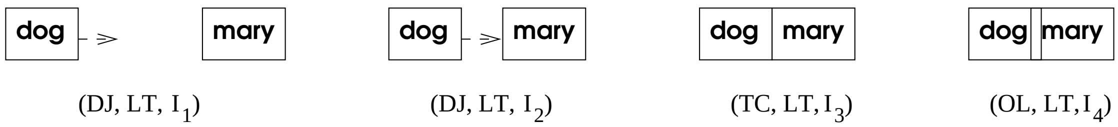


Figure 5.1: *dog* Approaches *mary* from Left

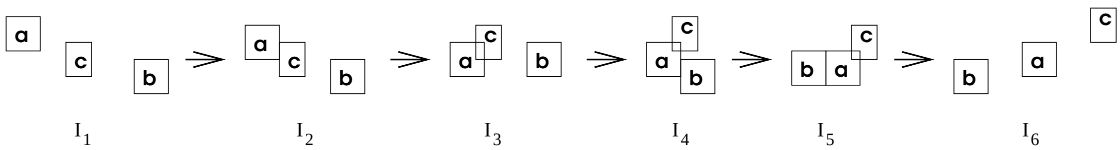


Figure 5.2: A Scene of Objects a, b, and c

Query 7 Retrieve all the clips which matches the scene described in Figure 5.2.

select	*clip*
from	*x,y,z* **in** *clip.B_salientObjects*, *clip* **in** *C_clip*
where	*x.B_value=a* **and** *y.B_value=b* **and** *z.B_value=c* **and**
	x.B_value.B_mstSet(y.B_value).B_exactMatch(abMstSet) **and**
	x.B_value.B_mstSet(z.B_value).B_exactMatch(acMstSet) **and**
	y.B_value.B_mstSet(z.B_value).B_exactMatch(bcMstSet)

Here, *abMstSet*, *acMstSet*, and *bcMstSet* are the mst-lists between a and b, a and c, and b and c. Furthermore, $abMstSet = \{(\texttt{DJ}, \texttt{NW}, I_1), (\texttt{DJ}, \texttt{NW}, I_2), (\texttt{DJ}, \texttt{LT}, I_3), (\texttt{TC}, \texttt{NW}, I_4), (\texttt{TC}, \texttt{ET}, I_5), (\texttt{DJ}, \texttt{NE}, I_6)\}$, $acMstSet = \{(\texttt{DJ}, \texttt{NW}, I_1), (\texttt{TC}, \texttt{NW}, I_2), (\texttt{OL}, \texttt{NULL}, I_3), (\texttt{OL}, \texttt{NULL}, I_4), (\texttt{OL}, \texttt{NULL}, I_5), (\texttt{DJ}, \texttt{SW}, I_6)\}$, and $bcMstSet = \{(\texttt{DJ}, \texttt{SE}, I_1), (\texttt{DJ}, \texttt{SE}, I_2), (\texttt{DJ}, \texttt{NE}, I_3), (\texttt{DJ}, \texttt{SE}, I_4), (\texttt{DJ}, \texttt{SW}, I_5), (\texttt{DJ}, \texttt{SW}, I_6)\}$. The scene in Figure 5.2 can be interpreted as part of a basketball game: at time I_1, players a and b are trying to catch the ball c, but a is faster so he touches the ball first and grabs it; since b does not get the ball, he has to try to block a's advance at time I_3; then players a and b collide with each other, but a is still holding the ball c; at time I_5, a manages to have passed b and shoots the ball. It is a very difficult query if it is expressed verbally. In summary, the query has been greatly simplified using the concept of moving spatial-temporal sets.

5.3.2 Basic Extensions

Most of the extensions introduced to OQL are in the **where** clause in the form of four new predicate expressions: *spatial_expression*, *temporal_expression*, *document_expression*, and *contains_predicate*.

A full BNF specification of these extensions is given in Appendix D. The *spatial_expression* is a spatial extension which includes spatial objects (such as points, lines, circles etc.), spatial functions (such as length, area, intersection, etc.), and spatial predicates (such as cover, disjoint, left etc.). A detailed discussion of spatial extensions is given in Section 5.4. The *temporal_expression* deals with temporal objects, temporal functions, and temporal predicates; these are discussed in Section 5.5. The *document_expression* defining extensions for querying structured documents is discussed in Section 5.6.3. The *contains_predicate* has the basic form:

contains_predicate ::= media_object **contains** salient_object |
document_object **contains** document_object | media_object

where, *media_object* represents an instance of a particular medium type, e.g., an image object or a video object, while salient_object is a *salient object* which is defined as an interesting physical object in some media object. Each media object has many salient objects, e.g. persons, houses, cars, etc. The **contains** predicate checks whether or not a salient object is in a particular media object. Predicate **contains** can also be applied to structured *document objects*. A document object represents an instance of a document element type. For example, typical document elements are articles, sections, figures, etc.

Query 8 Find all images in which a person appears.

select m
from Images m, Persons p
where m **contains** p

This simple query uses the **contains** predicate which checks whether a person p is in image m. In the following sections, these extensions are discussed and examples are given.

5.4 Spatial Primitives

5.4.1 Spatial Predicates

A spatial predicate compares the spatial properties of spatial objects and returns a boolean value as the result. First some image predicates are introduced. Then, some predicates for spatial objects are presented. They are useful in posting spatial queries.

5.4.1.1 Image Predicates

Let SO be the finite set of all salient objects in a database and $\mathcal{SO}$ be the power set of SO. Furthermore, let SS, CS, and TS be the sets of all spatial objects, color objects, and texture objects in the database respectively. A spatial object set describes the spatial properties of salient objects, such as shapes, locations etc., a color object set describes the color properties of salient objects and image backgrounds, and the texture object set describes the texture values of salient objects and image backgrounds. Then, an image is defined by a quadruple $< so,\ f,\ g,\ h >$ where $so \in \mathcal{SO}$ and mapping functions f, g, and h:

f: $so \longrightarrow SS$ maps a set of salient objects into a spatial set;
g: $so \longrightarrow CS$ maps a set of salient objects into a color set;
h: $so \longrightarrow TS$ maps a set of salient objects into a texture set.

Then, the following image operators are defined:

- *identical*: check if two images are identical; two images are identical if their salient object set so and functions f, g, h are equal respectively;
- *coincident*: check if two images have identical so and the same mapping function f;
- *subimage*: check if one image is contained inside another image, which requires that the image's so is subset of another image's so and their f, g, h are coincided. Two functions:
$$f_1 : so_1 \longrightarrow SS \qquad f_2 : so_2 \longrightarrow SS$$
are coincided if
$$f_1(x) = f_2(x) \qquad \forall x \in so_1 \cap so_2.$$
- *similar*: check if two images are similar with respect to some metric; such a metric can be based on salient objects, spatial relationships, colors, textures or combinations of these;
- *contains*: check if an image contains a particular salient object; i.e. check if a salient object is an element of so.

All the predicates are defined in terms of salient object set so, f, g, and h, except *similar*. Image similarity is an independent research area which has been very active [WJ96, SJ97]. Since image similarity search is out of the scope of this thesis, the query model is open to any existing proposed image similarity algorithm.

5.4.1.2 Spatial Predicate

Currently, MOQL supports three basic spatial domains: *point*, *segment*, and *region*. Although other domains, such as circle, rectangle etc., are provided, they are all special cases of *region*. A region may be represented by a set of points, a set of segments, a set of polygons, or other forms (e.g. a point and a radius) in some universe. In the case of continuous two-dimensional space, the universe is the whole plane. In the case of continuous three-dimensional space, the universe is the whole three-dimensional space. Regions in these cases correspond to areas and volumes respectively. The functions *nearest* and *farthest* are defined with respect to the set of points defined in a particular media object, such as an image or map. Table 5.1 shows basic spatial predicates defined in MOQL. Algorithms for computing these predicates are given in Appendix E.

The operands of the spatial predicates must be the same or compatible object types. For example, predicates *nearest* and *farthest* can apply only to two point objects, predicates *within* and

	point	segment	region
point	nearest, farthest	within, midpoint	centroid, inside
segment	cross	intersect	inside, cross
region	cover	cover, cross	topological_predicate, directional_predicate

Table 5.1: Spatial Predicates

midpoint can apply only to a point and a segment, and predicate *cover* may apply to a region and a point or to a region and a segment. The exact definitions of spatial predicates (or for the temporal predicates in the next subsection) are not given since they are self-explanatory. The directional relations include *left, right, above, below, front, back, south, north, west, east, northwest, northeast, southwest, southeast*, as well as the combinations of *front* and *back* with other directional relations. For example *front_left* or *front_northwest* may be possible. The topological predicates include *inside, covers, touch, overlap, disjoint, equal, coveredBy*, and *contains* which are specified in [EF91] as eight fundamental topological relations. Query 9 and Query 10 illustrate how the spatial predicates can be used.

Query 9 Select all the cities, from a map of Canada, which are within a 500km range of the longitude 60 and latitude 105 with populations in excess of 50000:

select c
from Maps m, m.cities c
where m.name="Canada" **and** c.location **inside circle**(**point**(60,105), 500) **and** c.population>50000

For each map, the method *cities* retrieves all the cities in this map. Then, a city's location is checked to see if it is within the required range. **point** is a constructor which accepts two values or three values to create a 2D point or 3D point respectively. Here, **point**(60, 105) represents a 2D point; **circle** is a circle object constructor which accepts a spatial point acting as the center of the circle and a radius; **inside** is one of the spatial predicates from Table 5.1.

Query 10 Find all the names of the objects inside a given region a in all images:

select o.name
from SalientObjects o, Images m
where m **contains** o **and** o.region **inside** a

5.4.2 Spatial Functions

A spatial function computes attributes of an object or a set of spatial objects. The spatial functions are shown in Table 5.2 and the algorithms to compute these functions are given in Appendix E. The return type refers to the type of objects returned by a spatial function. The *value* column contains some functions that return scalar values. Function *mbr* stands for minimum bounding rectangle. In addition to these, there is a universal function *distance*, which returns a scalar value when applied to any two spatial objects. Function *region* for both point and segment objects allows a point or segment to be converted to a region. Hence, all the predicates and functions for regions are applicable to points and segments. For example, directly checking a directional relation between a segment and a region is not allowed. However, after the conversion of a segment to a region, such a check can be made. Similarly, the following image functions are specified:

- *pan*: view different portions of an image
- *resize*: change the size of an image
- *superimpose*: synthesize two images into one

Query 11 illustrates the use of the spatial functions, while the use of the image functions is illustrated in Query 16.

Return Type	point	segment	region	value
point	nearest, farthest		region	
segment	cross	intersect	region	length, slope
region	centroid		interior, exterior, mbr	area, perimeter

Table 5.2: Spatial Functions

Query 11 Find the forests and their areas from the maritime region where each forest is covered by a single province.

select *forest*, **area**(*forest*.region)
from Forests *forest*
where *forest*.region **coveredBy any**
  select *p*.region
  from Provinces *p*
  where *p*.region **coveredBy** *maritimeRegion*

The above query illustrates the binding of two nested mappings combined with the spatial function **area** and spatial predicate **coveredBy**. The provincial region is passed from the interior level and used to direct the search in the exterior, to produce those forests in the maritime provinces which are completely covered by individual provinces.

5.5 Temporal Primitives

The inclusion of temporal data in a multimedia query language is an essential requirement. Research in temporal queries has focused more on historical (discrete) databases rather than on databases of temporal media (e.g., [Sno95]). Thus, the focus has been on the reflections of changes of the representation of real world objects in a database (e.g., President Clinton gave a speech at 2:00pm on July 4, 1997), rather than changes in continuous and dynamic media action. A complicated temporal SQL, called TSQL2 [Se94], has been proposed as a possible standard for historical databases. The interest here is in temporal relationships among salient objects in multimedia data, not the real world historical relationships which are the major concern of TSQL2. A typical temporal multimedia query is "*Find the last clip in which person A appears*". The specification of the temporal relationship *last* needs special support from query languages to process this query.

5.5.1 Temporal Functions

The choice of functional abstractions for temporal objects is influenced by the work of [GLÖS96]. Interval unary functions which return the *lower bound*, *upper bound* and *length* of the time interval are defined, while binary functions provide set-theoretic operations viz *union*, *intersection* and *difference*. A time interval can be *expanded* or *shrunk* by a specified time duration.

A wide range of operations can be performed on time instants. A time instant can be compared with another time instant with the transitive comparison operators "<" and ">". A time span can be *added* to or *subtracted* from a time instant to return another time instant. A time instant can be compared with a time interval to check if it falls *before*, *within* or *after* the time interval.

A *time span* is an unanchored relative duration of time. A time span is basically an atomic cardinal quantity, independent of any time instant or time interval. A time span can be compared with another time span using the transitive comparison operators "<" and ">". A time span can be *subtracted* from or *added* to another time span to return a third time span. Many functions have been defined in Table 3.1 in terms of behaviors. The following temporal granularities are considered: *year, month, day, hour, minute, second, ms* (millisecond).

5.5.2 Continuous Media Functions

For continuous media, only video data is considered. A *video* is modeled as a sequence of clips and a *clip* as a sequence of frames. A *frame*, the smallest unit of a video object, can be treated as an image. Each frame is associated with a timestamp or time instant while a clip or a video is associated with a time interval. This implies that frames, clips, and videos can be ordered. Therefore, the previous frame and the last frame can be obtained from a given clip or a given video. The continuous media functions are shown in Table 5.3. A universal function *timeStamp* applies to frames, clips, and videos and returns a time instant.

Since video data consists of sequences of images, they share all the attributes of image data such as color, shape, texture, and object. Unlike image, video has temporal relations. Such temporal relations introduce dynamicity (e.g. *motion*), which does not exist in image data. The implied motion in video data can be attributed to a camera (global) motion and an object (local) motion

Return Type	frame	clip	video
frame	prior, next	clip	
clip	firstFrame, lastFrame, *n*th	prior, next	video
video		firstClip, lastClip, nth	

Table 5.3: Continuous Media Functions

[ABL95]. In MOQL, an object motion is modeled by the multimedia database and then queried by using temporal predicates or functions. The definition of the abstract camera actions are based on [HK95]. A camera has six degrees of freedom, representing translation along each axis (x: track, y: boom, z: dolly) and rotation about each axis (x: tilt, y: pan, z: rotate). In addition, a change in the camera's focal length produces scaling or magnification of the image plane (zoom in and zoom out). To extract these features, each video stream should be first segmented into logical units by locating *cuts* (camera breaks). Cuts can be classified into different categories, such as fade, wipe, dissolve, etc. The following camera motion boolean functions are defined: *zoomIn, zoomOut, panLeft, panRight, tiltUp, tiltDown, cut, fade, wipe*, and *dissolve*.

In this section it is assumed that each continuous media object has a time interval associated with it that can be accessed through a method, *timestamp*, and that each salient object has a set of *timestamped physical representations*. The timestamped physical representation of a salient object indicates the physical characteristics of the salient object at different times. Typical physical characteristics of a salient object include geometric region, color, region approximation, etc. The set of physical representations of a salient object is accessible through method *prSet*. The following queries illustrate the temporal features of MOQL.

Query 12 Find the last clip in which person p appears in the video *myVideo*:

```
select    lastClip(select c from myVideo.clips c
                   where c contains p
                   order by upperBound(c.timestamp))
```

or

```
select    c
from      myVideo.clips c
where     c contains p and (upperBound(c.timestamp) >=  all
          select upperBound(d.timestamp)
          from myVideo.clips d
          where d contains p)
```

The first solution uses the features of the video function **lastClip** and OQL's **order by** clause. It is simpler than the second one. Assume that each video object has a method *clips* which returns a sequence of clips and each clip has a method *timestamp* which returns the time interval associated with this clip. Since two clips' intervals may overlap, simply relying on temporal predicates **after** or **meet** to perform the query is not possible. However, if it is sure that the upper bound of an interval is greater than or equal to all others, then its associated clip must be the last clip in a video. Therefore, a nested MOQL statement is used to express Query 12.

Query 13 List all the clip-pairs in which object o simultaneously appears:

select	c_1, c_2
from	Clips c_1, Clips c_2, o.prSet pr
where	c_1 **contains** o **and** c_2 **contains** o **and**
	pr.timestamp **during intersection**(c_1.timestamp, c_2.timestamp)

Note that because of video editing techniques such as fade and dissolve, clips can overlap in time. The tricky part of this query is in finding the overlap part of two neighboring clips. The temporal function **intersection** accomplishes this. Of course, object o must be within such an overlap and this constraint is satisfied by using the temporal predicate **during**.

Query 14 List clips where person p_1 is at the left of person p_2 and later the two exchange their positions:

select	c
from	Clips c, p_1.prSet pr_{11}, p_1.prSet pr_{12}, p_2.prSet pr_{21}, p_2.prSet pr_{22}
where	c **contains** p_1 **and** c **contains** p_2 **and** pr_{11} **left** pr_{21} **and**
	intersection(pr_{11}.timestamp, pr_{21}.timestamp) **during** pr_{11}.timestamp **and**
	pr_{12} **right** pr_{22} **and**
	intersection(pr_{12}.timestamp, pr_{22}.timestamp) **during** pr_{12}.timestamp **and**
	(pr_{11}.timestamp **before** pr_{12}.timestamp **or**
	pr_{11}.timestamp **meet** pr_{12}.timestamp)

Suppose clip c is the one looked for, then it contains both p_1 and p_2. In this case, both p_1 and p_2 must have at least two different physical representations respectively: one is p_1 at the left of p_2 and another one is p_1 at the right of p_2. pr_{11} and pr_{12} are used to represent the two states of p_1, and pr_{21} and pr_{22} are used to represent the two states of p_2. The spatial constraints bind p_1 to the left of p_2 and p_1 to the right of p_2 while the temporal constraints bind the intersection of pr_{11} and pr_{21} (as well as pr_{12} and pr_{22}) to be not empty. Such temporal constraints guarantee that p_1 and p_2 appear together sometime in this clip. Certainly, the timestamp of the relation p_{11} at the left of pr_{21} must be previous to the timestamp of the relation pr_{21} at the right of pr_{22}. Note this query may be simplified if the proposed moving object model is used as it is done in Query 5 and Figure 4.2.

5.5.3 Presentation Functions

The query language has to deal with the integration of all retrieved objects of different media types in a synchronized way. For example, consider displaying a sequence of video frames in which someone is speaking, and playing a sequence of speech samples in a news-on-demand video system. The final presentation makes sense only if the speaking person's lip movement is synchronized with the starting time and the playing speed of audio data. Because of the importance of delivering the output of query results, query presentation has become one of the most important functions in a multimedia query system. Both spatial and temporal information must be used to present query results for multimedia data. The spatial information tells a query system what the layout of the presentation is on physical output devices, and the temporal information tells a query system the sequence of the presentation along a time line (either absolute time or relative time). Query presentations are supported by adding a **present** clause as a direct extension to OQL:

select [**distinct**] projection_attributes
from query [[**as**] identifier] {, query [[**as**] identifier] }
[**where** query]
[**present** layout { **and** layout }]

where, *layout* consists of three components:

- spatial layout which specifies the spatial relationships of the presentation, such as the number of displaying windows, sizes and locations of the windows, etc.
- temporal layout which specifies the temporal relationships of the presentation, such as which media objects should start first, how long the presentation should last, etc.
- scenario layout which allows a user to specify both spatial and temporal layout using other presentation models or languages. This is necessary for handling prestored and complex presentation scenarios.

In order to support the spatio-temporal requirements of query presentations, the following presentation functions are defined:

atWindow(*identifier*, *point*, *point*): sets a spatial layout within a window defined by two points. *identifier* is an identifier of any media object and *point* is a spatial point object, e.g. point(10, 20).
atTime(*absoluteTime*): sets a real world time for supporting synchronization. *absoluteTime* specifies a date such as 1996/11/8/13:40:00;
display(*identifier*, *start_offset*, *duration*): presents a non-continuous media object. *identifier* is a graphical, image, or text object, *start_offset* is the time the display starts, and *duration* is the display duration (default **forever**).
play(*identifier*, *start_offset*, *duration*, *speed*): presents a continuous media object *identifier* is a frame, clip, video, or audio object; *start_offset* is the time the display starts; *duration* is the display duration (default: length of the identifier); and *speed* is the playing speed factor (1.0: normal, 0.5: half normal speed, 2.0: double speed).
thumbnail(identifier): presents the results as thumbnail objects. *identifier* is any allowable media type.
resize(*identifier*, *width*, *height*): resets the size of a media object. *identifier* is an identifier of a media object; *width* is the width of the result media object; and *height* is the height of the result media object.

parStart: two presentation events start simultaneously.

parEnd: as soon as one event ends, another event ends too.

after: an event starts right after another event finishes.

Note that most presentation functions are applied only to the first element of the result collection. Experience shows that sophisticated query presentation is difficult to specify in OQL-based (or SQL-based) query languages. One major reason is that the return result is a collection of objects and a user cannot select a particular object since there is no handle to reference it. Another major reason is that OQL-based query languages cannot support interactive multimedia presentations addressing quality of service. However, providing some basic functions within a query language for query presentation is still useful because in many cases, sophisticated presentation is not necessary. A typical example is searching information using the WWW. For more sophisticated cases, presentation and synchronization requirements need to be specified outside of the query specification. To this, MOQL provides a mechanism, *scenario layout*, to allow the invocation of such types of specifications. The following query examples show some features of the presentation functions.

Query 15 Find images, in which p appears, and display the result for 10 seconds with size (100, 100):

select m
from Images m
where m **contains** p
present **resize**(m, 100, 100) **and** **display**(m, 0, 10)

Function **resize** changes the size of the image to 100 pixels by 100 pixels. The offset of the **display** event is zero which means it starts immediately after the result is delivered. The **present** clause may be changed into **thumbnail**(m) which displays thumbnail images as references to the real images.

Query 16 Find all the image and video pairs such that the video contains all the cars in the image, show the image in a window at ((0, 0), (300, 400)) and the video in a window at ((301, 401), (500, 700)), and start the video 10 seconds after displaying the image; display the images for 20 seconds, but play the video for 30 minutes:

select m, v
from Images m, Videos v
where **for all** c **in** (**select** r **from** Cars r **where** m **contains** r)
v **contains** c
present **atWindow**(m, **point**(0, 0), **point**(300, 400)) **and**
atWindow(v, **point**(301, 401), **point**(500, 700)) **and**
play(v, 10, **normal**, 30*60) **parStart** **display**(m, 0, 20)

Operator **parStart** starts both video and image media objects simultaneously. Therefore, the image object is displayed immediately. However, since the start offset time for the video is 10 seconds, the video object will start 10 seconds after the image object starts. A default value (**normal**) is used for video playing. This can be changed for faster or slower playing by choosing a number either bigger than or less than one respectively.

Query 17 Find all clips in which both zoom-in and zoom-out exist and show their first frames in a thumbnail format:

select	**firstFrame**(c)
from	Clips c
where	**zoomIn**(c) **and zoomOut**(c)
present	**thumbnail**(**firstFrame**(c))

5.6 Structured Document Primitives

MOQL incorporates primitives for querying structured documents that conform to the SGML standard. Since understanding SGML document type definitions (DTDs) is very important to the rest of this section, a brief introduction to SGML is given first. The query primitives assume a particular SGML database schema [ÖEMI+97], which is also described.

5.6.1 SGML

The logical structure of a document is helpful for its contents to be fully understood. For example, document presentation, certain queries and hyperlinks all rely on the logical structure of the document. SGML is a meta-language which describes the logical structure of a document by using *markup* to mark the boundaries of its logical elements. The *generalized markup* approach of SGML separates the description of a structure from the processing of the structure. The philosophy is that processing instructions can be bound to the logical element at the time of formatting, or displaying. Descriptive (or generalized) markup identifies logical elements using *start tags* and *end tags* to mark their boundaries.

The markup in SGML is *rigorous* [Gol90] in that elements can contain other elements to form a hierarchy. Thus, `article` elements can contain `title` and `section` elements; `section` elements can contain `subsection` and `paragraph` elements and so on. The hierarchy is a tree, and entire subtrees can be manipulated as one unit. SGML does not specify what these elements should be, or what the hierarchy should look like. Instead, the list of elements types, and the relationships between them is expressed as a formal specification called a *Document Type Definition* (DTD). A DTD is written in SGML by the document designer for each category of documents.

A DTD (see Figure 5.3 for an example) consists of *element types*, *attributes*, and *entities*. An element type is defined using a declared value ELEMENT which specifies the hierarchical relationships between element types. An attribute defined with declared value ATTLIST contains information that is not part of the document content. An entity defined with declared value ENTITY must have the name of an entity defined in the DTD as its value. Such entity attributes are generally used to refer to an external entity containing non-SGML data.

```
(1)  <!DOCTYPE  article     [
(2)  <!ELEMENT  article     - - (title, edinfo, abstract, body?, section+)>
(3)  <!ATTLIST  article     id   ID   #IMPLIED>
(4)  <!ELEMENT  edinfo      - - (author+ & source & keywords & date)>
(5)  <!ELEMENT  section     - - ((title, body+) | (title, body?, subsection+))>
(6)  <!ATTLIST  section     id   ID   #IMPLIED>
(7)  <!ELEMENT  subsection  - - (title, body)>
(8)  <!ELEMENT  title       - - (#PCDATA)>
(9)  <!ELEMENT  author      - - (#PCDATA)>
(10) <!ELEMENT  source      - - (#PCDATA)>
(11) <!ELEMENT  keywords    - - (#PCDATA)>
(12) <!ELEMENT  date        - - (#PCDATA)>
(13) <!ELEMENT  abstract    - - (#PCDATA)>
(14) <!ELEMENT  body        - - (para | figure)+>
(15) <!ELEMENT  para        - - (link | #PCDATA)>
(16) <!ATTLIST  para        idref   IDREFS   #IMPLIED>
(17) <!ELEMENT  link        - - (quote | figure |#PCDATA)+>
(18) <!ATTLIST  link        linkend   CDATA   #REQUIRED>
(19) <!ELEMENT  figure      - - (figcaption?)>
(20) <!ELEMENT  figcaption  - - (#PCDATA)>
(21) <!ATTLIST  figure      filename   CDATA   #REQUIRED>
(22) <!ENTITY   myfigure    SYSTEM   "/usr/john/fig/john.gif">
(23) ]>
```

Figure 5.3: The DTD of `article`

An element structure is built using other elements or basic types such as #PCDATA, CDATA, EMPTY, etc. PCDATA stands for Parsed Character DATA (plain text) since the character that starts a tag ("<") might occur in such text. CDATA means that delimiters inside the element will be treated as literal data. EMPTY means that an element is a point event and cannot contain any text or other elements. Cross references are defined by ID and IDREF which permit the attachment of unique identifiers to elements so they can be referenced from other elements. There may be at most one attribute with declared value ID for a given element type.

Connectors can be further qualified with occurrence indicators. The comma connector (",") implies an order between elements. For example, an article is composed of a title followed by some `edinfo` and by an `abstract`, etc. (line (2) of Figure 5.3). An alternative connector ("&") can be used to ignore the ordering as in line (4) for `edinfo`. The bar connector ("|") provides a choice in the type definition. For example, a paragraph (`para`) can be either a `link` or #PCDATA (line (15) of Figure 5.3). An asterisk ("*") indicates zero or more occurrences, a question mark ("?") indicates zero or one occurrence, and a plus sign ("+") indicates one or more occurrences of an element. For example, an article may not have a body (line (2) of Figure 5.3) while its subsection must have one; a figure may have one or zero captions (line (19) of Figure 5.3); and a body may contain multiple figures (line (14) of Figure 5.3). Figure 5.4 shows an SGML instance of type `article` defined in Figure 5.3.

5.6.2 Modeling Document Structures

In this subsection, a technique to modeling SGML documents using the object-oriented technology is presented [ÖSEMV95]. Figure 5.5 shows the type hierarchy for logical document elements. The general characteristics of the design are:

- Each logical element type in the DTD is represented by a type in the type system[1]. For example, the `article` element is represented by the type `Article`. It can be argued that the textual element types have no semantic significance attached to them by either the DTD or the SGML standard. That is, there is no difference between the operations applied to a `title` element and those applied to a `para` element. There could be just one type, say `Element`, where an attribute value would indicate the type of the element. However, the markup in SGML is *generalized*. Elements are chosen by the DTD designer because each element has a different semantic significance that needs to be maintained. This can be done by ensuring as much static type checking as is possible. This is explained in the next item.

- To model the hierarchical structure, any type whose instance occurs in a non-leaf node in the hierarchy, has attribute(s) which are reference(s) to child instances. For example, the

[1]The only exception is the type `Pcdata`. Although not a logical element, it is considered to be a *pseudo-element*. For this reason, a type is declared for it, and instantiated whenever it occurs in an element with a mixed (logical elements mixed with plain character data) content model.

(1) <article> <title> Querying Structured Documents </title>
(2) <author> J. Z. Li </author>
(3) <author> M. T. Özso </author>
(4) <author> D. Szafron </author>
(5) <abstract> Querying structured documents has received a lot of research interest
(6) in last several years... </abstract>
(7) <section>
(8) <title> Introduction </title>
(9) <body> <para> An important trend in database research is the development of
applications ... </para> ... </body>
(10) </section>
(11) ...
(12) <section> <title> Modeling SGML Documents </title>
(13) <body> <para> In this section we give ... </para> </body>
(14) <subsection> <title> SGML </title>
(15) ...
(16) </subsection>
(17) </section>
(18) </article>

Figure 5.4: An SGML Instance of `article`

`subsection` has the content model:
<!ELEMENT subsection - - (title, body)>
Therefore, the type `Subsection` has attributes [2] that are references to instances of `Title` and `Body` types. This design ensures as much static type checking as possible. For example, an attempt to make an `Edinfo` instance as a child element of a `Subsection` instance would be disallowed at compile time. If there was just one type to represent all elements, then this checking would have to be done at run-time.

- Any SGML attribute defined in the DTD for a particular element is represented by an attribute in the representative type. The attributes themselves are always string-valued. Therefore, new types for the attributes are not defined. The types may have methods which access the values of these attributes.

In Figure 5.5 the supertype of all elements is the `Element` type. This models the *is-a* relationship — every SGML element instance is an `Element` instance. The model reflects the fact that all elements need to maintain a reference to their parent element in the document instance hierarchy. Therefore, the hierarchy can be navigated starting from any element. SGML documents are stored

[2] These attributes, which are references to instances of `Title` and `Body`, are different from ATTLIST attributes, which can be constants, variables, or references.

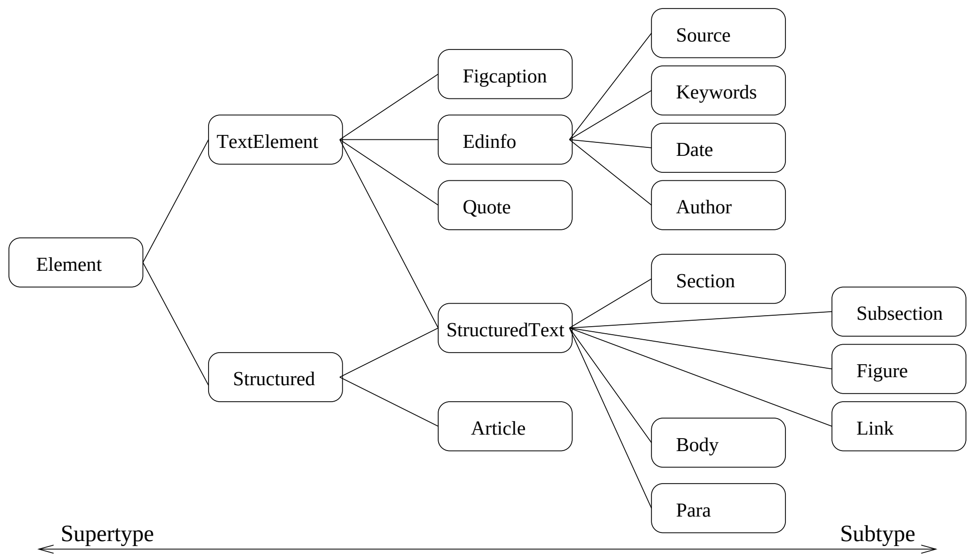


Figure 5.5: Type System for Elements

in the database using annotations [ÖSEMV95]. Elements defined for textual data in the DTD have corresponding types in the type system each with an attribute whose value is the annotation of the element in the article instance. For example, a subsection is an annotation that stores the start index and end index of its text in the whole text of the document. `TextElement` is their supertype which has methods to manipulate these annotation values. The types under `TextElement` correspond to text elements that do not have any subelements as shown in Figure 5.5. The type `Structured` is a supertype for elements in the DTD with complex content models. That is, structured elements can have child elements in the document instance and need to maintain references to them. Elements which are both structured and based on text have a common supertype called `StructuredText`. The subtypes of this type includes all text elements with complex content models, like `section`, `figure`, etc.

Instances of the `Article` type are at the root of the composition hierarchy. According to the DTD, they should have references to instances of `Title`, `Edinfo`, `Abstract`, etc. In addition, the `date`, `source`, `author` are attributes of `Article`, even though these values are already stored (by means of annotations as instances of `Date`, `Source`, `Author`).

5.6.3 Querying Structured Documents

A *document_expression* is defined as:

document_expression ::= *document_function_expression comparison_operator document_function_expression*

or

document_expression ::= *document_object document_predicate document_object.*

```
class Article : public Structured {
    String title;
    Edinfo edinfo;
    Abstract abstract;
    Body body;
    Set<Section> sections;
};
class Edinfo : public TextElement {
    Set<String> authors;
    String source;
    Set<String> keywords;
    Date date;
};
class Abstract {
    Set<Para> paras;
};
class Body : public StructuredText, public Para_Figure {
    Set<Para> paras;
    Set<Figure> figures;
};
class Section : public StructuredText {
    String title;
    Body body;
    Set<Subsection> subsections;
};
class Subsection : public StructuredText {
    String title;
    Body body;
};
class Para : public StructuredText {
    Set<String> content;
};
```

Figure 5.6: A C++ Expression of The Type System

A document function computes attribute values of an element object or a set of element objects and a document predicate compares the document properties of element objects and returns a boolean value as the result. Therefore, a document predicate can be viewed as a special case of a document function. The following document functions are added into MOQL:

- `title`(o): returns the title of an element object;
- `content`(o): returns the text body of an element object;
- `name`(o): returns the attribute name of an element object;
- `first`(o): returns the first element of element object o where o is a collection type object;
- `last`(o): returns the last element of element object o where o is a collection type object;
- `nth`(o, n): returns the nth element of element object o where o is a collection type object and n is an integer variable;
- `sf`(o): returns a set of structures which are substructures of element object o and *sf* stands for *Structure Function*;
- `sf`(o_1, o_2): returns a set of structures which are substructures of element object o_1 with the same type of element object o_2.

The following set of document predicates are also added into MOQL:

- `precede`: true if one structure precedes another structure in a document instance;
- `after`: true if one structure is after another structure in a document instance;
- `substructOf`: true if one structure is a substructure of another structure in a document instance;
- `contains`: true if one element object contains another element object.

All the above functions and predicates will be further explained with query examples. In particular, the structure function `sf` is designed to support querying without precise knowledge.

Query 18 Select all the section titles from *myArticle* in which "SGML" is in the body of the section:

select s.title
from s **in** *myArticle*.sections
where **content**(s) **contains** "SGML"

For each section s in *myArticle*, the `content` function retrieves s's body and the **contains** predicate checks if they contain the word "SGML".

Query 19 Find all the possible structure names under all the articles:

select `name`(b)
from Articles a
where b **substructOf** a

Function `substructOf` can be used to express this query easily. Possible substructures of `article` may include `edinfo`, `title`, `author`, etc. Function `name` is used to project a structure to its name. The query may be changed into "*Find all the articles with figure structures*":

select a
from Articles a, Figures f
where f **substructOf** a

It is no surprise that ordering is important in structured documents. For example, users may be interested in the first author of a document, the first sentence of the second paragraph, the third section of an article, etc. MOQL has introduced several functions and predicates to deal with ordering.

Query 20 Find the titles of all articles in which the first section contains "SGML" and the last section contains "HyTime":

select a.title
from Articles a
where (`title`(`first`(a.sections)) **contains** "SGML" **or**
`content`(`first`(a.sections)) **contains** "SGML") **and**
(`title`(`last`(a.sections)) **contains** "HyTime" **or**
`content`(`last`(a.sections)) **contains** "HyTime")

This query asks for any appearance of "SGML" and "HyTime" in the first section and last section respectively. Therefore, both the titles and bodies of the appropriate sections have to be checked. Functions `first` and `last` are used to select correct sections while functions `title` and `content` are used to select titles and bodies of the sections.

Suppose that the database contains letters with a preamble including the recipient (attribute *to*) and the sender (attribute *from*) addresses defined as:

<!ELEMENT preamble - - (to & from)>

which indicates that the ordering of *to* and *from* can be arbitrary. Consider the following query:

Query 21 Find letters where the sender precedes the recipient in the preamble:

select l
from Letters l
where l.preamble.from **precede** l.preamble.to

or

select l
from Letters l
where l.preamble.from **after** l.preamble.to

These queries show how to use structure predicates **precede** and **after**. Comparing the approach used in [CACS94] which uses *ordered tuples*,

select l
from Letters l, l.preamble[i].to, l.preamble[j].from
where i < j

the new approach is both simple and clean as the user does not have to know that element `preamble` must be an ordered element and no subscription is necessary to specify such an ordering.

Query 22 Find the names of all the substructures if *myArticle* is one article:

select `name`(s)
from `sf`(*myArticle*) s

This query illustrates the usage of the structure function `sf`. Note the difference between this query and Query 19.

Query 23 Find all the titles in *myArticle* with "SGML" in them as well as the lengths of their titles.

select t, t.length
from `sf`(*myArticle*, title) t
where t **contains** "SGML"

This query uses structure function `sf`(*myArticle*, title) which is different from *myArticle*.title. *myArticle*.title returns only the title of *myArticle* while `sf`(*myArticle*, title) returns the title of *myArticle*, all the section titles of *myArticle*, and all the subsection titles of *myArticle* if they exist. It is the system responsibility to look at the database schema to ensure the proper substructure instantiation from *myArticle*. Finally, the **select** projects the correct titles and their lengths.

Query 24 Find the titles of all articles with an image in which Bill Clinton is at the left of Jean Chretien:

select a.title
from Articles a, `sf`(a) b, Figures f, Images m, Persons p_1, Persons p_2
where (a **contains** f **or** b **contains** f) **and** f **contains** m **and** m **contains** p_1 **and** m **contains** p_2 **and** p_1.name = "Bill Clinton" **and** p_2.name = "Jean Chretien" **and** p_1 **left** p_2

This is a query with the feature of content-based querying which is reflected by a spatial predicate **left**. Since figures can only appear in bodies of an article as defined in Figure 5.3, the trick is to use the `sf` function to exhaust all the possible places where a figure can appear. These can include article's paragraphs, section's paragraphs, subsection's paragraphs, etc. The image must be referenced by the figure and this is accomplished by the condition f **contains** m. Furthermore, the image m must have Bill Clinton and Jean Chretien in it. This condition is checked by m **contains** p_1 **and** m **contains** p_2, assuming p_1 is Bill Clinton and p_2 is Jean Chretien.

5.7 The Expressiveness of MOQL

After defining MOQL, a natural question is what is the expressiveness of MOQL? The answer is found in the context of the TIGUKAT query model. The TIGUKAT query model defines a formal object calculus with a logical foundation that introduces a function symbol to incorporate the behavioral paradigm of the object model into the calculus. It also defines a behavioral/functional object algebra with a comprehensive set of object-preserving and object-creating operators. It has been proven that the object calculus is equivalent to the object algebra.

Although the TIGUKAT object calculus and algebra are designed for TQL (TIGUKAT Query Language), they can be used to investigate the expressiveness of MOQL. This is because the TIGUKAT object calculus and algebra include many powerful operators. It is proven that each TQL statement corresponds to a TIGUKAT object expression [Pet94]. Therefore, there is a complete reduction from TQL to the object calculus or algebra. From this result, a conclusion that the TIGUKAT object calculus or algebra is more expressive than TQL can be drawn. Consequently, it may be possible to use TIGUKAT object expressions to express MOQL statements although MOQL is more powerful than TQL in the sense that MOQL possesses more functionalities than TQL.

The formal semantics of MOQL has been studied in terms of the TIGUKAT object calculus. The major results are follows: the queries in the **select** and **from** clause can be eliminated from these clauses; any **select** statement in MOQL can be expressed in TIGUKAT object calculus; and the extended multimedia functions and predicates can be expressed by a bounded-depth domain independent TIGUKAT object calculus expression. The proofs of these claims are rather technical, and not essential for understanding the rest of the thesis. They are therefore deferred to the Appendix F.

Chapter 6

System Implementation

One objective of this research is the feasibility of the proposed model and language. This is accomplished within the context of DISIMA [OÖL+97]. The DISIMA research project deals specifically with the development of technology to facilitate the management of and access to images using a DBMS. Specifically, it aims at developing an image DBMS which will provide the users with uniform access to multiple, historically separated, and possibly heterogeneous image and spatial repositories. The DISIMA model aims at efficient representation of images to support a wide range of queries. Typical queries that DISIMA would support include: "*Find images that look like this sample image*"; "*Find images that contain a dog*"; and "*Find images which contains a man and a house where the man is to the left of the house*". Since the proposed model and query language have been adapted by the DISIMA project, the described implementation reflects part of the DISIMA system. On the other hand, DISIMA deals with other issues (such as a visual query language, distributed image DBMSs, meta type system, etc.) which are not part of this thesis.

As a first proof-of-concept prototype, the research has been implemented on top of ObjectStore. Figure 6.1 shows the system architecture and the shaded boxes are the topics of this thesis. Raw image and video data are processed by an *Image/Video Analyzing* module which uses image and video processing techniques via an *Image and Video Tool Library* to recognize *salient (physical) objects*, video shots, etc. The main function of the *Image and Video Analyzing* module is to extract features. Types of features to be extracted are application dependent. Restricted by the current technology, this analysis has to be domain specific. For video data, the salient objects are encoded in the *CVOT Generating* module by their properties, such as size, location, moving direction, etc. and a CVOT tree is generated. The *Image and Video Structuring* module builds the necessary indexes from the output of both the *Image and Video Analyzing* module and the *CVOT Generating* module to provide efficient access. A unified model is necessary for users to query the system and an *Object-oriented Processing* module is used for this purpose, because of its powerful representation of the users' views and its suitability for multimedia data. A user posts queries through an MOQL or an Iconic Graphical User Interface (ICON). MOQL allows users to elaborate their queries precisely using a textual language. On the other hand, ICON, like other Graphical User Interfaces (GUI), allows users to sketch iconic objects and to ask the database to return the target images or videos with the closest match according to the objects attributes. All the queries are handled by the *Query Processing* module internally. The *Query Processing* module defines translation of the queries into an internal query algebra which can be optimized and executed by the system. During this translation some query optimization can be done; therefore the *Query Processing* module needs to account for the specifics of image and video structuring. While executing user queries, the system may not have all the essential information, but it can make temporal or spatial inferences from the existing relations using the reasoning rules provided by the *Knowledge Base*. The answer to the queries is returned to the end user through a graphical interface.

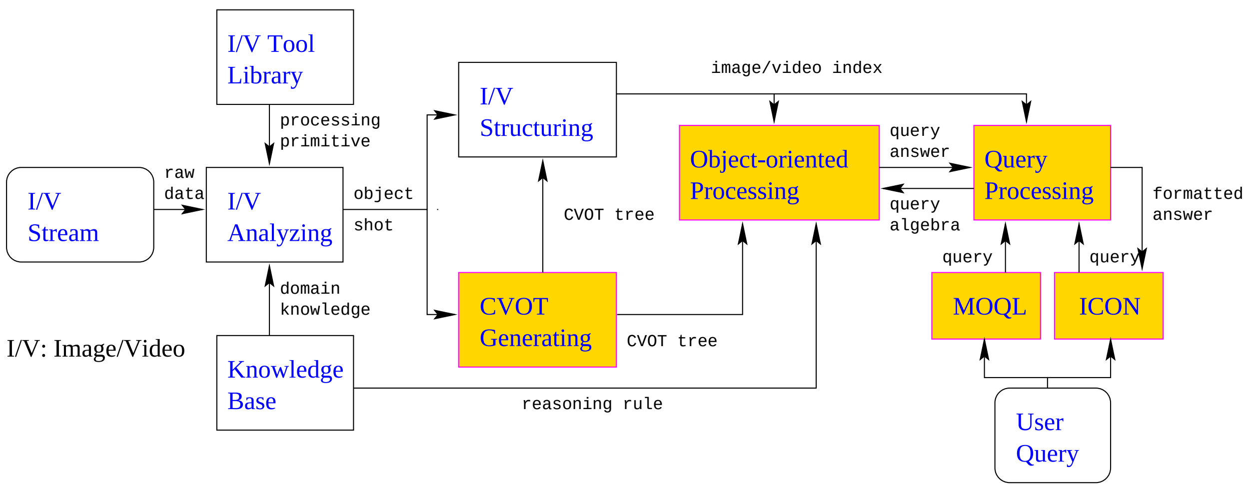


Figure 6.1: System Architecture

As indicated in previous chapters, images or video frames contain *salient objects*. A salient object may be in one or more images or video frames while each image or video frame can have many salient objects. In order to efficiently access salient objects two types of salient objects are distinguished: *logical salient objects* and *physical salient objects*. *Logical salient objects* correspond to objects independent of their appearance in a media and describe their attributes independent of any media. *Physical salient objects* correspond to objects specific to a media and describe their attributes characterized by the underlying media. For example, suppose *John* is a person who appears in image m. Then, the name and address of *John* are considered logical properties of the salient object *John*. The spatial attributes, the color attributes, and the texture attributes of *John* in image m are considered the physical properties of the salient object *John*. Since *John* might appear in many different images, the relationship between logical salient objects and physical salient objects is one-to-many.

In order to further explore the commonalities of different media, e.g., images, videos, graphics,

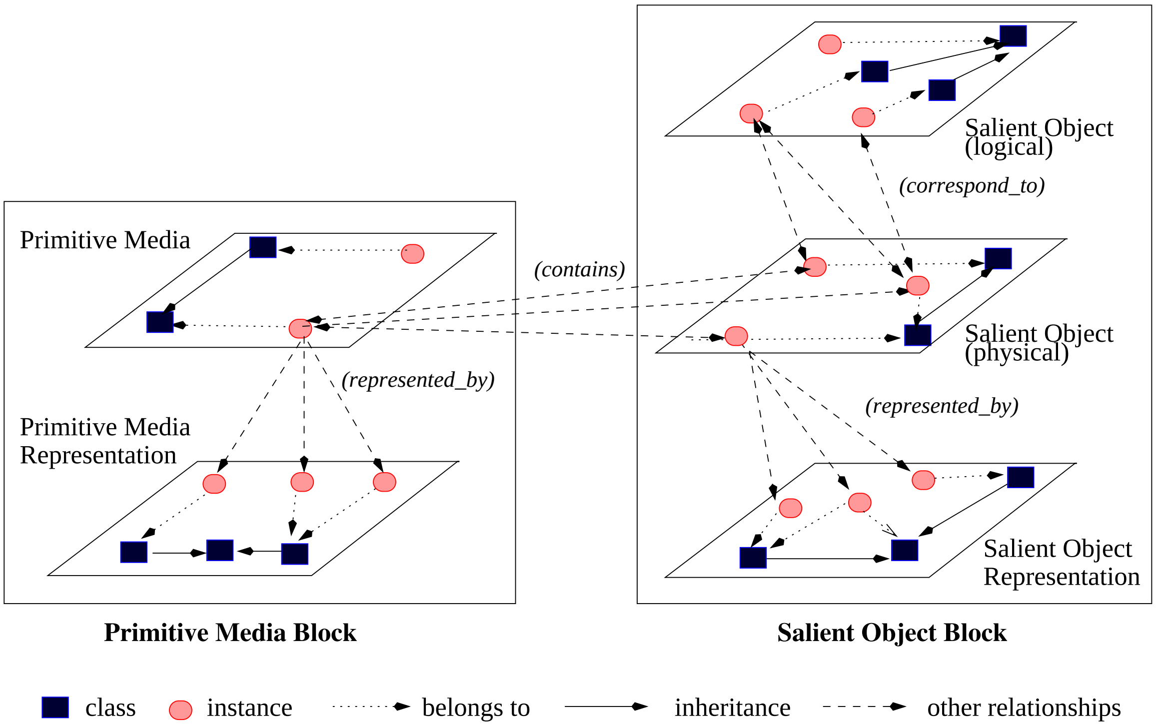


Figure 6.2: Image and Salient Object Model

etc., *Primitive Media* is introduced as an abstract class[1] to class *Image* and *Video* (see Figure 3.8). From Figure 6.2 it is clear that the model is composed of two main blocks: the Primitive Media Block and the Salient Object Block. A block is defined as a functional first-level entity that can be broken down into several entities. The *primitive media block* is made up of two layers: the *primitive media* layer and the *primitive media representation* layer. A primitive media is distinguished from its representations to maintain an independence between them, referred to as *representation independence*. In the *primitive media* layer, the user defines primitive media type classification which allows the user to define functional relationships between primitive media.

The content of a media is viewed as a set of salient objects (both logical and physical) with certain temporal and spatial relationships to each other. The *salient object block* is designed to model salient objects. Multiple physical salient objects belonging to images may correspond to one logical salient object. Hence, a logical salient object does not have any representation, but a physical salient object may have one or more. The logical salient object level models the semantics of physical salient objects. For a given application, the set of salient objects is known and can be defined. The definition of salient objects can lead to a physical salient object which may have one or more representations (an MBR, a point-set, a vector etc.). Salient object relationships such as spatial relationships (e.g., *left, northeast*) make sense only at the physical salient object level. A

[1] In this chapter type and class are used interchangeably.

logical salient object has some functional relationships (is-a) with other logical salient objects. A logical salient object gives a meaning (a semantics) to a physical salient object.

Depending on the application, the user can assign different semantics to the same physical salient object. Detailed relationships among images, logical salient objects and physical salient objects are given in Figure 6.3. The two levels of salient objects ensure the semantic independence and multi-representation of salient objects. For example, only the information at logical salient object level is necessary to answer the query "*find all the images in which* John *appears*" whereas the query "*find all the images in which* John *is to the left of* Paul" requires information from the physical salient object level to check the spatial relationship among the two salient objects. The processing can indeed benefit from the existence of spatial indexes.

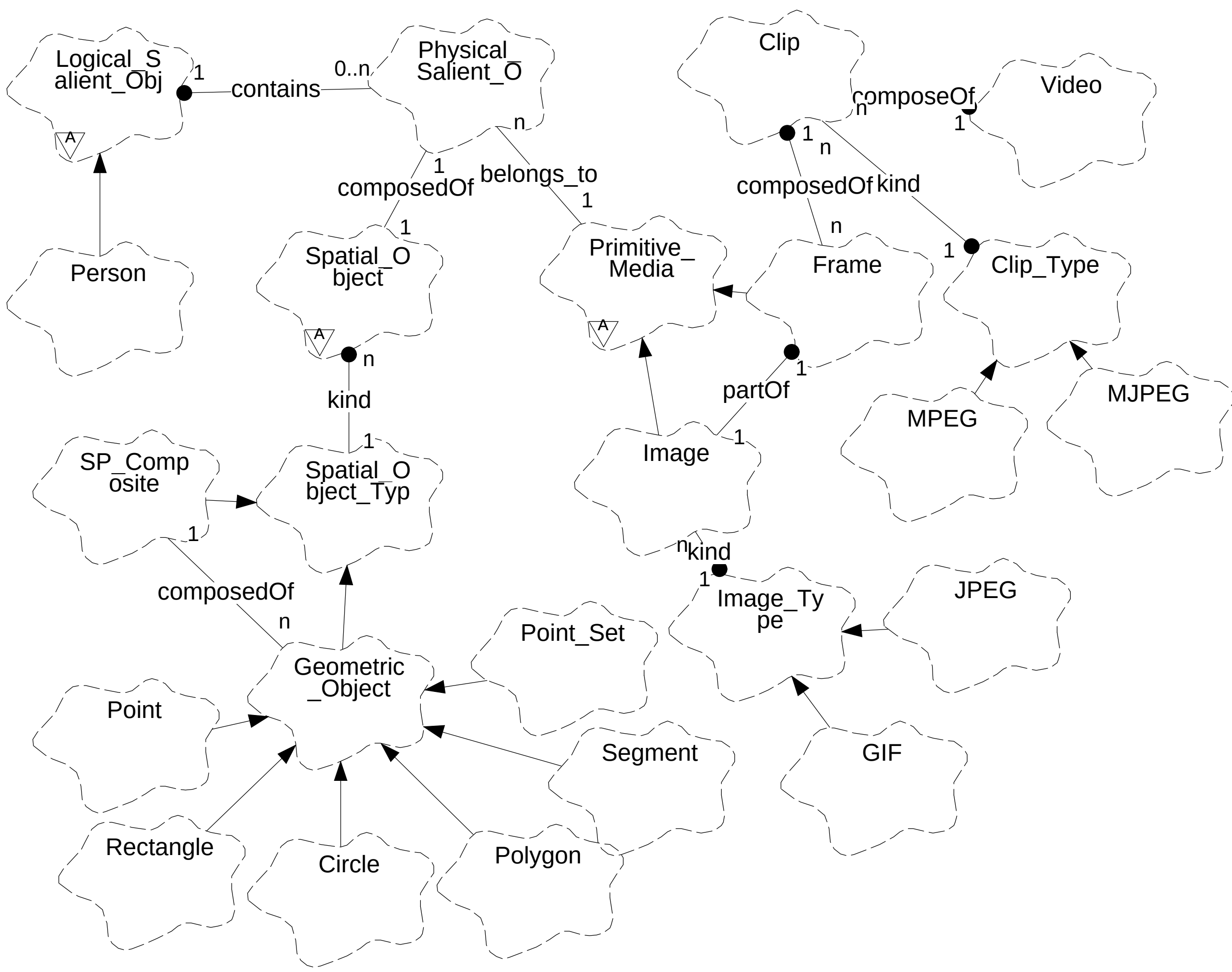


Figure 6.3: The Core Class System

Figure 6.3 shows the core classes which are implemented in the prototype. This figure is generated directly from Rational Rose [WG94] which is a design tool for object-oriented development.

A line that ends with an arrow represents a *subclass* relation while a line that ends with a bullet represents an *aggregation* relation. Other types of relations are marked explicitly, such as *contains*, *composedOf*, etc. The numbers, such as 0, 1 and n, are the cardinalities of participating classes. *abstract classes* are marked by a small triangle with character "A" inside.

Two abstract classes deserve more explanation. One is *Primitive_Media* and the other is *Spatial_Object*. *Primitive_Media* is the super class of all media, e.g. still images, videos. Only *Primitive_Media* interacts with *Physical_Salient_Object* and their relationship is one-to-one. This allows the addition of any new medium into the system without changing the type system. For example, if it is planned to model animation as a new type of media, then a new subtype animation is added under the *Primitive_Media* without touching any existing class.

Spatial_Object is the super class of all geometrical objects which include *Point*, *Line*, *Rectangle*, *Polygon*, *Point_set*, etc. A *Spatial_Object* may consist of one geometric object or multiple geometric objects. A special class *SP_Composite* is designed for the later. This allows the system to model complex objects by simple geometric objects. For example, the spatial object of a person might consist of a head (a circle), a body (a rectangle), and two arms (polygons). Some generic temporal primitive classes are also defined, such as *Interval*, *Instant*, *Video_Instant*, *Time*, and *Date*. Class *Video_Instant* is for modeling individual frames in videos. This group of classes is designed for the videos and sequences of images and they are not shown in Figure 6.3 because they are not crucial in understanding the type system.

Figure 6.4 reveals more details of some method definitions in C++ for several core classes in the system. The classes *Relationship_1_m* (one-to-many), *Relationship_m_1* (many-to-one), and *Set* are provided by ObjectStore. The definitions are self-explanatory.

By using the functional approach described in Section 2.4 over the reduced images, it is possible to store the precise geometrical shapes and locations of salient objects with little storage space. The tradeoff is a slight increase in computation time. Then, the MBRs of salient objects can be easily computed from this precise information. A unique feature of this system is its support for both the precise and the MBR-based spatial relation computations in MDBMSs.

The system uses a two-level approach to process spatial queries over image data. At first level (or filter level), all the spatial regions are approximated by their minimum bounding rectangles (MBRs). The computation of topological relations between MBRs are simple and efficient. The possible candidates generated from this level are passed to the next level which is the point-set model. This level deals with significantly fewer spatial regions than the first level.

An MOQL interpreter parses MOQL queries and generates an algebraic tree. The procedure of query processing is depicted in Figure 6.5 where the components that have been completed are shaded. Following the syntactic check, a semantic check is performed on the query to validate types, classes and class extents. At this point, it is possible to perform some semantic query optimization [Kim82] although the current research has not yet fully addressed this issue. The correct MOQL

```
class Logical_SO {
   Relationship_m_1(Logical_SO, physical_so, Physical_SO,
         logical_so, Set<Physical_SO*>) physical_so;
   String annote, type;
public:
   String& get_annotation() const;
   Set<Physical_SO*>& physical_objects();
   void add_physical_object(Physical_SO* pso);
   void remove_physical_object(Physical_SO* pso);
   virtual String& get_type() const = 0;
};

class Physical_SO {
   Relationship_1_m(Physical_SO, pm, Primitive_Media,
         physical_so, Primitive_Media*) pm;
   Relationship_1_m(Physical_SO, logical_so, Logical_SO,
         physical_so, Logical_SO*) logical_so;
   Spatial_Object *spatial_object;
public:
   Primitive_Media* primitive_media() const;
   Logical_SO* logical_object() const;
   Spatial_Object* spatial_object() const;
   void set_spatial_object(Spatial_Object* so);
   void set_logical_object(Logical_SO* lso);
   void remove_logical_object(const Logical_SO* lso);
   void set_primitive_media(Primitive_Media* pm);
   void remove_primitive_media(const Primitive_Media* pm);
   Rectangle* get_mbr();
class Primitive_Media {
};

   Relationship_m_1(Primitive_Media, physical_so,
         Physical_SO, pm, Set<Physical_SO*>) physical_so;
public:
   Set<Physical_SO*>& physical_objects();
   void add_physical_object(Physical_SO* pso);
   void remove_physical_object(Physical_SO* pso);
};
```

Figure 6.4: Three Key Classes Definition in C++

queries are then translated into an object algebra [ÖPS+95] that is used as the basis of optimization. This algebra is sufficiently powerful to support OQL queries.

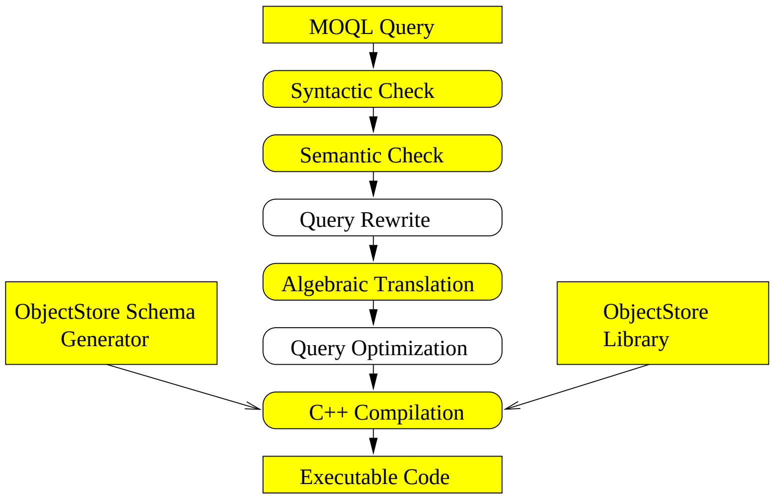


Figure 6.5: MOQL Query Processing

One of the research objectives is to study algebraic primitives to support optimization of multimedia queries. The target algebra does not yet include these primitives and therefore, extensions are necessary. Furthermore, query optimization is a topic of future research. In the current prototype each algebraic operator is implemented in terms of ObjectStore functions. This establishes a link between the query processor and the underlying object repository while enabling independent development of the query processor.

The whole system consists of C++ code (the core system), Lex/Yacc code (parsing MOQL), Java code (the graphical user interface), and C code (generating MOQL query tree and setting the web communication between Java user interface and the core system). Figure 6.6 shows the on-line architecture of the system. A *client* in Figure 6.6 can be a Netscape or MS Internet Explorer client which is able to display HTML documents. The interface of this client is written in Java. The CGI (Common Gateway Interface) script simply passes whatever passed from the database server (DB Server) to a Web Server. The DB Server is responsible for accessing the image database (an ObjectStore database) and translating query results into HTML documents. Currently the system is functional with a small image database and can be accessed through web browsers (Netscape or MS Internet Explorer) [2]. More images from the Photography Service, University of Alberta, have been digitized and will be added into the database soon.

As shown in Figure 6.7, the graphic user interface is divided into two major parts: the left part is used to display useful information (such as background about the system, help page) and results while the right part allows users to post queries. The right part is further divided into two

[2]The WWW address is http://www.cs.ualberta.ca/~database/disima.html/.

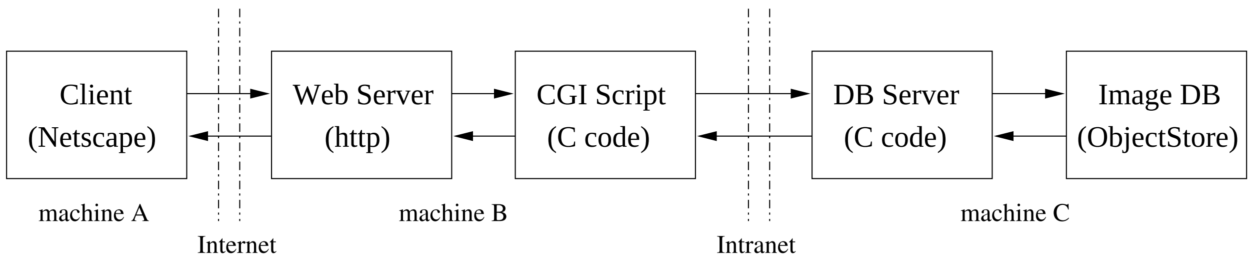


Figure 6.6: The On-line Architecture

areas: the top area is called **IconSpace** and the bottom area is called **MoqlSpace**. **IconSpace** is designed to help a user retrieve images and videos by spatial similarity while **MoqlSpace** is designed to help a user retrieve images and videos by spatial relationships.

In **IconSpace** there is a sketch pad allowing a user to draw any spatial relationships between salient objects. Each object can be either associated with an icon (such as *Sun, Sea, Person*) or a particular salient object (such as *John, my blue-balloon, Paul's bike*). Users can set the *similarity threshold* value (from 1% to 99%) to decrease or increase the qualified results. Lowering this value can result in more returns with less similarity. Detailed discussion of spatial similarity is given in Section 2.5. Queries can be posted on images, videos, or both. Figure 6.8 shows a query image with three icon objects (Sun, Sea, and Tree) spread out like a triangle.

The right bottom of Figure 6.8 shows the query "*find all the images in which* John *is to the left of* Paul or *there is a tree*". Such a query will be translated into the following MOQL query:

select m
from Images m, Persons p, q, Tree t
where m **contains** t **or** m **contains** p **and** m **contains** q **and**
p **left** q **and** p.name = "John" **and** q.name = "Paul"

In **MoqlSpace**, if the spatial relation is *NONE*, the system will automatically interpret the query as predicate **contains**. For example, the query "*find all the images having a person*" can be simply expressed by selecting a *Person* item and then click on *Query* button. Again users can post the queries over images, videos or both. In **MoqlSpace**, either MBRs or precise spatial properties can be specified to compute user queries. In any case, help information is always provided.

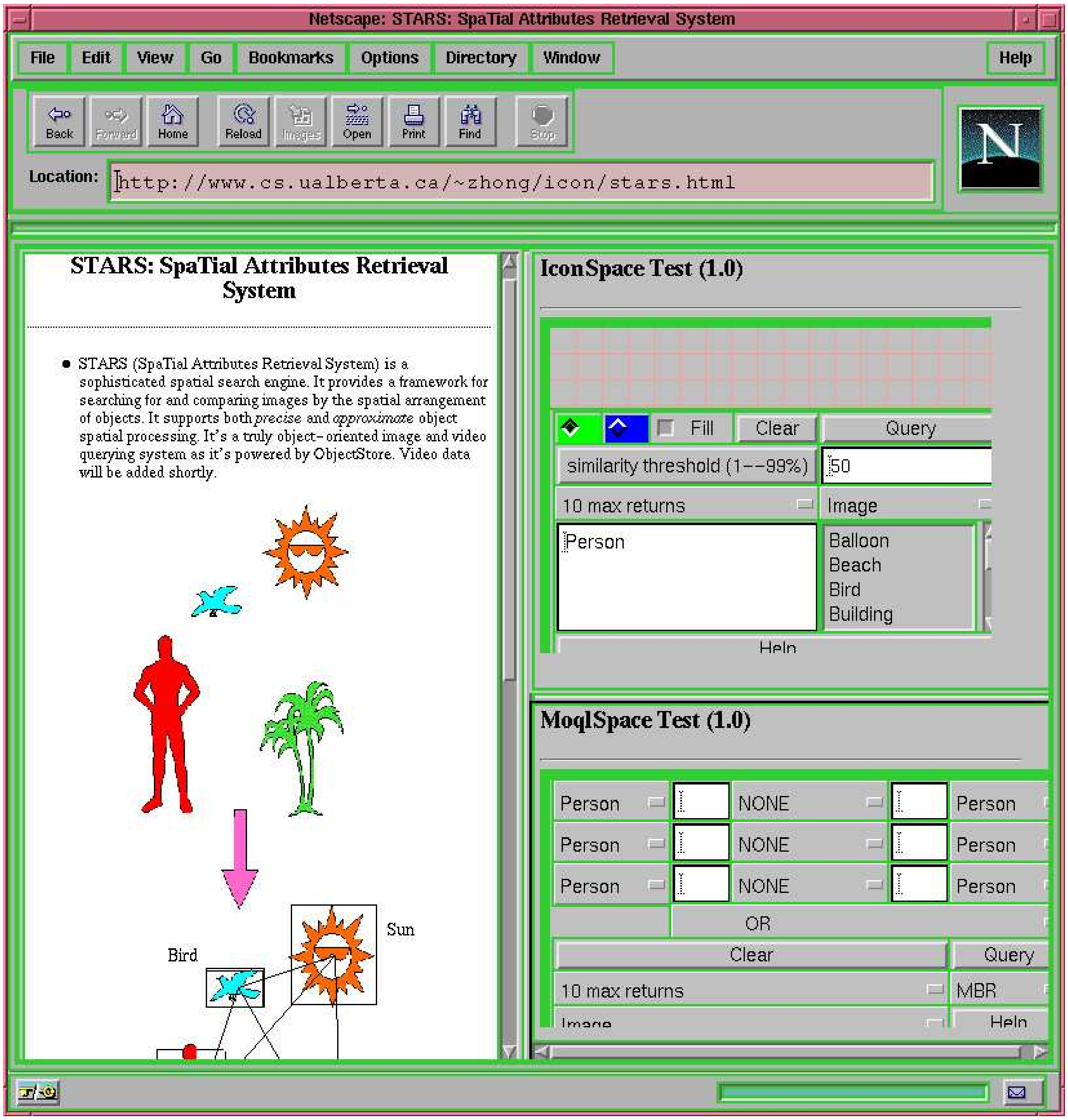


Figure 6.7: The Graphical User Interface

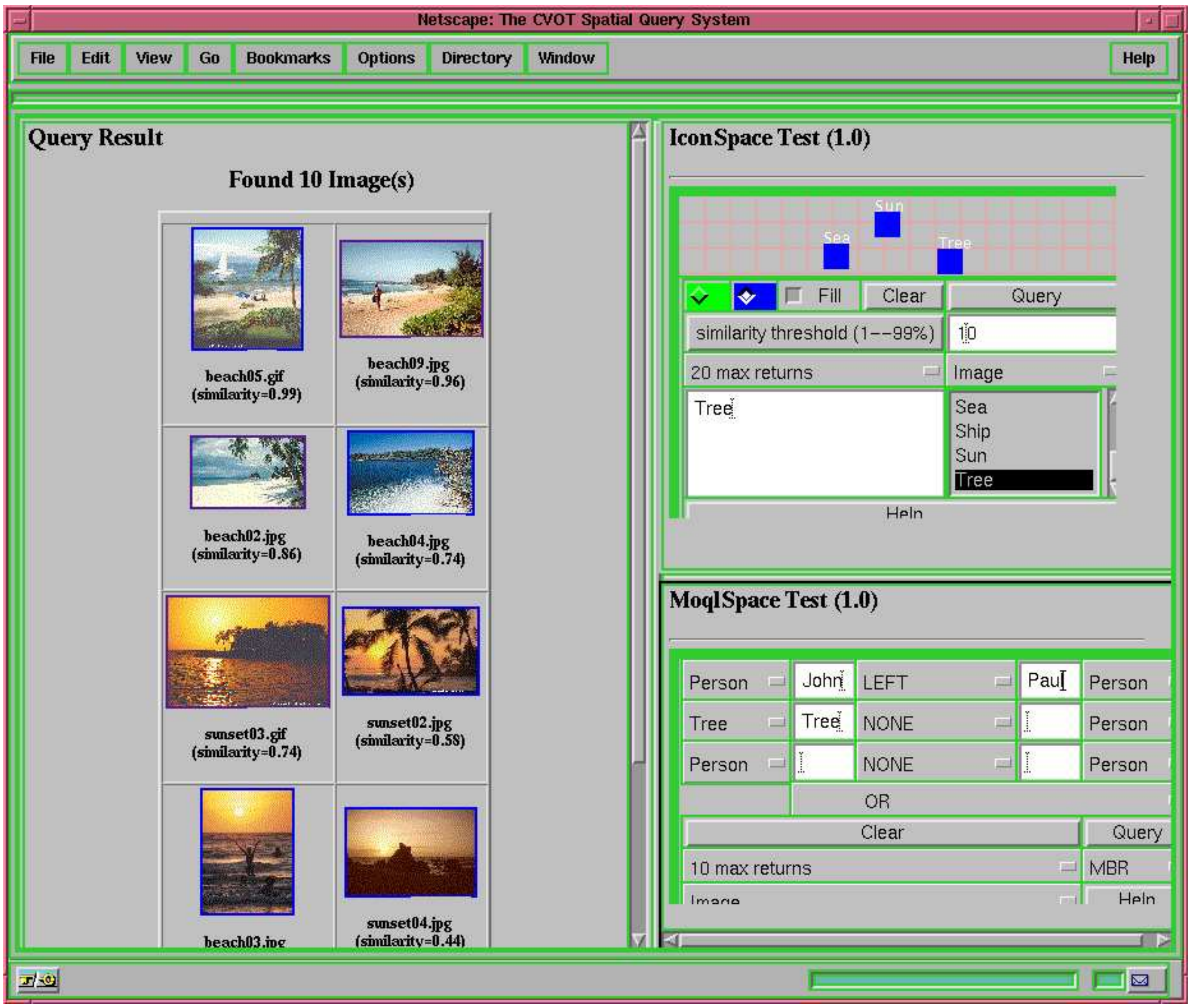


Figure 6.8: An IconSpace Query

Chapter 7

Conclusions

7.1 Summary and Contributions

The main topic of this thesis is modeling and querying multimedia data. An object-oriented approach is used to search for possible solutions. The proposed model is built around the *salient object* concept. One novelty of this thesis is to further distinguish *logical* and *physical* salient objects allowing the underlying system to clearly separate media-dependent information from media-independent information. Therefore, the implementation of the proposed model and query language becomes much easier.

The first step is to model spatial attributes of multimedia data and a two-level representation approach is presented. At the lower level, the point-set technique is used in defining *topological relations*. At the higher level, the spatial approximation of salient objects is represented by their MBRs. User queries can be evaluated by a point-set technique, by MBR approximation, or a combination of both. However, to achieve fast query processing a filter step is always used before a point-set technique is applied. The following new features of such a spatial model have been identified [LÖS96a, LÖS96c, LÖS96d, LÖS96e, LÖ98]:

1. A completely new way of computing point-set topological spatial relations is presented, which results in significant storage reduction. The trick is to use a functional approach to represent the interior of a spatial object and the tradeoff is longer computation time.

2. Comprehensive experiments have been conducted and the results show that the new functional approach is both feasible and efficient.

3. A qualitative spatial representation of salient objects is introduced using the temporal interval algebra. Such a representation defines a complete set of both topological and directional relations.

4. The model supports comprehensive spatial queries and temporal queries, as well as spatio-temporal queries. The queries can be quantitative, qualitative, or a combination of both.

5. A rich set of spatial inference rules based on the new spatial model has been identified and their correctness is proven.

6. A conceptual integration of the spatial model into an ODBMS is presented. TIGUKAT, the in-house ODBMS, was chosen for its powerful modeling capabilities.

On issues of modeling the temporal attributes of video data, the focus is on finding an efficient way to support queries over the content, such as salient objects. The major contributions of modeling temporality are as follows [LGÖS96, LGÖS97]:

1. A formalization of an abstract multimedia model (CVOT) is proposed. This CVOT model is more flexible in organizing video data than other models because it can handle both perfectly ordered clip sets and weakly ordered clip sets.

2. A uniform approach to model frames, clips, videos, and salient objects using a type *history* is introduced. This approach allows native support for a rich set of temporal multimedia operations.

3. A greedy algorithm, which can automatically construct a CVOT for a given video is described. The time complexity of the algorithm is analyzed.

4. A comprehensive video type system has been developed over TIGUKAT. Such a type system supports a broad range of user queries.

The most striking difference between still image and video comes from movements and variations. Much research has been focusing on how to retrieve moving objects as part of content-based querying. The rich set of spatial and temporal operators in the spatial and CVOT models provides an effective way of modeling moving objects. The proposed model captures not only the trajectory of single moving object, but also the relative spatial relations between objects. Several algorithms are designed to efficiently retrieve moving objects based on exact matching and similarity matching. In general, similarity matching is more useful than exact matching in MDBMSs. The novelties of the proposed model for moving objects are the following [LÖS96b, LÖS97a]:

1. A unique way of qualitatively representing the trajectory of a moving object.

2. The first research work proposing a way of modeling the relative spatio-temporal relations between moving objects.

3. New algorithms designed for exact matching of trajectories and spatio-temporal relations.

4. New algorithms designed for similarity matching of trajectories and spatio-temporal relations.

5. The final model supporting a broad range of queries about moving objects.

The definition of a formal multimedia model provides a good platform from which a systematic and consistent investigation of query languages is conducted. An object-oriented, general-purpose multimedia query language (MOQL) is introduced. MOQL contains many multimedia object constructs and is easy to use if users are familiar with OQL. The important contributions of MOQL are summarized as [LÖS95, LÖS97b, LÖSO97, LÖS98]:

1. It is a complete multimedia query language that supports general media and applications.
2. It has a rich set of spatial and temporal operators that support content-based queries.
3. It possesses the ability to support query presentations, which is an increasingly important feature for Internet applications.
4. It has support for querying structured documents. In particular a unique approach is invented to support querying without precise structure knowledge through a function constructor.
5. The formal semantics and expressiveness of MOQL have been investigated under the context of TIGUKAT object calculus and algebra. The reduction from MOQL statements to TIGUKAT object calculus is illustrated step by step.

Based on the proposed model and language, a proof-of-concept prototype has been implemented. The core system is built on top of ObjectStore using C++. An MOQL preprocessor which parses MOQL queries and generates parsing trees for subsequent processing (e.g., query optimization) is written in Lex/Yacc and C while the graphical user interface is implemented in Java. Currently the system is functional with a small image database and can be accessed through web browsers. Major contributions of this prototype are as follows [LÖ97, OÖL+97]:

1. A unique architecture in organizing multimedia data is coupled with a new type system. This makes the system extensible.
2. A clear separation of *logical salient objects* and *physical salient objects* enables the system to be simple and efficient.
3. An MOQL parser allows both syntax and semantic checking over MOQL queries. It further generates algebraic parsing trees for subsequent processing.
4. A two-step spatial processing is implemented which greatly enhances spatial query evaluation.
5. While both quantitative and qualitative spatial queries are supported, spatial queries based on similarity are also supported.

7.2 Future Research

The work presented in this thesis suggests some interesting directions for future research. On the modeling side, another popular media, *audio*, is left-out. In many current systems, audio is treated as an opaque collection of bytes with only the most primitive fields attached: name, file format, sampling rate, and so on. How might people want to access sounds through content-based retrieval? The following methods are considered useful [WBKW96]:

- *Simile*: saying one sound is like another sound or a group of sounds in terms of some characteristics. For example, "like the sound of a herd of cows".
- *Acoustical/perceptual features*: describing the sounds in terms of commonly understood physical characteristics such as brightness, pitch, and loudness.
- *Subjective features*: describing the sounds using personal descriptive language. For example, a user might be looking for a "shimmering" sound.
- *Onomatopoeia*: making a sound similar in some quality to the sound you are looking for. For example, the user could making a buzzing sound to find bees or electric hum.

In a retrieval application, all of the above could be used in combination with traditional keyword and text queries. However, the difficulty comes from what sound features should be extracted from audio data. Are those extracted features good enough to represent the audio data? Furthermore, how can the system be trained to recognize those features so queries can be answered is probably even tougher? Therefore, an analytical model must be worked out before any modeling work can proceed.

Query optimization for MDBMSs is another important issue. This requires making full use of traditional indexing techniques as well as new multidimensional indexing techniques [Fal96] and spatial indexing techniques such as 2D string [CJT96]. Since rules have been a very popular tool in implementing query optimization, research can focus on finding specific rules for a multimedia environment. Limited work [Gay97] has been done in this area. Thus, further investigation into the special treatment of multimedia queries is necessary.

Graphical user interfaces have played an important role in MDBMSs as an effective communication tool between systems and users. The currently implemented graphical user interface is not sufficient because it allows only simple queries. There is no way for users to populate and update the database. A more sophisticated graphical user interface is absolutely necessary. Such a graphical user interface allows a broader range of queries (such as full content-based queries, similarity queries, incremental queries, etc.). Research on these issues, as well as database population and database updates, is under way within the DISIMA project.

On the implementation front, the work on MOQL has been kept minimal. All the aggregating functions, such as **group by**, **order by**, **max**, etc. are not handled by the current system. If

TIGUKAT object algebra is used as the implementation of MOQL, all the operators of TIGUKAT object algebra must be implemented. This alone can be a big project. Since videos have been studied extensively in this thesis, the system should be able to handle video data. If the approach of using representative frames for clips is adopted, the addition of the videos to the system is straightforward. This is because each representative frame can be viewed as a still image and some fundamental temporal operators have been implemented in the system for handling spatial relationships by extension. Any useful multimedia database should have a reasonable amount of data. The current data size in this implementation is very limited. Increasing the size of the database is also necessary since only a sizable multimedia database can test the efficiency of all the new features as an integrated system.

Bibliography

[AB91] R. Ahad and A. Basu. ESQL: A query language for the relational model supporting image domains. In *Proceedings of the 7th International Conference on Data Engineering*, pages 550—559, Kobe, Japan, 1991.

[AB95] S. Abiteboul and C. Beeri. On the power of languages for the manipulation of complex values. *The VLDB Journal*, 4(4):727—794, 1995.

[ABL95] G. Ahanger, D. Benson, and T. D. C. Little. Video query formulation. In *Proceedings of Storage and Retrieval for Images and Video Databases II, IS&T/SPIE Symposium on Electronic Imaging Science and Technology*, pages 280—291, San Jose, CA, February 1995.

[ACC+96] S. Adali, K. S. Candan, S. S. Chen, K. Erol, and V. S. Subrahmanian. The advanced video information system: Data structures and query processing. *ACM-Springer Multimedia Systems Journal*, 4(August):172—186, 1996.

[ACC+97] S. Abiteboul, S. Cluet, V. Christophides, T. Milo, G. Moerkotte, and J. Simeon. Querying documents in object databases. *Journal of Digital Libraries*, 1(1):5—19, 1997.

[AEG94] A. I. Abdelmoty and B. A. El-Geresy. An intersection-based formalism for representing orientation relations in a geographic database. In *Proceedings of the 2nd ACM Conference on Advances in GIS Theory*, pages 134—143, Gaithersburg, MD, 1994.

[All83] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of ACM*, 26(11):832—843, 1983.

[ANSI86] American National Standards Institution. The database language SQL. ANSI X 3.135, 1986.

[ANSI92] American National Standards Institution. The SQL-92 document. ANSI X3H2, 1992.

[ATS96] H. Arisawa, T. Tomii, and K. Salev. Design of multimedia database and a query language for video image data. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 462—467, Hiroshima, Japan, June 1996.

[BA96] K. Böhm and K. Aberer. HyperStorM – administering structured documents using object-orie nted database technology. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 547—547, Montreal, Canada, June 1996.

[BAK97] K. Böhm, K. Aberer, and W. Klas. Building a hybrid database application for structured documents. *Multimedia Tools and Applications*, pages 275—300, 1997.

[BH94] H. Buxton and R. Howarth. Behavioural descriptions from image sequences. In *Proceedings of Workshop on Integration of Natural and Vision Processing Language*, pages 231—239, August 1994.

[BKSS90] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger. The R^*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 322—331, 1990.

[BKSS94] T. Brinkhoff, H. P. Kriegel, R. Schneider, and B. Seeger. Multi-step processing of spatial joins. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 197—207, Minneapolis, Minnesota, May 1994.

[BRG88] E. Bertino, F. Rabitti, and S. Gibbs. Query processing in a multimedia document system. *ACM Transactions on Office Information Systems*, 6(1):1—41, January 1988.

[CACS94] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 313—324, Minneapolis, Minnesota, May 1994.

[Cat94] R. Cattell, editor. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, San Francisco, CA, 1994.

[CIT+93] A. F. Cardenas, I. T. Ieong, R. K. Taira, R. Barker, and C. M. Breant. The knowledge-based object-oriented PICQUERY+ language. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):644—657, August 1993.

[CIT94] W. W. Chu, I. T. Ieong, and R. K. Taira. A semantic modeling approach for image retrieval by content. *The VLDB Journal*, 3:445—477, 1994.

[CJT96] S. K. Chang, E. Junger, and G Tortora, editors. *Intelligent Image Database Systems*. World Scientific Publishing Co., 1996.

[CK95] S. Christodoulakis and L. Koveos. Multimedia information systems: Issues and approaches. In W. Kim, editor, *Modern Database Systems*, pages 318—337. Addison-Wesley, 1995.

[CSE94] E. Clementini, J. Sharma, and M. J. Egenhofer. Modelling topological spatial relations: Strategies for query processing. *Computers and Graphics*, 18(6):815—822, 1994.

[DDI+95] Y. F. Day, S. Dagtas, M. Iino, A Khokhar, and A. Ghafoor. Object-oriented conceptual modeling of video data. In *Proceedings of the 11th International Conference on Data Engineering*, pages 401—408, Taipei, Taiwan, 1995.

[Deu90] O. Deux. The store of O_2. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):91—108, March 1990.

[DG92] N. Dimitrova and F. Golshani. EVA: A query language for multimedia information systems. In *Proceedings of Multimedia Information Systems*, Tempe, Arizona, February 1992.

[DG94] N. Dimitrova and F. Golshani. Rx for semantic video database retrieval. In *Proceedings of the 2nd ACM International Conference on Multimedia*, pages 219—226, San Francisco, CA, October 1994.

[EAT92] M. Egenhofer and K. K. Al-Taha. Reasoning about gradual changes of topological relationships. In *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, pages 196—219. Springer Verlag, September 1992.

[EF91] M. Egenhofer and R. Franzosa. Point-set topological spatial relations. *International Journal of Geographical Information Systems*, 5(2):161—174, 1991.

[Ege94] M. Egenhofer. Spatial SQL: A query and presentation language. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):86—95, January 1994.

[Fal96] C. Faloutsos, editor. *Searching Multimedia Databases by Content*. Kluwer Academic Publishers, Boston, MA, 1996.

[Gal91] D. L. Gall. MPEG: A video compression standard for multimedia applications. *Communications of ACM*, 34(4):46—58, 1991.

[Gay97] K. Gayer. Using knowledge about the document type definition for query optimization in structured document databases. M.Sc. thesis, Technical University Darmstadt Germany and GMD-IPSI, Darmstadt, Germany, March 1997.

[GBT94] S. Gibbs, C. Breiteneder, and D. Tscichritzis. Data modeling of time-based media. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 91—102, Minneapolis, Minnesota, May 1994.

[GD95] D. M. Gavrila and L. S. Davis. 3-D model-based tracking of human upper body movement: a multi-view approach. In *Proceedings of IEEE International Symposium on Computer Vision*, pages 253—258, November 1995.

[Gha96] S. Ghandeharizadeh. Stream-based versus structured video objects: Issues, solutions, and challenges. In V. S. Subrahmanian and S. Jajodia, editors, *Multimedia Database Systems: Issues and Research Directions*, pages 215—236. Springer Verlag, 1996.

[GJ96] V. N. Gudivada and G. S. Jung. An algorithm for content-based retrieval in multimedia databases. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 56—61, Hiroshima, Japan, June 1996.

[GLÖS96] I. A. Goralwalla, Y. Leontiev, M. T. Özsu, and D. Szafron. Modeling time: Back to basics. Technical Report TR-96-03, Department of Computing Science, University of Alberta, February 1996.

[GMP95] G. Gardarin, F. Machuca, and P. Pucheral. Topological relations in the world of minimum bounding rectangles: A study with R-trees. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 59—70, San Jose, CA, May 1995.

[Gol90] C. F. Goldfarb. *The SGML Handbook*. Oxford University Press, Oxford, UK, 1990.

[Gor98] I. Goralwalla. Temporality in object database management systems. PhD thesis, Department of Computing Science, University of Alberta, (*in preparation*)., 1998.

[GÖS97] I.A. Goralwalla, M.T. Özsu, and D. Szafron. Modeling Medical Trials in Pharmacoeconomics using a Temporal Object Model. *Computers in Biology and Medicine (Special Issue on Time-Oriented Systems in Medicine)*, 27(5):369—387, 1997.

[GR96] E. J. Guglielmo and N. C. Rowe. Natural-language retrieval of images based on descriptive captions. *ACM Transactions on Information Systems*, 14(3):237—267, July 1996.

[Gut84] A. Guttman. R trees: A dynamic index structure for spatial searching. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 47—57, 1984.

[Güt94] R. H. Güting. GraphDB: Modeling and querying graphs in databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 297—308, Santiago, Chile, 1994.

[GWJ91] A. Gupta, T. Weymouth, and R. Jain. The VIMSYS model. In *Proceedings of the 17th International Conference on Very Large Data Bases*, pages 297—308, Barcelona, Spain, 1991.

[Her94] D. Hernández. *Qualitative Representation of Spatial Knowledge*. Springer-Verlag, New York, 1994.

[HJ97] A. Hampapur and R. Jain. Virage video engine. In *Proceedings of Storage and Retrieval for Images and Video Databases V, IS&T/SPIE Symposium on Electronic Imaging Science and Technology*, pages 188—197, San Jose, CA, February 1997.

[HJW95] A. Hampapur, R. Jain, and T. E. Weymouth. Production model based digital video segmentation. *Multimedia Tools and Applications*, 1(1):9—46, March 1995.

[HK95] N. Hirzalla and A. Karmouch. A multimedia query specification language. In *Proceedings of International Workshop on Multimedia Database Management Systems*, pages 73—81, Blue Mountain Lake, New York, August 1995.

[IB95] S. S. Intille and A. F. Bobick. Visual tracking using closed-worlds. In *Proceedings of International Conference on Computer Vision*, Cambridge, MA, June 1995.

[IKO+96] H. Ishikawa, K. Kato, M. Ono, N. Yoshizawa, K. Kubota, and A. Kondo. A next-generation industry multimedia database system. In *Proceedings of the 12th International Conference on Data Engineering*, pages 364—371, New Orleans, Louisiana, February 1996.

[ISO86] International Standards Organization. Information processing – text and office information systems – Standard Generalized Markup Language. ISO 8870, 1986.

[ISO92] International Standards Organization. Hypermedia/Time-based Structuring Language: HyTime. ISO 10744, 1992.

[Jai91] R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc., 1991.

[JH94] R. Jain and A. Hampapur. Metadata in video database. *ACM SIGMOD RECORD*, 23(4):27—33, December 1994.

[KA95] W. Klas and K. Aberer. Multimedia applications and their implications on database architectures. In P. Apers and H. Blanken, editors, *Advanced Course on Multimedia Databases in Perspective*, The Netherlands, 1995. University of Twente.

[KC96] T. C. T. Kuo and A. L. P. Chen. A content-based query language for video databases. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 456—461, Hiroshima, Japan, June 1996.

[Kim82] W. Kim. On optimizing an SQL-like nested query. *ACM Transaction on Database Systems*, 7(3):443—469, September 1982.

[Kim95] W. Kim. Multimedia information systems: Issues and approaches. In W. Kim, editor, *Modern Database Systems*, pages 5—17. Addison-Wesley, 1995.

[LAF+93] T. D. C. Little, G. Ahanger, R. J. Folz, J. F. Gibbon, F. W. Reeve, D. H. Schelleng, and D. Venkatesh. Digital on-demand video service supporting content-based queries. In *Proceedings of the First ACM International Conference on Multimedia*, pages 427—436, Anaheim, CA, 1993.

[LG93] T. C. C. Little and A. Ghafoor. Interval-based conceptual models for time-dependent multimedia data. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):551—563, August 1993.

[LGÖS96] J. Z. Li, I. Goralwalla, M. T. Özsu, and D. Szafron. Video modeling and its integration in a temporal object model. Technical Report TR-96-02, Department of Computing Science, University of Alberta, January 1996.

[LGÖS97] J. Z. Li, I. Goralwalla, M. T. Özsu, and D. Szafron. Modeling video temporal relationships in an object database management system. In *IS&T/SPIE International Symposium on Electronic Imaging: Multimedia Computing and Networking*, pages 80—91, San Jose, CA, February 1997.

[LLOW91] C. Lamb, G. Landis, J. Orenstein, and D. Weinreb. The ObjectStore database system. *Communications of ACM*, 34(10):19—20, 1991.

[LÖ97] J. Z. Li and M. T. Özsu. STARS: A spatial attributes retrieval system for images and videos. In *Proceedings of The Fourth International Conference on Multimedia Modeling*, pages 69—84, Singapore, November 1997.

[LÖ98] J. Z. Li and M. T. Özsu. Point-set topological relations processing in image databases. In *Proceedings of The First International Forum On Multimedia & Image Processing (in press)*, Anchorage, Alaska, May 1998.

[LÖS95] J. Z. Li, M. T. Özsu, and D. Szafron. Query languages in multimedia database systems. Technical Report TR95-25, Department of Computing Science, University of Alberta, December 1995.

[LÖS96a] J. Z. Li, M. T. Özsu, and D. Szafron. Integrating video spatial relationships into an object model. Technical Report TR-96-06, Department of Computing Science, University of Alberta, March 1996.

[LÖS96b] J. Z. Li, M. T. Özsu, and D. Szafron. Modeling of moving objects in a video objectbase management system. Technical Report TR-96-12, Department of Computing Science, University of Alberta, April 1996.

[LÖS96c] J. Z. Li, M. T. Özsu, and D. Szafron. Modeling of video spatial relationships in an object database management system. In *Proceedings of the International Workshop on Multimedia Database Management Systems*, pages 124—133, Blue Mountain Lake, NY, August 1996.

[LÖS96d] J. Z. Li, M. T. Özsu, and D. Szafron. Spatial reasoning rules in multimedia management systems. In *Proceedings of the International Conference on Multimedia Modeling*, pages 119—133, Toulouse, France, November 1996.

[LÖS96e] J. Z. Li, M. T. Özsu, and D. Szafron. Spatial reasoning rules in multimedia management systems. Technical Report TR-96-05, Department of Computing Science, University of Alberta, March 1996.

[LÖS97a] J. Z. Li, M. T. Özsu, and D. Szafron. Modeling of moving objects in a video objectbase management system. In *Proceedings of The IEEE International Conference on Multimedia Computing and Systems*, pages 336—343, Ottawa, Canada, June 1997.

[LÖS97b] J. Z. Li, M. T. Özsu, and D. Szafron. Multimedia extensions to database query languages. Technical Report TR-97-01, Department of Computing Science, University of Alberta, January 1997.

[LÖS98] J. Z. Li, M. T. Özsu, and D. Szafron. *Querying Structured Documents.* (in preparation), 1998.

[LÖSO97] J. Z. Li, M. T. Özsu, D. Szafron, and V. Oria. MOQL: A multimedia object query language. In *Proceedings of The Third International Workshop on Multimedia Information Systems*, pages 19—28, Como, Italy, September 1997.

[LPE96] R. Lienhart, S. Pfeiffer, and W. Effelsberg. The MoCA workbench: Support for creativity in movie content analysis. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 314—321, Hiroshima, Japan, June 1996.

[Mac90] I. A. Macleod. Storage and retrieval of structured documents. *Information Processing & Management*, 26(2):197—208, 1990.

[Mas91] Y. Masunaga. Design issues of OMEGA: An object-oriented multimedia database management system. *Journal of Information Processing*, 14(1):60—74, 1991.

[MHM96] Elina Megalou, Thanasis Hadzilacos, and Nikos Mamoulis. Conceptual title abstractions: Modeling and querying very large interactive multimedia repositories. In *Proceedings of the International Conference on Multimedia Modeling*, pages 323—338, Toulouse, France, November 1996.

[MS96a] S. Marcus and V. S. Subrahmanian. Foundations of multimedia database systems. *Journal of ACM*, 43(3):474—523, 1996.

[MS96b] S. Marcus and V. S. Subrahmanian. Foundations of multimedia information systems. In S. Jajodia and V. S. Subrahmanian, editors, *Multimedia Database Systems: Issues and Research Directions*. Springer Verlag, 1996.

[NSN95] M. Nabil, J. Shepherd, and H. H. Ngu. 2D projection interval relationships: A symbolic representation of spatial relationships. In *Proceedings of 4th International Symposium on Large Spatial Databases*, pages 292—309, Portland, Maine, USA, August 1995.

[ÖEMI+97] M. T. Özsu, S. El-Medani, P. Iglinski, M. Schone, and D. Szafron. An object-oriented SGML/HyTime compliant multimedia database management system. In *Proceedings of The ACM International Multimedia Conference*, pages 239—249, Seattle, WA, November 1997.

[OM88] J. A. Orenstein and F. A. Manola. PROBE spatial data modeling and query processing in an image database application. *IEEE Transactions on Software Engineering*, 14(5):611—629, May 1988.

[OÖL+97] V. Oria, M. T. Özsu, L. Liu, X. Li, J. Z. Li, Y. Niu, and P. J. Iglinski. Modeling images for content-based queries: the DISIMA approach. In *Proceedings of The 2nd International Conference on Visual Information Systems*, pages 339—346, San Diego, CA, December 1997.

[ÖPS+95] M. T. Özsu, R. J. Peters, D. Szafron, B. Irani, A. Lipka, and A. Munoz. TIGUKAT: A uniform behavioral objectbase management system. *The VLDB Journal*, 4:100—147, 1995.

[ÖSEMV95] M. T. Özsu, D. Szafron, G. El-Medani, and C. Vittal. An object-oriented multimedia database system for a news-on-demond application. *ACM Multimedia Systems*, 3:182—203, 1995.

[OT93] E. Oomoto and K. Tanaka. OVID: Design and implementation of a video-object database system. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):629—643, August 1993.

[Par95] J. Paredaens. Spatial databases, the final frontier. In *Proceedings of the Fifth International Conference on Database Theory*, pages 14—32, Prague, Czech Republic, January 1995.

[PE88] D. Pullar and M. Egenhofer. Toward formal definitions of topological relations among spatial objects. In *Proceedings of the 3rd International Symposium on Spatial Data Handling*, pages 165—176, Sydney, Australia, 1988.

[Pet94] R. J. Peters. TIGUKAT: A uniform behavioral objectbase management system. PhD thesis, Department of Computing Science, University of Alberta. Available as Technical Report TR-94-06, 1994.

[PLÖS93] R. J. Peters, A. Lipka, M. T. Özsu, and D. Szafron. An extensible query model and its languages for a uniform behavioral object management system. In *Proceedings of the Second International Conference on Information and Knowledge Management*, pages 403—412, Washington D.C., USA, November 1993.

[PS94] D. Papadias and T. Sellis. The qualitative representation of spatial knowledge in two-dimensional space. *The VLDB Journal (Special Issue on Spatial Databases)*, 4:100—138, 1994.

[PS95] P. Pazandak and J. Srivastawa. The language components of DAMSEL: An embeddable event-driven declarative multimedia specification language. In *Proceedings of Storage and Retrieval for Images and Video Databases V, IS&T/SPIE Symposium on Electronic Imaging Science and Technology*, pages 121—128, October 1995.

[PSV96] C. H. Papadimitriou, D. Suciu, and V. Vianu. Topological queries in spatial databases. In *Proceedings of the 15th Symposium on Principles of Database Systems*, pages 81—92, Montreal, Canada, June 1996.

[PTSE95] D. Papadias, Y. Theodoridis, T. Sellis, and M. J. Egenhofer. Topological relations in the world of minimum bounding rectangles: A study with R-trees. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 92—103, San Jose, CA, May 1995.

[RFS88] N. Roussopoulos, C. Faloutsos, and T. Sellis. An efficient pictorial database system for PSQL. *IEEE Transactions on Software Engineering*, 14(5):639—650, May 1988.

[SAG95] H. S. Sawhney, S. Ayer, and M. Gorkani. Model-based 2D&3D dominant motion estimation for mosaicing and video representation. In *Proceedings of International Conference on Computer Vision*, Cambridge, MA, June 1995.

[SC96] J. Smith and S. F. Chang. VisualSEEK: a fully automated content-based image query system. In *Proceedings of The ACM International Multimedia Conference*, Boston, MA, 1996.

[Sch68] H. Schubert, editor. *Topology*. Allyn and Bacon, Inc, Boston, USA, 1968.

[Se94] R. T. Snodgrass and et al. TSQL2 language specification. *SIGMOD Record*, 23(1):65—86, March 1994.

[SF95] J. Sharma and D. M. Flewelling. Inferences from combined knowledge about topology and directions. In *Proceedings of 4th International Symposium on Large Spatial Databases*, pages 279—291, Portland, Maine, USA, August 1995.

[SJ97] S. Santini and R. Jain. Similarity is a geometer. *Multimedia Tools and Applications*, 5(3):306—377, November 1997.

[SK96] M. Shibata and Y. B. Kim. Content-based structuring of video information. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 330—333, Hiroshima, Japan, June 1996.

[Sno95] R. T. Snodgrass. Temporal object-oriented databases: A critical comparison. In W. Kim, editor, *Modern Database Systems*, pages 386—408. Addison-Wesley, 1995.

[SRF87] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R^+ tree: A dynamic index for multidimensional objects. In *Proceedings of the 13th International Conference on Very Large Data Bases*, pages 507—518, 1987.

[SYH94] P. Sistla, C. Yu, and R. Haddack. Reasoning about spatial relationships in picture retrieval systems. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 570—581, Santiago, Chile, 1994.

[SZ94] S. W. Smoliar and H. J. Zhang. Content-based video indexing and retrieval. *IEEE Multimedia*, 1(2):62—72, Summer 1994.

[TZ86] S. Tsur and C. Zaniolo. LDL: A logic based data language. In *Proceedings of the 12th International Conference on Very Large Data Bases*, pages 33—41, Koyoto, Japan, 1986.

[Ull85] J. D. Ullman. Implementation of logical query language for databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 444—444, Austin, Texas, May 1985.

[WBKW96] E. Wold, T. Blum, D. Keilar, and J. Wheaton. Content-based classification, search, and retrieval of audio. *IEEE Multimedia*, 3(3):27—36, Fall 1996.

[WDG94] R. Weiss, A. Duda, and D. K. Gifford. Content-based access to algebraic video. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 140—151, 1994.

[WG94] I. White and M. Goldberg, editors. *Using The Booch Method: A Rational Approach.* The Bejamin/Cummings Publishing Company, Inc., 1994.

[WJ96] D. A. White and R. Jain. Similarity indexing: Algorithms and performance. In *Proceedings of the SPIE: Storage and Retrieval for Image and Video Databases IV*, volume 2670, pages 62—75, February 1996.

[WJ97] D. A. White and R. Jain. ImageGREP: Fast visual pattern matching in image databases. In *Proceedings of Storage and Retrieval for Images and Video Databases V, IS&T/SPIE Symposium on Electronic Imaging Science and Technology*, pages 96—107, San Jose, CA, February 1997.

[WK87] D. Woelk and W. Kim. Multimedia information management in an object-oriented database system. In *Proceedings of the 13th International Conference on Very Large Data Bases*, pages 319—329, Brighton, England, September 1987.

[Wor94] M. F. Worboys. A unified model for spatial and temporal information. *The Computer Journal*, 37(1):26—34, 1994.

[YA94] T. W. Yan and J. Annevelink. Integrating a structure-text retrieval system with an object-oriented database system. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 740—749, Santiago, Chile, 1994.

[YGBC89] C. T. Yu, K. Guh, D. Brill, and A. Chen. Partition strategy for distributed query processing in fast local networks. *IEEE Transactions on Software Engineering*, pages 780—793, 1989.

[YIU96] M. Yoshikawa, O. Ichikawa, and S. Uemura. Amalgamating SGML documents and databases. In *Proceedings of the 5th International Conference on Extended Database Technology*, pages 259—274, Avignon, France, March 1996.

[YYHI96] A. Yoshitaka, M. Yoshimitsu, M. Hirakawa, and T. Ichikawa. V-QBE: Video database retrieval by means of example motion of objects. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 453—457, Hiroshima, Japan, June 1996.

[YYL96] M. Yeung, B. L. Yeo, and B. Liu. Extracting story units from long programs for video browsing and navigation. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 296—305, Hiroshima, Japan, June 1996.

Appendix A

Spatial Relation Definitions

In this section, definitions of the spatial relations are further explained through graphs. Note that Table 2.2 gives the formal definitions of the spatial relations. Since the topological relations have been shown in Figure 2.1, there is no need to repeat them. The consideration in defining directional relations can be viewed from Figure A.1. First, look at the positional relations (*above, below, left*, and *right*). For example, for relation *left* only the horizontal intervals are considered. As long as an object's interval is *before* or *meets* another object's interval, the relation *left* holds, similarly to the relation *right*. For the relations *above* and *below*, only vertical intervals are considered.

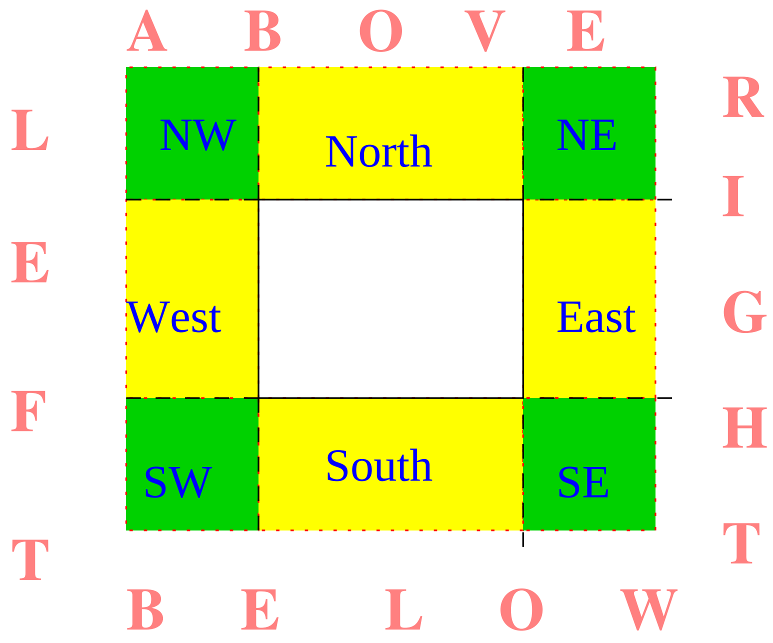


Figure A.1: Space Division by Directional Relations

Now consider strict directional relations (*north, south, west*, and *east*) and mixed directional relations (*northeast, southeast, northwest*, and *southwest*). All of their possible cases of strict directional relations and mixed directional relations are shown in Figure A.2 and Figure A.3 respectively.

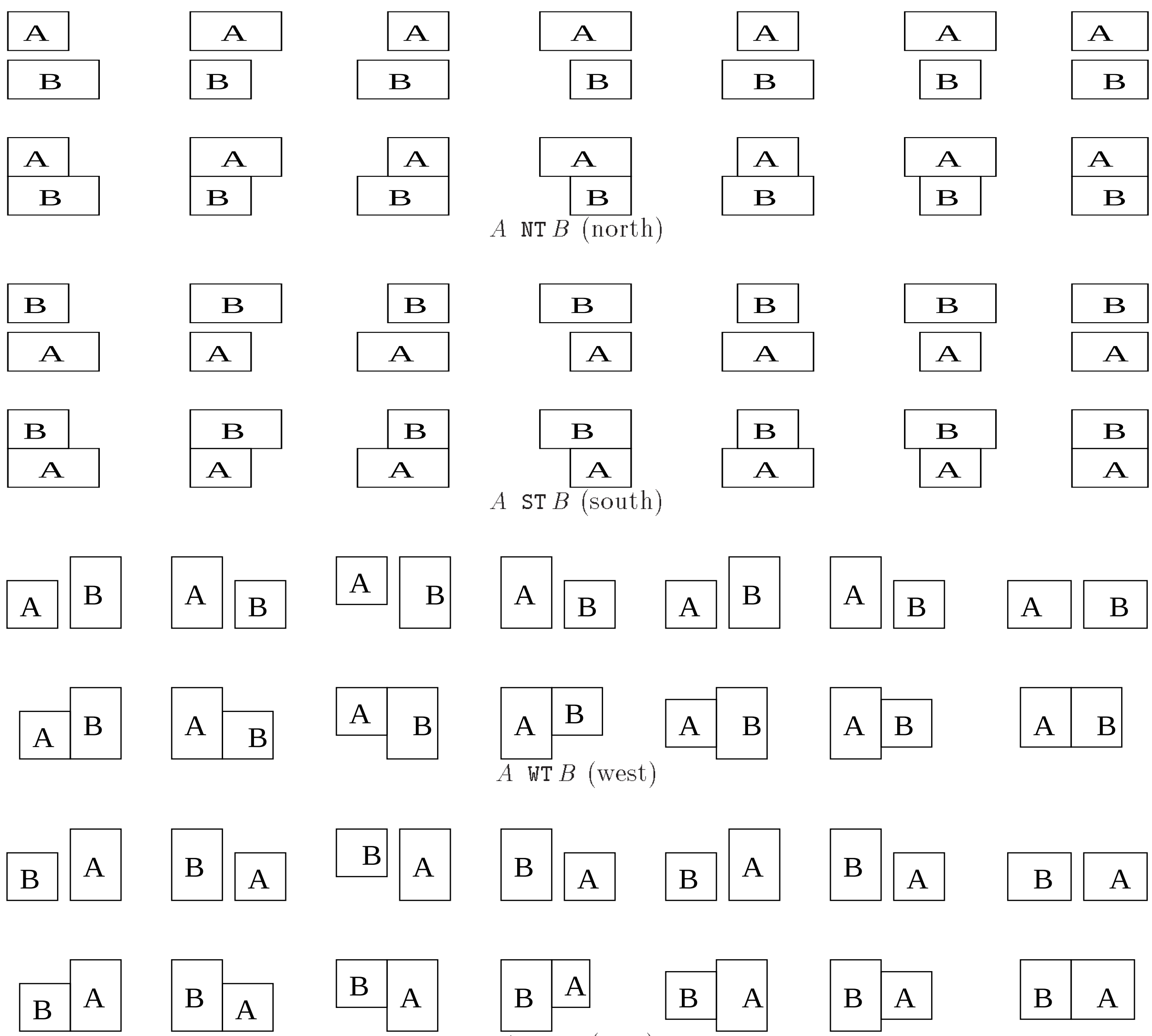


Figure A.2: Definitions of Strict Directional Relations

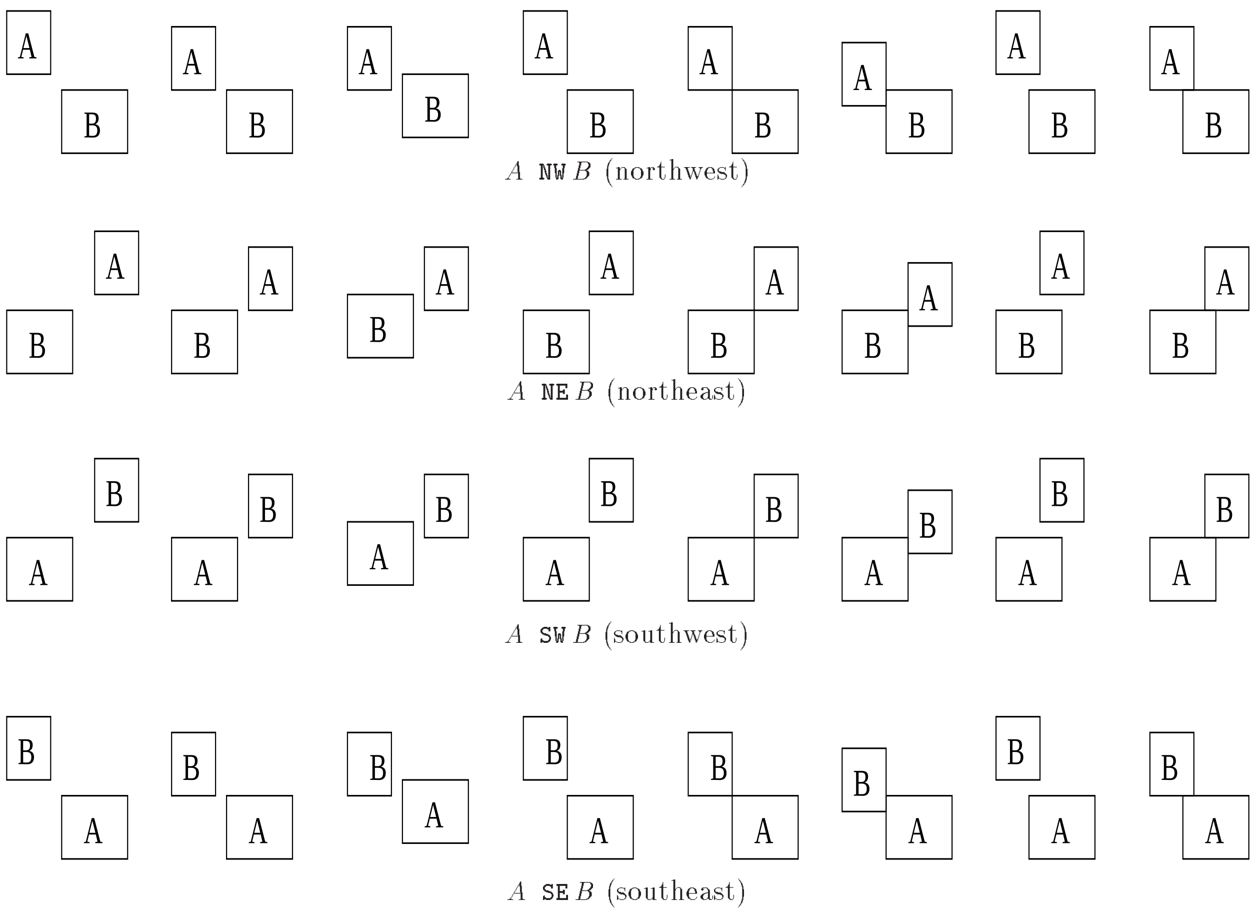


Figure A.3: Definitions of Mixed Directional Relations

Appendix B

Proof of Theorem 2.1

Section 2.4.1 introduces some fundamental topology concepts. However, in order to prove Theorem 2.1, more are necessary. The concepts of separation and connectedness are crucial for establishing the forthcoming topological spatial relations between sets. Let X be a set with a topology Ψ. If S is a subset of X then S inherits a topology from Ψ. This topology is called the *subspace topology* and is defined such that $U \subset S$ is open in the subspace topology iff $U = S \cap V$ for some set $V \in \Psi$. Under such circumstances, S is called a *subspace* of X.

Definition 13 Let X be a topological space. A *separation* of X is a pair A and B of disjoint nonempty open subsets of X whose union is X. The space X is said to be *connected* if there does not exist a separation of X. Connectedness is formulated entirely in terms of the collection of open sets of X. Connectedness can be understood as follows: A space X is connected iff the only subsets of X that are both open and closed in X are the empty set and X itself. A subset A of X is said to *separate* X if $X - A$ is disconnected. Assuming $A \subset X$, if $A^\circ \neq \mathit{empty}$ and $\overline{A} \neq X$, then A° and $X - \overline{A}$ form a separation of $X - A^\bullet$, and thus $A^\bullet$ separates X. In summary, A topological space is said to be connected if it is not the topological sum of two (non-empty) subspaces.

Definition 14 Let X be a connected topological space. A spatial region in X is a non-empty proper subset A of X satisfying (1) A° is connected and (2) $A = \overline{A^\circ}$.

Theorem 2.1 Let A be a convex region in X. In a two dimensional space, for all $p \in A^\circ$ there always exist four points p_1, p_2, p_3, p_4 in $A^\bullet$, i.e., $p_i \in A^\bullet$ $(i = 1, 2, 3, 4)$, such that
$\mathcal{F}_x(p_1) = \mathcal{F}_x(p)$ and $\mathcal{F}_y(p_1) < \mathcal{F}_y(p)$,
$\mathcal{F}_x(p_2) = \mathcal{F}_x(p)$ and $\mathcal{F}_y(p_2) > \mathcal{F}_y(p)$,
$\mathcal{F}_y(p_3) = \mathcal{F}_y(p)$ and $\mathcal{F}_x(p_3) < \mathcal{F}_x(p)$, and
$\mathcal{F}_y(p_4) = \mathcal{F}_y(p)$ and $\mathcal{F}_x(p_4) > \mathcal{F}_x(p)$.

Proof: The proof is straightforward. Consider Figure B.1 and note the shaded area A°. For any point $p \in A^\circ$, two lines along p's x axis and y axis can be drawn separately as in Figure B.1 (a). These two lines will eventually intersect with A's boundary ($A^\bullet$) because $A^\bullet$ is not empty and $A^\bullet$ separates X. Since the property of $A^\bullet$ is connected, there is no hole in spatial region A. The constraints of "$<$" and "$>$" eliminate the case that the extended two lines across p may have four intersections or more with $A^\bullet$, but p does not belong to A° as indicated in Figure B.1 (b). Because the region is a convex, the case of Figure B.1 (c) is also excluded. ■

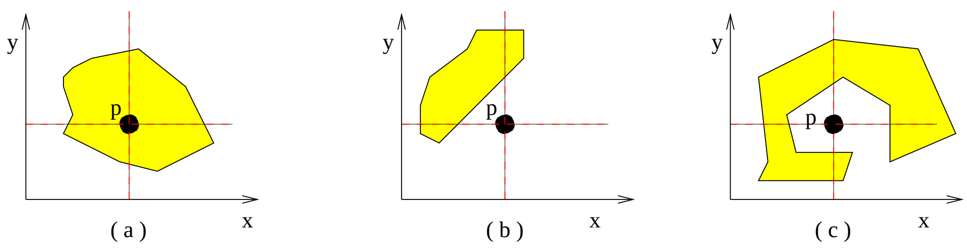


Figure B.1: Proof Examples

Appendix C

Computation of Effects

Only the procedure of computing original approach effect is discussed as the functional approach effect can be computed similarly. Table 2.3 is shown here again as Table C.1 to make this discussion complete. A good reference about the used method is [Jai91].

Index	Interior-100		Interior-200	
	Boundary-100	Boundary-200	Boundary-100	Boundary-200
No Hash	489	257	205	194
Hash	201	163	107	108

Table C.1: CPU Times (seconds) of the Original Approach

Let us define three variables x_A, x_B, and x_C as follows:

$$x_A = \begin{cases} -1 & \text{if interior set is 100} \\ 1 & \text{if interior set is 200} \end{cases}$$

$$x_B = \begin{cases} -1 & \text{if boundary set is 100} \\ 1 & \text{if boundary set is 200} \end{cases}$$

$$x_C = \begin{cases} -1 & \text{if no index is used} \\ 1 & \text{if index is used.} \end{cases}$$

The performance y can now be regressed on x_A, x_B, and x_C using nonlinear regression model of the form

$$y = q_0 + q_A x_A + q_B x_B + q_C x_C + q_{AB} x_A x_B + q_{AC} x_A x_C + q_{BC} x_B x_C + q_{ABC} x_A x_B x_C$$

Substituting the eight experiment results in the model, the following eight equations can be established:

$$489 = q_0 - q_A - q_B - q_C + q_{AB} + q_{AC} + q_{BC} - q_{ABC}$$

$$205 = q_0 + q_A - q_B - q_C - q_{AB} - q_{AC} + q_{BC} + q_{ABC}$$
$$257 = q_0 - q_A + q_B - q_C - q_{AB} + q_{AC} - q_{BC} + q_{ABC}$$
$$194 = q_0 + q_A + q_B - q_C + q_{AB} - q_{AC} - q_{BC} - q_{ABC}$$
$$201 = q_0 - q_A - q_B + q_C + q_{AB} - q_{AC} - q_{BC} + q_{ABC}$$
$$107 = q_0 + q_A - q_B + q_C - q_{AB} + q_{AC} - q_{BC} - q_{ABC}$$
$$163 = q_0 - q_A + q_B + q_C - q_{AB} - q_{AC} + q_{BC} - q_{ABC}$$
$$108 = q_0 + q_A + q_B + q_C + q_{AB} + q_{AC} + q_{BC} + q_{ABC}$$

These eight equations can be solved uniquely for the eight unknowns and the results are:

$$q_0 = \frac{1}{8}(+489 + 205 + 257 + 194 + 201 + 107 + 163 + 108) = 215$$
$$q_A = \frac{1}{8}(-489 + 205 - 257 + 194 - 201 + 107 - 163 + 108) = -35$$
$$q_B = \frac{1}{8}(-489 - 205 + 257 + 194 - 201 - 107 + 163 + 108) = -62$$
$$q_C = \frac{1}{8}(-489 - 205 - 257 - 194 + 201 + 107 + 163 + 108) = -70$$
$$q_{AB} = \frac{1}{8}(+489 - 205 - 257 + 194 + 201 - 107 - 163 + 108) = 32$$
$$q_{AC} = \frac{1}{8}(+489 - 205 + 257 - 194 + 201 + 107 - 163 + 108) = 25$$
$$q_{BC} = \frac{1}{8}(+489 + 205 - 257 - 194 - 201 - 107 + 163 + 108) = 24$$
$$q_{ABC} = \frac{1}{8}(-489 + 205 + 257 - 194 + 201 - 107 - 163 + 108) = -22.$$

The *total variation of y* or *sum of squares total* (SST) is:

$$SST = 8(q_A^2 + q_B^2 + q_C^2 + q_{AB}^2 + q_{AC}^2 + q_{BC}^2 + q_{ABC}^2) = 101424.$$

Therefore, the portion of variation explained by the seven effects are

$$E_{q_A} = 8 * q_A^2/SST = 0.10 = 10\%$$
$$E_{q_B} = 8 * q_B^2/SST = 0.30 = 30\%$$
$$E_{q_C} = 8 * q_C^2/SST = 0.39 = 39\%$$
$$E_{q_{AB}} = 8 * q_{AB}^2/SST = 0.08 = 8\%$$
$$E_{q_{AC}} = 8 * q_{AC}^2/SST = 0.05 = 5\%$$
$$E_{q_{BC}} = 8 * q_{BC}^2/SST = 0.05 = 5\%$$
$$E_{q_{ABC}} = 8 * q_{ABC}^2/SST = 0.03 = 3\%.$$

Appendix D

MOQL Specification

The OQL grammar convention is followed with only one change: “⌊” and “⌉” are used as an optional symbol to avoid confusion with symbols “[” and “]”. The MOQL extensions are italicized.

D.1 Axiom

query_program ::= {define_query;} query
define_query ::= **define** identifier **as** query

D.2 Basic

query ::= **nil** | **true** | **false** | integer_literal | float_literal | character_literal
| string_literal | entry_name | query_name | (query)

D.3 Simple Expression

query ::= query + query | query - query | query * query
| query / query | - query | query **mod** query | **abs** (query)
| query || query

D.4 Comparison

query ::= query comparison_operator query | query **like** string_literal
comparison_operator ::= = | != | > | < | >= | <=

D.5 Boolean Expression

query ::= **not** query | query **and** query | query **or** query | *spatial_expression*
| *temporal_expression* | *document_expression* | *special_expression*

D.6 Constructor

query ::= type_name (⌊ query ⌉)
| type_name (identifier : query {, identifier :} query })
| **struct** (identifier : query {, identifier : query })
| **set** (⌊ query {, query }⌉)
| **bag** (⌊ query {, query }⌉) | **list** (⌊ query {, query }⌉)
| (query , query {, query }) | ⌊ **list** ⌉(query .. query)
| **array** (⌊ query , query }⌉)

D.7 Accessor

query ::= query * query | dot attribute_name | query dot relationship_name
| query dot operation_name (query {, query }) | query [query : query]
| query [query] | **first** (query) | **last** (query)
| function_name (⌊ query {, query }⌉)

dot ::= **.** | **−>**

D.8 Collection Expression

query ::= **for all** identifier **in** query : query
| **exists** identifier **in** query : query
| **exists** (query) | **unique** (query) | query **in** query
| **count** (query) | **count** (*) | **sum** (query) | **min** (query)
| **max** (query) | **avg** (query)

quantifier ::= **some** | **any** | **all**

D.9 Select Expression

query ::= **select** ⌊ **distinct** ⌉ projection_attributes
from variable_declaration {, variable_declaration }
⌊ **where** query ⌉
⌊ **group by** partition_attributes ⌉

[**having** query]
[**order by** sort_criterion {, sort_criterion }]
[***present*** *layout* { **and** *layout*}]

projection_attributes ::= projection {, projection } | *

projection ::= query | identifier : query | query **as** identifier

variable_declaration ::= query [[**as**] identifier]

partition_attributes ::= projection {, projection }

sort_criterion ::= query [ordering]

ordering ::= **asc** | **desc**

D.10 Set Expression

query ::= query **intersect** query | query **union** query | query **except** query

D.11 Conversion

query ::= **listtoset** (query) | **element** (query) | **distinct** (query)
| **flatten** (query) | (class_name) query

D.12 Spatial Expression

spatial_expression ::= *spatial_function_expression comparison_operator*
spatial_function_expression

spatial_expression ::= *spatial_object spatial_predicate spatial_object*

spatial_function_expression ::= *spatial_term* [*arithmetic_operator spatial_function_expression*]

spatial_term ::= *float_literal* | *integer_literal* | *spatial_object* | *spatial_function* (*query*)

spatial_function ::= ***nearest*** | ***farthest*** | ***length*** | ***slope*** | ***centroid*** | ***area*** | ***mbr***
| ***distance*** | ***intersect*** | ***interior*** | ***exterior*** | ***perimeter***

spatial_predicate ::= *generic_spatial_predicate* | *topological_predicate*
| *directional_predicate* | *picture_predicate*

generic_spatial_predicate ::= ***nearest*** | ***farthest*** | ***intersect*** | ***inside*** | ***cross***
| ***centroid*** | ***midpoint***

topological_predicate ::= ***equal*** | ***touch*** | ***inside*** | ***contains*** | ***overlap*** | ***disjoint***
| ***cover*** | ***coveredBy***

directional_predicate ::= *positional_predicate* | *depth_predicate*
| *simple_directional_predicate* | *complex_directional_predicate*

positional_predicate ::= ***left*** | ***right*** | ***above*** | ***below***

depth_predicate ::= ***back*** | ***front***

simple_directional_predicate ::= ***north*** | ***south*** | ***west*** | ***east*** | ***northwest*** | ***northeast***
| ***southwest*** | ***southeast***
complex_predicate ::= *depth_predicate* _ *positional_predicate*
| *depth_predicate* _ *simple_directional_predicate*
picture_predicate ::= ***equal*** | ***coincident*** | ***subpicture*** | ***similar***
spatial_object ::= *query* | ***point*** (*integer_literal*, *integer_literal*) | ***segment*** (*point*, *point*)
| ***window*** (*point*, *point*) | ***circle*** (*point*, *radius*)

D.13 Temporal Expression

temporal_expression ::= *temporal_object temporal_predicate temporal_object*
| *temporal_function_expression comparison_operator temporal_function_expression*
temporal_function_expression ::= *temporal_term* [*arithmetic_operator*
temporal_function_expression]
temporal_term ::= *integer_literal* | *float_literal* | *temporal_object*
| *temporal_function* (*query*)
temporal_function ::= ***union*** | ***intersection*** | ***difference*** | ***add*** | ***subtract***
| ***upperBound*** | ***lowerBound*** | ***length***
| ***prior*** | ***next*** | ***nth*** | ***firstClip*** | ***lastClip*** | ***firstFrame*** | ***lastFrame***
temporal_predicate ::= ***equal*** | ***before*** | ***after*** | ***meet*** | ***metBy*** | ***overlap*** | ***overlapedBy***
| ***during*** | ***include*** | ***start*** | ***startedBy*** | ***finish*** | ***finishedBy***
temporal_object ::= *query* | ***instant*** (*integer_literal*) | ***span*** (*integer_literal*)
| ***interval*** (*integer_literal*, *integer_literal*)

D.14 Document Expression

document_expression ::= *document_function_expression comparison_operator*
document_function_expression
document_expression ::= *document_object document_predicate document_object*
document_function_expression ::= *document_term* [*arithmetic_operator*
document_function_expression]
document_term ::= *float_literal* | *integer_literal* | *document_object*
| *media_object* | *document_function* (*query*)
document_function ::= ***title*** | ***content*** | ***name*** | ***first*** | ***last*** | ***nth*** | ***sf***
document_predicate ::= ***precede*** | ***after*** | ***substructOf*** | ***contains***

D.15 Special Expression

special_expression ::= *contains_predicate* | *special_function*

contains_predicate ::= *media_object* ***contains*** *identifier*

media_object ::= *identifier*

special_function ::= *special_function_name*(*media_object*)

special_function_name ::= ***cut*** | ***fade*** | ***wipe*** | ***dissolve*** | ***zoomIn*** | ***zoomOut***
| ***panLeft*** | ***panRight*** | ***tiltUp*** | ***tiltDown***

D.16 Presentation Layout

layout ::= *spatial_layout* | *temporal_layout* | *scenario_layout*

spatial_layout ::= ***atWindow*** (*identifier*, *point*, *point*) | *image_function* (*identifier*)

temporal_layout ::= *event present_operator event*

event ::= *display_event* | *play_event* | *thumbernail_event*
| *event* ***atTime*** (*absolute_time*)

display_event ::= ***display*** (*identifier* ⌊,*start_offset* ⌊, *duration*⌉⌉)

play_event ::= ***play*** (*identifier* ⌊, *start_offset* ⌊, *duration* ⌊, *speed*⌉⌉⌉)

thumbnail_event ::= ***thumbnail*** (*identifier*)

start_offset ::= *integer_literal* ⌊*temporal_granularity*⌉

duration ::= *integer_literal* ⌊*temporal_granularity*⌉| ***forever***

speed ::= *float_literal* | ***normal***

present_operator ::= ***parStart*** | ***parEnd*** | ***after***

temporal_granularity ::= ***year*** | ***month*** | ***day*** | ***hour*** | ***minute*** | ***second*** | ***ms***

arithmetic_operator ::= + | − | ∗ | /

absolute_time ::= year / month / day / hour : minute : second

scenario_layout ::= *user_presentation_specification*

Appendix E

Algorithms for Predicates and Functions

Let O be a set of salient objects in an image. For any objects $o_1 \in O$ and $o_2 \in O$, following notations are defined:

- $c(o_1) \equiv (x_{o_1}, y_{o_1})$ is the centroid of o_1. If o_1 is a point, then its centroid is itself;
- $ps(o_1)$ is the point-set representation of o_1. $ps(o_1)$ is the union of the interior and the boundary of o_1.
- $DIST(o_1, o_2) \equiv \sqrt{(x_{o_1} - x_{o_2})^2 + (y_{o_1} - y_{o_2})^2}$ is the Euclidean distance between two objects o_1 and o_2
- a segment $l = (l_s, l_f)$, which has a starting point l_s and a finishing point l_f, can be represented by its point-set $ps(l)$.
- a region r can be represented by its point-set $ps(r)$.

Following spatial predicates (see Table 5.1) are defined:

nearest Point p_2 is the nearest point to point p_1 if $\forall p_3 \in O$, $p_3 \neq p_1$, and $DIST(p_1, p_2) \leq DIST(p_1, p_3)$.

furthest Point p_2 is the furthest point to point p_1 if $\forall p_3 \in O$, $p_3 \neq p_1$, and $DIST(p_1, p_2) \geq DIST(p_1, p_3)$.

within Point p_1 is within segment l if $c(p_1) \in ps(l)$.

midpoint Point p_1 is the midpoint of segment l if $DIST(p_1, l_s) = DIST(p_1, l_f)$.

inside Point p_1 is inside region r if $c(p_1) \in ps(r)$.

cross Segment l crosses point p if $c(p) \in ps(l)$.

intersect Segment l_1 intersects segment l_2 if there exists a point p such that $p \in ps(l_1)$ and $p \in ps(l_2)$.

inside Segment l is inside of region r if $ps(l) \subseteq ps(r)$.

cross Segment l crosses region r if $ps(l) \cap ps(r) \neq \emptyset$ and l is not inside of r.

cover Region r covers point p Segment l crosses point p if $c(p) \in ps(l)$.

cover Region r covers segment l if $ps(l) \subseteq ps(r)$.

cross Region r crosses segment l if $ps(l) \cap ps(r) \neq \emptyset$ and l is not inside of r.

Following spatial functions (see Table 5.2) are defined:

nearest Given point p_1, return point p_2 if $\forall p_3 \in O$, $p_3 \neq p_1$, and $DIST(p_1, p_2) \leq DIST(p_1, p_3)$.

furthest Given point p_1, return point p_2 if $\forall p_3 \in O$, $p_3 \neq p_1$, and $DIST(p_1, p_2) \geq DIST(p_1, p_3)$.

region Given point p, return its region $ps(p)$.

cross Given segment l, return point p for any $p \in ps(l)$.

intersect Given segments l_1 and l_2, return their intersection where $ps(l_1) \cap ps(l_2) \neq \emptyset$.

region Given segment l, return its region $ps(l)$.

length Given segment l, return its length $DIST(l_s, l_f)$.

slope Given segment l, return its slope s where $s = \frac{y_{l_s} - y_{l_f}}{x_{l_s} - x_{l_f}}$.

centroid Given region r, return $c(r)$.

interior Given region r, return r°.

exterior Given region r, return $r^{\bullet}$.

mbr Given region r, return points $p_1 = (x_{p_1}, y_{p_1})$ and $p_2 = (x_{p_2}, y_{p_2})$ where $\forall p \in ps(r), x_{p_1} \leq x_p \wedge x_{p_2} \geq x_p \wedge y_{p_1} \leq y_p \wedge y_{p_2} \geq y_p$.

area Given region r, return the cardinality of its point-set $ps(r)$ as the area. This is a gross computation value and further accuracy needs more information. For example, the scale of each point in an image must known in order to compute a meanful region area. This also applies to the following *perimeter* function.

perimeter Given region r, return the cardinality of its boundary set $r^\bullet$ as the perimeter.

Since the computation of continuous media functions (see Table 5.3) is trivial, they are omitted.

Appendix F

Formal Semantics of MOQL

The formal semantics of MOQL is investigated in this appendix. TIGUKAT defines a formal object calculus with a logical foundation that introduces a function symbol to incorporate the behavioral paradigm of the object model into the calculus. It also defines a behavioral/functional object algebra with a comprehensive set of object-preserving and object-creating operators. One important result of TIGUKAT query model is that the TIGUKAT object calculus is equivalent to the TIGUKAT object algebra. In this appendix, first TIGUKAT object calculus is introduced. Then it is proven that any MOQL **select** statement can be reduced to a TIGUKAT object calculus expression and consequently, a TIGUKAT object algebra expression. Much of the result in this appendix is borrowed from [PLÖS93, Pet94]. The expressiveness of multimedia extensions (both functions and predicates) introduced into MOQL is discussed in terms of interpreted functions and predicates.

F.1 TIGUKAT Object Calculus

TIGUKAT object calculus [PLÖS93] defines the *well-founded formulae* of the language using first-order theory. The alphabet of the calculus consists of following symbols:

Object constants:	a, b, c, d
Object variables:	o, p, q, u, v, x, y, z
Predicate symbols	
monadic:	C, P, Q, R, S, T
dyadic:	$=, \neq, \in, \notin$
n-ary:	$Eval$
Function symbol:	β
Logical connectives:	$\exists, \forall, \wedge, \vee, \neg$
Delimiters:	$($, $)$

Note that the object constants, object variables, monadic predicates and function symbols may be subscripted (e.g., a_3, o_i, C_n, β_1, etc.). In addition, a vector notation $\vec{s}$ is adopted to denote a countably infinite list of symbols $s_1, s_2, \ldots, s_n$ where $n \geq 0$.

From object constants and object variables, the syntax and semantics of the function symbol β called a *behavioral specification* (Bspec) are developed. The "evaluation" of a Bspec is accomplished by predicate $Eval$. A *term* is an object constant, an object variable or a Bspec. A Bspec is an $n+2$-ary function $\beta(s, b, \vec{t})$ where s and each t_i denote terms and where b is an object constant. For $n = 0$, $\beta(s, b)$ is used without loss of generality.

The ordered list of terms $s, b, \vec{t}$ is considered to be behaviorally consistent if and only if the following properties hold:

1. b is an object constant denoting a behavior, meaning b is not allowed to range over behaviors (functions) which ensures a first-order semantics when incorporated into a language with quantification;

2. the type of the object denoted by s defines behavior b as part of its interface, meaning b is applicable to s because it is defined on the type of s;

3. $\vec{t}$ is compatible with the arity of the argument list for behavior b, meaning the number of arguments expected by b is equivalent to the number of terms in $\vec{t}$; and

4. the type of the objects denoted by $\vec{t}$ are compatible with the argument types of behavior b, meaning the types of the terms are compatible with the argument types of b.

A Bspec $\beta(s, b, \vec{t})$ is *consistent* if and only if $s, b, \vec{t}$ are *behaviorally consistent*. The "evaluation" of a consistent Bspec involves applying the behavior b to the object denoted by term s using objects denoted by terms $\vec{t}$ as arguments. The "result" of Bspec evaluation denotes an object in the objectbase. The "evaluation" of Bspecs has the following logical formation. The $n+3$-ary predicate $Eval(R, s, b, \vec{t})$ is introduced as an axiom in the language such that $Eval(R, s, b, \vec{t})$ is true if and only if R denotes the "result" of applying behavior b to the object denoted by term s using terms $\vec{t}$ as arguments. The function symbol $\beta(s, b, \vec{t})$ is a logical representation of R. The $Eval$ predicate also serves as an enforcement of the consistency property of Bspecs. From now on only those Bspecs that are consistent are considered.

Bspecs may be composed. This provides the capability of building *path expressions* in queries. For example, given the object constants *emp*, *department* and *budget* with the obvious semantics, the Bspec $\beta(\beta(emp, department), budget)$, that denotes the object representing the annual budget of the department that employee *emp* works in, can be composed. Also note that the example Bspec has the properties of a *ground term*.

For brevity, the syntax of Bspecs is recasted into the dot notation as $s.b(\vec{t})$ which is intended as being semantically equivalent to the original specification. If behavior b does not require any arguments, then the notation simplifies to $s.b$. The previous example can then be represented as *emp.department.budget* assuming left-associativity of behavioral applications. Parenthesis may be used to change the order.

Definition 15 A *ground term* is recursively defined as follows:

1. every object constant is a ground term;
2. if $\beta(s, b, \vec{t})$ is a Bspec and all of $s, \vec{t}$ are ground terms (note that b must be a ground term by the definition of Bspec), then $\beta(s, b, \vec{t})$ is a ground term;
3. nothing else is a ground term.

From now on, symbols defined as denoting an object constant, including symbols a, b, c, d, are extended to include ground terms as well. Any term that is not a ground term is called a *variable term* since it must contain at least one object variable. If $\vec{o}$ are the object variables appearing in some term r, then r is called a *variable term over* $\vec{o}$. The variables can be thought of as the term defined by Bspec $s.b(\vec{t})$ and $\vec{o}$ represents the object variables appearing in the Bspec, then r is a variable term over $\vec{o}$. The notation $r\{\vec{o}\}$ is used to denote that r is a variable term over $\vec{o}$. This notation is generalized to $\beta\{\vec{o}\}$ when the form of the term is immaterial. If $\vec{o}$ is empty, then $\beta\{\}$ denotes a generic ground term.

The *atomic formula* or *atom* are the building blocks of calculus expressions. They represent the fundamental predicates of the calculus. The atoms of the calculus consist of the following:

Range Atom $C(o)$ is called a *range atom* for o where C corresponds to a unary predicate representing a collection and o denotes an object variable. C is said to be the *range* of o. A range atom is true if and only if o denotes an object in collection C. When C is defined for a class, it denotes the deep extent of the class and the notation is extended to include $C^+(o)$ which is true iff o denotes an object in the *shallow extent* of the class. One may think of C^+ as a separate monadic predicate for specifying the shallow range of o. Range atom specification of the form $C(s)$ where s is a term denoting an object constant or Bspec (i.e., not an object variable) are represented by membership atoms defined below.

Equality Atom $s = t$ is a built-in predicate called an *equality atom* where s and t are terms. The predicate is true iff the object denoted by term s is object identity equal to the object denoted by term t. This atom is type consistent for all objects since all objects must support an object identity equality behavior. Note, as a syntactical convenience, an equality atom where both terms are boolean and where one of the terms is the object constant *true*, say $s = ture$ where s is boolean, is simplified to s. If one of the terms is the object constant *false*, the specification is simplified to $\neg s$. The built-in predicate $s \neq t$ is the complement of equality.

Membership Atom $s \in t$ is a built-in predicate called a *membership atom* where s and t are terms and t is a term denoting a collection. The predicate is true iff the object denoted by s is an element of the collection denoted by t. Note that a range specification of the form $C(s)$ where s is an object constant or Bspec (i.e., not an object variable) is represented as a membership atom $s \in C'$ where C' is a constant denoting the collection represented by predicate C. The built-in predicate $s \notin t$ is the complement of membership.

Generating Atom An equality atom of the form $o = t$ or a membership atom $o \in t$, where o is an object variable, t is an appropriate term for the atom, and o does not appear in t, are called *generating atoms* for o. They are so named because the object denotations for o can be generated from t. o is called the *generated variable* and t is called the *generator*. Any atom that is not a generating atom is called a *restriction atom* and any variable that is not generated is called a *restriction variable*.

A *ground atom* is an atom that contains only ground terms. A *literal* is either an atom or a negated atom. A *ground literal* is a literal whose atom is a ground atom.

From atoms, the definition of a *first-order well-formed formulas*, or simply *formula* (abbreviated WFF) of the object calculus, is built. WFFs are defined in terms of *free* and *bound* object variables. An object variable is *bound* in a formula if it has been previously introduced by the quantifier $\exists$ or $\forall$. If the variable has not been introduced with a quantifier it is *free* in the formula. WFFs are defined recursively as follows:

1. Every atom is a formula. All object variables in the atom are free in the formula.
2. If ψ is a formula, then $\neg\psi$ is a formula. Object variables are free or bound in $\neg\psi$ as they are free or bound in ψ.
3. If ψ_1 and ψ_2 are formulas, then $\psi_1 \wedge \psi_2$ and $\psi_1 \vee \psi_2$ are formulas. Object variables are free or bound in $\psi_1 \wedge \psi_2$ and $\psi_1 \vee \psi_2$ as they are free or bound in ψ_1 and ψ_2.
4. If ψ is a formula, then $\exists o(\psi)$ is a formula. Free occurrences of o in ψ are bound to $\exists o$ in $\exists o(\psi)$.
5. If ψ is a formula, then $\forall o(\psi)$ is a formula. Free occurrences of o in ψ are bound to $\forall o$ in $\forall o(\psi)$.
6. Nothing else is a formula.

A, B, F, G and ψ, ω are used to denote formulas and subformulas. Furthermore, $A(x)$ means that variable x is free in formula A.

Theorem 4 Every query expressible in the first-order calculus is expressible in the TIGUKAT calculus.

The proof of this theorem can be found in [Pet94].

F.2 Formal Semantics of MOQL

The semantics of MOQL is defined in terms of TIGUKAT object calculus. It is shown in this section that every MOQL statement without aggregation function corresponds to the object calculus expression; thus there is a reduction from MOQL to TIGUKAT object calculus. Although, MOQL allows many different ways to query databases, they all can be either translated into corresponding **select** statement or easily expressed by TIGUKAT object calculus. For example, query

Persons

retrieves all the persons in the database and can be rewritten into

select p
from p in *Persons*

while

for all x **in** *Students*: x.student_id > 0

can be rewritten by TIGUKAT object calculus into:

$\forall x \in Students(x.student\_id() > 0)$.

The **select** statement is focused because it is the most complex query statement in MOQL. Throughout this section the following notation is used. Every MOQL select statement of the form:

select $p_1, \ldots, p_k$
from p_1 **in** $P_1, \ldots, p_k$ **in** P_k, q_1 **in** $Q_1, \ldots, q_n$ **in** Q_n
where $F(p_1, \ldots, p_k, q_1, \ldots, q_n)$

is referred as $S(p_1, \ldots, p_k)$. In other words, queries can be modeled as functions $S(p_1, \ldots, p_k)$ which operate upon one or more collections, and return collections as results. Lists which are returned in the result collections are made up of objects referenced by $p_1, \ldots, p_k$, and they are denoted as $< p_1, \ldots, p_k >$. Furthermore, for groups of quantifiers like $\exists p_1, \ldots, \exists p_k$ or $\forall p_1, \ldots, \forall p_k$, the shorthand notation is used: $\exists < p_1, \ldots, p_k >$ and $\forall < p_1, \ldots, p_k >$ respectively. Finally, $< p_1, \ldots, p_k > = < x_1, \ldots, x_k >$ is a short notation for $< p_1 = x_1, \ldots, p_k = x_k >$.

It is shown in this section that every select statement $S(p_1, \ldots, p_k)$ corresponds to the OCE: $\{< p_1, \ldots, p_k > | \phi(< p_1, \ldots, p_k >)\}$. The *select clause* in $S(p_1, \ldots, p_k)$ defines free variables of the OCE formula. The *from clause* specifies the ranges of variables which can either be given explicitly as constant references to collection, or implicitly in form of subqueries. If the range variable is defined over a constant collection reference, then it corresponds to the **range atom**. If it ranges over a collection defined by a variable or a path expression then it corresponds to a **membership atom**. Otherwise, in case of subqueries, the semantics of the range variable is defined by a complex OCE formula. However, as shown in the theorem, every query can be rewritten into an equivalent flat query.

Lemma 1 Any subquery in **select** clause can be rewritten into the **from** clause.

Proof: The proof is done by introducing a temporary collection variable for each subquery in the **select** clause. Then the temporary variable is used as the surrogate of the subquery. I.e.,

$$
\begin{array}{lll}
\textbf{select} & \textbf{select} & p_{11}, \ldots, p_{1n_1} \\
 & \textbf{from} & p_{11} \textbf{ in } S_{11}(q_{11}), \ldots, p_{1n_1} \textbf{ in } S_{1n_1}(q_{1n_1}), \ldots, r_1 \textbf{ in } \#R_1 \\
 & \textbf{where} & F_1(p_{11}, \ldots, p_{1n_1}, r_1), \\
 & \textbf{select} & p_{21}, \ldots, p_{2n_2} \\
 & \textbf{from} & p_{21} \textbf{ in } S_{21}(q_{21}), \ldots, p_{2n_2} \textbf{ in } S_{2n_2}(q_{2n_2}), \ldots, r_2 \textbf{ in } \#R_2 \\
 & \textbf{where} & F_2(p_{21}, \ldots, p_{2n_2}, r_2), \\
 & & \vdots \\
 & \textbf{select} & p_{k1}, \ldots, p_{kn_k} \\
 & \textbf{from} & p_{k1} \textbf{ in } S_{k1}(q_{k1}), \ldots, p_{kn_k} \textbf{ in } S_{kn_k}(q_{kn_k}), \ldots, r_k \textbf{ in } \#R_k \\
 & \textbf{where} & F_k(p_{k1}, \ldots, p_{kn_k}, r_k), \\
 & \multicolumn{2}{l}{p_{k+1}, p_{k+2}, \ldots, p_m} \\
\textbf{from} & \multicolumn{2}{l}{p_{k+1} \textbf{ in } S_{k+1}(q_{k+1}), \ldots, p_m \textbf{ in } S_m(q_m), r \textbf{ in } \#R} \\
\textbf{where} & \multicolumn{2}{l}{F(p_{k+1}, \ldots, p_m, r)}
\end{array}
$$

can be rewritten into

$$
\begin{array}{lll}
\textbf{select} & \multicolumn{2}{l}{p_1, \ldots, p_k, p_{k+1}, \ldots, p_m} \\
\textbf{from} & \textbf{select} & p_{11}, \ldots, p_{1n_1} \\
 & \textbf{from} & p_{11} \textbf{ in } S_{11}(q_{11}), \ldots, p_{1n_1} \textbf{ in } S_{1n_1}(q_{1n_1}), \ldots, r_1 \textbf{ in } \#R_1 \\
 & \textbf{where} & F_1(p_{11}, \ldots, p_{1n_1}, r_1) \quad \textbf{as } p_1, \\
 & \textbf{select} & p_{21}, \ldots, p_{2n_2} \\
 & \textbf{from} & p_{21} \textbf{ in } S_{21}(q_{21}), \ldots, p_{2n_2} \textbf{ in } S_{2n_2}(q_{2n_2}), \ldots, r_2 \textbf{ in } \#R_2 \\
 & \textbf{where} & F_2(p_{21}, \ldots, p_{2n_2}, r_2) \quad \textbf{as } p_2, \\
 & & \vdots \\
 & \textbf{select} & p_{k1}, \ldots, p_{kn_k} \\
 & \textbf{from} & p_{k1} \textbf{ in } S_{k1}(q_{k1}), \ldots, p_{kn_k} \textbf{ in } S_{kn_k}(q_{kn_k}), \ldots, r_k \textbf{ in } \#R_k \\
 & \textbf{where} & F_k(p_{k1}, \ldots, p_{kn_k}, r_k) \quad \textbf{as } p_k, \\
 & \multicolumn{2}{l}{p_{k+1} \textbf{ in } S_{k+1}(q_{k+1}), \ldots, p_m \textbf{ in } S_m(q_m), r \textbf{ in } \#R} \\
\textbf{where} & \multicolumn{2}{l}{F(p_{k+1}, \ldots, p_m, r)}
\end{array}
$$

■

Lemma 2 All the **as** predicates in **from** clause can be eliminated from the **from** clause.

Proof:

$$
\begin{array}{lll}
\textbf{select} & \multicolumn{2}{l}{p_1, \ldots, p_k, p_{k+1}, \ldots, p_m} \\
\textbf{from} & \textbf{select} & p_{11}, \ldots, p_{1n_1} \\
 & \textbf{from} & p_{11} \textbf{ in } S_{11}(q_{11}), \ldots, p_{1n_1} \textbf{ in } S_{1n_1}(q_{1n_1}), \ldots, r_1 \textbf{ in } \#R_1 \\
 & \textbf{where} & F_1(p_{11}, \ldots, p_{1n_1}, r_1) \quad \textbf{as } p_1, \\
 & \textbf{select} & p_{21}, \ldots, p_{2n_2} \\
 & \textbf{from} & p_{21} \textbf{ in } S_{21}(q_{21}), \ldots, p_{2n_2} \textbf{ in } S_{2n_2}(q_{2n_2}), \ldots, r_2 \textbf{ in } \#R_2 \\
 & \textbf{where} & F_2(p_{21}, \ldots, p_{2n_2}, r_2) \quad \textbf{as } p_2, \\
 & & \vdots \\
 & \textbf{select} & p_{k1}, \ldots, p_{kn_k} \\
 & \textbf{from} & p_{k1} \textbf{ in } S_{k1}(q_{k1}), \ldots, p_{kn_k} \textbf{ in } S_{kn_k}(q_{kn_k}), \ldots, r_k \textbf{ in } \#R_k \\
 & \textbf{where} & F_k(p_{k1}, \ldots, p_{kn_k}, r_k) \quad \textbf{as } p_k, \\
 & \multicolumn{2}{l}{p_{k+1} \textbf{ in } S_{k+1}(q_{k+1}), \ldots, p_m \textbf{ in } S_m(q_m), r \textbf{ in } \#R} \\
\textbf{where} & \multicolumn{2}{l}{F(p_{k+1}, \ldots, p_m, r)}
\end{array}
$$

Since $F(p_{k+1}, \ldots, p_m, r)$ does not contain $p_1, \ldots, p_k$, they do not interfere with each other. Therefore, the above query can be rewritten into

$$
\begin{array}{ll}
\textbf{select} & p_1, \ldots, p_k, p_{k+1}, \ldots, p_m \\
\textbf{from} & p_{k+1} \textbf{ in } S_{k+1}(q_{k+1}), \ldots, p_m \textbf{ in } S_m(q_m), r \textbf{ in } \#R \\
\textbf{where} & F(p_{k+1}, \ldots, p_m, r) \quad \textbf{and} \\
 & p_1 = \textbf{select } p_{11}, \ldots, p_{1n_1} \\
 & \qquad \textbf{from } p_{11} \textbf{ in } S_{11}(q_{11}), \ldots, p_{1n_1} \textbf{ in } S_{1n_1}(q_{1n_1}), \ldots, r_1 \textbf{ in } \#R_1 \\
 & \qquad \textbf{where } F_1(p_{11}, \ldots, p_{1n_1}, r_1) \quad \textbf{and} \\
 & p_2 = \textbf{select } p_{21}, \ldots, p_{2n_2} \\
 & \qquad \textbf{from } p_{21} \textbf{ in } S_{21}(q_{21}), \ldots, p_{2n_2} \textbf{ in } S_{2n_2}(q_{2n_2}), \ldots, r_2 \textbf{ in } \#R_2 \\
 & \qquad \textbf{where } F_2(p_{21}, \ldots, p_{2n_2}, r_2) \quad \textbf{and} \\
 & \qquad \qquad \vdots \\
 & p_k = \textbf{select } p_{k1}, \ldots, p_{kn_k} \\
 & \qquad \textbf{from } p_{k1} \textbf{ in } S_{k1}(q_{k1}), \ldots, p_{kn_k} \textbf{ in } S_{kn_k}(q_{kn_k}), \ldots, r_k \textbf{ in } \#R_k \\
 & \qquad \textbf{where } F_k(p_{k1}, \ldots, p_{kn_k}, r_k)
\end{array}
$$

and further,

$$
\begin{array}{ll}
\textbf{select} & p_1, \ldots, p_k, p_{k+1}, \ldots, p_m \\
\textbf{from} & p_{k+1} \textbf{ in } S_{k+1}(q_{k+1}), \ldots, p_m \textbf{ in } S_m(q_m), r \textbf{ in } \#R \\
\textbf{where} & F(p_{k+1}, \ldots, p_m, r) \quad \textbf{and} \\
 & p_1 = S_1(q_1) \quad \textbf{and} \\
 & p_2 = S_2(q_2) \quad \textbf{and} \\
 & \qquad \vdots \\
 & p_k = S_k(q_k)
\end{array}
$$

which is equivalent to

$$
\begin{array}{ll}
\textbf{select} & p_1, \ldots, p_k, p_{k+1}, \ldots, p_m \\
\textbf{from} & p_{k+1} \textbf{ in } S_{k+1}(q_{k+1}), \ldots, p_m \textbf{ in } S_m(q_m), r \textbf{ in } \#R \\
\textbf{where} & F'(p_1, p_2, \ldots, p_k, p_{k+1}, \ldots, p_m, r)
\end{array}
$$

■

Lemma 3 Every MOQL **select** statement with nested queries in the **from** clause can be rewritten into equivalent flat query.

Proof: From Lemma 1 and Lemma 2, it is known that every MOQL **select** statement can be translated into following format:

$$
\begin{array}{ll}
\textbf{select} & p_1, \ldots, p_k, p_{k+1}, \ldots, p_m \\
\textbf{from} & p_1 \textbf{ in } \#P_1, \ldots, p_k \textbf{ in } \#P_k, \\
 & p_{k+1} \textbf{ in } S_{k+1}(q_{k+1}), \ldots, p_m \textbf{ in } S_m(q_m), \\
 & r \textbf{ in } \#R \\
\textbf{where} & F(p_1, \ldots, p_k, p_{k+1}, \ldots, p_m, r)
\end{array}
$$

which is equivalent to the object formula:

$$
\exists p_1, \ldots, \exists p_m \quad (P_1(p_1) \wedge \ldots \wedge P_k(p_k) \wedge p_{k+1} \textbf{ in } S_{k+1}(q_{k+1}) \wedge \ldots \wedge
$$
$$
p_m \textbf{ in } S_m(q_m) \wedge \exists r(R(r) \wedge F(p_1, \ldots, p_k, p_{k+1}, \ldots, p_m, r)))
$$

$P_1, \ldots, P_k$ in this query are constant references to collections, r represents all variables which appear in the query, but not in the **select** clause, and $S_{k+1}(q_{k+1}), \ldots, S_m(q_m)$ represent subqueries. Thus, every $S_{k+i}(i = 1, \ldots, m-k)$ is also a query, and it is represented in MOQL as:

$$
S_{k+i}(q_{k+i}) \quad \equiv \quad
\begin{array}{l}
\textbf{select } q_{k+i} \\
\textbf{from } q_{k+i} \textbf{ in } \#Q_{k+i}, r_{k+i} \textbf{ in } \#R_{k+i} \\
\textbf{where } F_{k+i}(q_{k+i}, r_{k+i})
\end{array}
$$

which is equivalent to the following OCE:

$$S_{k+i}(q_{k+i}) \equiv \exists q_{k+i}(Q_{k+i}(q_{k+i}) \wedge \exists r_{k+i}(R_{k+i}(r_{k+i}) \wedge F_{k+i}(q_{k+i}, r_{k+i})))$$

Furthermore, every subformula in the **from** clause which is in the form: p_{k+i} **in** $S_{k+i}(q_{k+i})$ is equivalent to:

$$p_{k+i} \textbf{ in } S_{k+i}(q_{k+i}) \equiv \exists q_{k+i}(Q_{k+i}(q_{k+i}) \wedge \exists r_{k+i}(R_{k+i}(r_{k+i}) \wedge F_{k+i}(q_{k+i}, r_{k+i})) \wedge p_{k+i} = q_{k+i})$$

In the above formula, every q_{k+i} can be replaced by p_{k+i} yielding an equivalent formula:

$$p_{k+i} \textbf{ in } S_{k+i}(q_{k+i}) \equiv \exists q_{k+i}(Q_{k+i}(p_{k+i}) \wedge \exists r_{k+i}(R_{k+i}(r_{k+i}) \wedge F_{k+i}(p_{k+i}, r_{k+i}))$$

Thus, by replacing each p_{k+i} **in** $S_{k+i}(q_{k+i})$ in 6.1 the following equivalent formula is obtained:

$$\begin{aligned} \exists p_1, \ldots, p_k, p_{k+1}, \ldots, \exists p_m \quad (P_1(p_1) \wedge \ldots \wedge P_k(p_k) \wedge (Q_{k+1}(p_{k+1} \wedge \exists r_{k+1}(R_{k+1}) \wedge \\ F_{k+1}(p_{k+1}, r_{k+1})) \wedge \ldots \wedge (Q_m(p_m) \wedge \exists r_k(R_m(r_m) \wedge F_m(p_m, r_m))) \wedge \exists r(R(r) \wedge \\ F(p_1, \ldots, p_m, r))) \end{aligned}$$

The above OCE is in conjunctive form: therefore, changing the order of predicates results in a logically equivalent OCE. Thus, in a new OCE, all range atoms of the form $P_i(p_i), Q_i(q_i), R_i(r_i)$ are put together, and all well-formed formulas of the form

$$F(p_1, \ldots, p_m, r), \ldots, F_k(p_k, r_k)$$

are put together. The equivalent OCE is as follows:

$$\begin{aligned} \exists p_1, \ldots, p_k, p_{k+1}, \ldots, \exists p_m \quad (P_1(p_1) \wedge \ldots \wedge P_k(p_k) \wedge (Q_{k+1}(p_{k+1} \wedge \exists r_{k+1}(R_{k+1}) \wedge \\ F_{k+1}(p_{k+1}, r_{k+1})) \wedge \ldots \wedge (Q_m(p_m) \wedge \exists r_{k+}(R_{k+1}(r_{k+1}) \wedge \ldots \wedge \\ \exists r_m(R_m(r_m) \wedge F_{k+1}(p_{i+1}, ri+1) \wedge \ldots \wedge F_m(p_m, r_m) \wedge F(p_1, \ldots, p_m, r))))) \ldots) \end{aligned}$$

Thus, the original query $S(p_1, \ldots, p_k, p_{k+1}, \ldots, p_m)$ can be rewritten to the following form:

select $p_1, \ldots, p_k, p_{k+1}, \ldots, p_m$
from p_1 **in** $\#P_1, \ldots, p_k$ **in** $\#P_k$,
p_{k+1} **in** $\#Q_{k+1}, \ldots, p_m$ **in** $\#Q_m$,
r_{k+1} **in** $\#R_{k+1}, \ldots, r_m$ **in** $\#R_m$, r **in** $\#R$
where $F_{k+1}(p_{k+1}, r_{k+1}) \wedge \ldots \wedge F_m(p_m, r_k) \wedge F(p_1, \ldots, p_k, p_{k+1}, \ldots, p_m, r)$

This proves the lemma.

■

Lemma 4 Every boolean expression in the **where** clause of the **select** statement corresponds to a well-formed formula in OCE.

Proof:

1. ***in* predicate:** it has the general form: *identifier* **in** *query*. Since a query always returns a collection (a bag or a set depends on whether or not **distint** is specified). If *query* is a constant reference to a collection, it corresponds to a **range atom** in the OCE. Otherwise, it corresponds to a **membership atom**.

2. ***exists* predicate:** the exist quantifier in MOQL is expressed by: **exists** (*query*) — **exists** *identifier* **in** $query_1$: $query_2$. The **exists** predicate is *true* if *query* result is not empty or there exists an *identifier* in $query_1$ such that $query_2$ is evaluated to be *true*. Otherwise, the predicate is *false*. The *query*, $query_1$, or $query_2$ can be either a collection constant reference or a **select** statement. In the first case, the **exists** predicate has the format: **exists** P, and it is equivalent to the OCE $\exists x P(x)$. In the second case it has the format: **exists** $S(p_1, \ldots, p_m)$. Then it corresponds to the following:

$$\exists < x_1, \ldots, x_m > \ \ (\exists < p_1, \ldots p_m > (S(p_1, \ldots p_m)) \wedge < p_1, \ldots, p_k > = < x_1, \ldots, x_k >)$$

3. ***for all* predicate:** the universal quantifier is expressed in MOQL by: **for all** *identifier* **in** $query_1$: $query_2$. This predicate is *true* if for every *identifier* in $query_1$, the $query_2$ is evaluated to be *true*. On the other hand, if there exists one *identifier* in $query_1$ such that $query_2$ is evaluated to be *false*, the predicate is evaluated to be *false*. In general, this predicate can be written as follows:

$$\textbf{for all } p_1 \textbf{in} \# P_1, \ldots, p_k \textbf{in} \# P_k, p_{k+1} \textbf{in} \# S_{k+1}(q_{k+1}) \ldots, p_m \textbf{in} S_m(q_m) \\ F(p_1, \ldots, p_k, p_{k+1}, \ldots, p_m)$$

where $P_1, \ldots, P_k$ are constant collection references, $S_{k+1}(q_{k+1}), \ldots, S_m(q_m)$ are MOQL queries, and $F(p_1, \ldots, p_k, p_{k+1}, \ldots, p_m)$ are complex boolean expression which may contain queries. The following OCE is equivalent to the above predicate:

$$\forall p_1 \ldots, p_k, p_{k+1} \ldots, p_m((\neg P_1(p_1) \vee \ldots \vee \neg P_k(p_k) \vee \neg(S_{k+1}(q_{k+1}) \wedge p_{k+1} = q_{k+1}) \\ \vee \ldots \vee \ \neg(S_m(q_m) \wedge p_m = q_m)) \vee F(p_1, \ldots, p_k, p_{k+1}, \ldots, p_m)$$

4. **Boolean expression:** The translation from Boolean expression (connected by **not**, **and**, and **or**) into OCE is trivial and is omitted.

5. **Set expression:** MOQL set expressions have following three formats:

$$query_1 \ \textbf{intersect} \ query_2$$
$$query_1 \ \textbf{union} \ query_2$$
$$query_1 \ \textbf{except} \ query_2$$

Since both $query_1$ and $query_2$ are sets, they can be expressed by Q_1 and Q_2 respectively. Then, following OCE can be used to represent the above set expressions respectively:

$$\{\ o \mid \ Q_1(o) \wedge \ Q_2(o)\}$$
$$\{\ o \mid \ Q_1(o) \vee \ Q_2(o)\}$$
$$\{\ o \mid Q_1(o) \wedge \neg Q_2(o)\}$$

■

Theorem 5 Every **select** statement without aggregate clauses in MOQL has an equivalent OCE.

Proof: This is a direct result from Lemmas 3 – 5.

■

F.3 Interpreted Functions and Predicates

In this section the use of arbitrary interpreted functions and predicates in MOQL are considered. In a practical language (like MOQL), predefined built-in functions (like *area, average*), and predicates (like *left, zoomIn*), are necessary. Otherwise, users have to define their owns. Such functions and predicates are called *interpreted* [AB95] since their interpretations (including the interpretation of their input and output domains) are fixed. This is contrast to the use of function symbols in logical programming, where the functions are assigned a meaning in the Herbrand Universe, hence are not interpreted. The extensions to OQL, including multimedia functions and predicates, can be viewed as interpreted.

domain independence and related notions must be considered because defining the notion of domain independence in the presence of interpreted functions is a problem. Assume that a set Θ_{func} of functions and a set Θ_{pred} of predicates are given. Functions in Θ_{func} and predicates in Θ_{pred} are typed. Furthermore, the domain names used in those types are associated with fixed domains. Also it is assumed that for each f in Θ_{func} , f^{-1} is also in Θ_{func}. Note that when a function is not 1–1, its inverse is set-valued; since there are sets in the calculus, that is not a problem. The assumption implies that for each x, the set $f^{-1}(x)$ is finite. A database instance is constrained in that the interpretations for some of the domains are fixed as described in the scheme. It also "contains" the interpreted functions and predicates. Such a database structure is denoted by $DBI =< [D_1, \ldots, D_k], \vec{T}, I >$, where D_i $(i = 1, \ldots, k)$ are domain names, $\vec{T}$ is the vector of tuples and I is the interpretations of the fixed domains and of the additional functions and predicates.

Now consider domain independence. Given a query, it is not enough to consider for the active domain the values that appear in the database or in the query; it must be closed under applications of functions and their inverses. For example if the database contains 1, and the functions include $+$, then queries can ask for each of the integers, hence the domain should contain all integers. On the other hand, having infinite sets included in the active domain seems to defeat the intention in the definition of domain independence. In a database scheme, some of the domain names are now attached to fixed domains. For example, the name *int* may be used in the scheme, and its interpretation is fixed to be the integers. In addition, the sets Θ_{func} and Θ_{pred} are included in the scheme, with their fixed interpretations. In order to have a useful notion of domain independence, a restricted (semantic) notion of domain independence presented in [AB95] is used. The intuition behind the following definition is that in any given query, there is a bound on the number of times functions (and their inverses) are applied.

Definition 16 Given a database DBI, and a set of constants C, let $\Phi(\vec{T}, C)$ be the set of atomic values of any atomic type that appear in $\vec{T}$ and C, and for any value x, let $\Phi(x)$ be the set of atomic values that appear in x. The *n-closure* of $\Phi(\vec{T}, C)$, denoted by $close^n(\vec{T}, C)$, is defined as follows:

- $close^0(\vec{T}, C) = \Phi(\vec{T}, C)$
- $close^{n+1}(\vec{T}, C) = close^n(\vec{T}, C) \cup \bigcup\{\Phi(I(f)(x_1, \ldots, x_l)) \mid$
  $f \in \Theta_{func}, \forall i = 1, \ldots, l, \Phi(x_i) \subseteq close^n(\vec{T}, C)\} \cup \{\Phi(x) \mid$
  $\exists x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_l \; \Phi(f(x_1, x_{i-1}, x, x_{i+1}, \ldots, x_l)) \in close^n(\vec{T}, C)\}$

Thus, assuming the natural numbers with *succ* and $succ^{-1}$ and initially given the numbers 1 and 8, then two closure steps generate the set $\{0, 1, 2, 3, 6, 7, 8, 9, 10\}$. Now a query q is defined to be n-depth domain independent if q depends only on the values of the n-closure of $\Phi(\vec{T}, C_q)$, where C_q is the set of constants used in the query and this set is finite. This implies that the query on any database can be evaluated, where the interpretations of the interpreted domains are with their fixed interpretations on this closure, but may be quite different outside it. Necessarily, the interpreted functions and predicates must be given an interpretation on those new domains. All are required is that these interpretations agree with the fixed interpretations on the given closure. Thus, to make this notion of n-depth domain independence precise, the restriction on the interpretations of the functions and predicates has to be relaxed in $\Theta_{func} \cup \Theta_{pred}$ and their domains. Formally, for n-depth independence, it is allowed to use any domains that include $close^n(\vec{T}, C_q)$, and such that the computation of the n-closure in these domains gives the same results as in the original domains. The last requirement ensures, in particular, that if an inverse function is applied to any elements in $close^{i-1}(\vec{T}, C)$, only elements in $close^i(\vec{T}, C)$, for all $i = 1, \ldots, n$, are obtained even though the functions may have arbitrary behavior outside this set. Assuming this relaxation, a query is *n-depth domain independent* if $q(DBI) = q(DBI')$, for any DBI' that agrees with DBI on $close^n(\vec{T}, C_q)$, as described above. A query is *bounded depth domain independent* if it is n-depth domain independent, for some n.

Theorem 6 Let Θ_{func} and Θ_{pred} are given. A query q containing $f \in \Theta_{func}$ or $p \in \Theta_{pred}$ is expressible by a bounded-depth domain independent OCE.

Proof: The proof is completed by first showing that q is expressible by TIGUKAT object algebra and then, applying the result of the equivalence of TIGUKAT object algebra and TIGUKAT object calculus. For an algebraic query that contains no functions, the depth is 0. For a **map** operation that is constructed using *tuple construction, set construction*, and *attribute selection*, the depth is the maximum of the depths of the argument **map** operations. The same holds when a function or predicate is applied to **map** operation. Assume the **map** operation $f = bfmap < g_1 > (g_2)$, and the depths of g_1, g_2 being n_1, n_2, respectively, then the depth of f is $n_1 + n_2$. Finally, if the depth of g is n, and $f \in \Theta_{func}$, then the depth of $f(g)$ is $n+1$. For queries, the depth of $\phi(Q_1, \ldots, Q_m)$ is the maximum of the depths of Q_i's, except when ϕ is **map**. For **map** $< f > (E)$, the depth is the sum of the depths of f and E.

It is easy to see that if the depth of an algebraic expression E is n, then it is n-depth domain independent. Further, the construction in [Pet94] is used to construct an equivalent object calculus expression, which is also n-depth independent. The only extensions to the constructions are in the treatment of replace specifications: in a *select*, θ may be any of the predicates in Θ_{pred}, not just one of the three built-in predicates $\in, =, \subseteq$, and a function from Θ_{func} may be applied to **map** operations, in addition to being able to apply algebraic operations. For example, if formula ψ_g is for the **map** g, then the formula for $f(g)$ is $\exists v_1(\psi_g(u, v_1) \wedge v = f(v_1))$.

■

Appendix G

Proofs of Spatial Reasoning Rules

In this appendix the correctness (soundness) of all the inference rules given in Chapter 2 is proven. Since the results of the transitivity table described in [All83] as Figure 4 are frequently referenced, the table is redraw in Table G.1 and is cited it as *Trans. Table*. Several handy lemmas are introduced before the formal proofs are given.

	b	bi	d	di	o	oi	m	mi	s	si	f	fi
b	b		b o m d s	b	b	b o m d s	b	b o m d s	b	b	b o m d s	b
bi		bi	bi oi mi d f	bi	bi oi mi d f	bi	bi oi mi d f	bi	bi oi mi d f	bi	bi	bi
d	b	bi	d		b o m d s	bi oi mi d f	b	bi	d	bi oi mi d f	d	b o m d s
di	b o m di fi	bi oi mi si di	o oi d di e	di	o di fi	oi di si	o di fi	oi di si	di fi o	di	di si oi	di
o	b	bi oi mi si di	o d s	b o m di fi	b o m	o oi d di e	b	oi di si	o	di fi o	d s o	b o m
oi	b o m di fi	bi	oi d f	bi oi di si mi	o oi d di e	bi oi mi	o di fi	bi	oi d f	oi bi mi	oi	oi di si
m	b	bi oi mi si di	o d s	b	b	o d s	b	f fi e	m	m	d s o	b
mi	b o m di fi	bi	oi d f	bi	oi d f	bi	s si e	bi	d f oi	bi	mi	mi
s	b	bi	d	b o m di fi	b o m	oi d f	b	mi	s	s si e	d	b m o
si	b o m di fi	bi	oi d f	di	o di fi	oi	o di fi	mi	s si e	si	oi	di
f	b	bi	d	bi oi di si mi	o d s	bi oi mi	m	bi	d	bi oi mi	f	f fi e
fi	b	bi oi di si mi	o d s	di	o	oi di si	m	si oi di	o	di	f fi e	fi

Table G.1: The Transitivity Table (Figure 4 in [All83])

Lemma 5 $\{\mathtt{d},\mathtt{di},\mathtt{s},\mathtt{si},\mathtt{f},\mathtt{fi},\mathtt{e}\}$ and $\{\mathtt{d},\mathtt{di},\mathtt{s},\mathtt{si},\mathtt{f},\mathtt{fi},\mathtt{o},\mathtt{oi},\mathtt{e}\}$ are symmetric:
$A\,\{\mathtt{d},\mathtt{di},\mathtt{s},\mathtt{si},\mathtt{f},\mathtt{fi},\mathtt{e}\}\,B \Leftrightarrow B\,\{\mathtt{d},\mathtt{di},\mathtt{s},\mathtt{si},\mathtt{f},\mathtt{fi},\mathtt{e}\}\,A$
$A\,\{\mathtt{d},\mathtt{di},\mathtt{s},\mathtt{si},\mathtt{f},\mathtt{fi},\mathtt{o},\mathtt{oi},\mathtt{e}\}\,B \Leftrightarrow B\,\{\mathtt{d},\mathtt{di},\mathtt{s},\mathtt{si},\mathtt{f},\mathtt{fi},\mathtt{o},\mathtt{oi},\mathtt{e}\}\,A$.

Proof: The proof is trivial if note that all the relations and their inverses are in the set. For example, from $A\,\{\mathtt{d}\}\,B \Leftrightarrow B\,\{\mathtt{di}\}\,A$, it is true that $A\,\{\mathtt{d},\mathtt{di}\}\,B \Leftrightarrow B\,\{\mathtt{d},\mathtt{di}\}\,A$. ∎

Lemma 6 $A\,\{\mathtt{o}\}\,B \wedge B\,\{\mathtt{o}\}\,C \Rightarrow A\,\{\mathtt{b},\mathtt{m},\mathtt{o}\}\,C$. (a direct result from Trans. Table)

Lemma 7 $A\,\{\mathtt{b},\mathtt{m}\}\,B \wedge B\,\{\mathtt{o}\}\,C \Rightarrow A\,\{\mathtt{b}\}\,C$. (a direct result from Trans. Table)

Lemma 8 $A\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,B \wedge B\,\{\mathtt{bi},\mathtt{mi}\}\,C \Rightarrow A\,\{\mathtt{bi}\}\,C$.

Proof: $A\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,B$ indicates that A is always to the right of B and $B\,\{\mathtt{bi},\mathtt{mi}\}\,C$ indicates that B is always to the right of C. Therefore, in a one dimensional space, A is always to the right of C, i.e., $A\,\{\mathtt{bi}\}\,C$. Furthermore $A\,\{\mathtt{mi}\}\,C$ cannot be true because A and B have neither the relation *started_by* (si) nor the relation *equal* (e). ∎

Lemma 9 $A\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,B \wedge B\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,C \Rightarrow A\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,C$.

Proof: $A\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,B \wedge B\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,C$
$\Leftrightarrow (A\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,B \wedge B\,\{\mathtt{bi},\mathtt{mi}\}\,C) \vee (A\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,B \wedge B\,\{\mathtt{oi}\}\,C)$
$\Rightarrow A\,\{\mathtt{bi}\}\,C \vee (A\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,B \wedge B\,\{\mathtt{oi}\}\,C)$ (Lemma 8)
$\Rightarrow A\,\{\mathtt{bi}\}\,C \vee A\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,C$ (Trans. Table)
$\Rightarrow A\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,C$. ∎

Lemma 10 $A\,\{\mathtt{b},\mathtt{m},\mathtt{o},\mathtt{d},\mathtt{s}\}\,B \wedge B\,\{\mathtt{b},\mathtt{m}\}\,C \Rightarrow A\,\{\mathtt{b}\}\,C$.

Proof: $A\,\{\mathtt{b},\mathtt{m},\mathtt{o},\mathtt{d},\mathtt{s}\}\,B \wedge B\,\{\mathtt{b},\mathtt{m}\}\,C$
$\Leftrightarrow (A\,\{\mathtt{b},\mathtt{m},\mathtt{o}\}\,B \wedge B\,\{\mathtt{b},\mathtt{m}\}\,C) \vee (A\,\{\mathtt{d},\mathtt{s}\}\,B \wedge B\,\{\mathtt{b},\mathtt{m}\}\,C)$
$\Rightarrow A\,\{\mathtt{b}\}\,C \vee (A\,\{\mathtt{d},\mathtt{s}\}\,B \wedge B\,\{\mathtt{b},\mathtt{m}\}\,C)$ (Inverse Property and Lemma 8)
$\Rightarrow A\,\{\mathtt{b}\}\,C \vee (A\,\{\mathtt{b}\}\,C)$ (Trans. Table)
$\Rightarrow A\,\{\mathtt{b}\}\,C$. ∎

Lemma 11 $A\,\{\mathtt{b},\mathtt{m},\mathtt{o},\mathtt{d},\mathtt{s},\mathtt{f},\mathtt{fi},\mathtt{e}\}\,B \wedge B\,\{\mathtt{b},\mathtt{m}\}\,C \Rightarrow A\,\{\mathtt{b},\mathtt{m}\}\,C$

Proof: $A\,\{\mathtt{b},\mathtt{m},\mathtt{o},\mathtt{d},\mathtt{s},\mathtt{f},\mathtt{fi},\mathtt{e}\}\,B \wedge B\,\{\mathtt{b},\mathtt{m}\}\,C$
$\Leftrightarrow (A\,\{\mathtt{b},\mathtt{m},\mathtt{o},\mathtt{d},\mathtt{s}\}\,B \wedge B\,\{\mathtt{b},\mathtt{m}\}\,C) \vee (A\,\{\mathtt{f},\mathtt{fi},\mathtt{e}\}\,B \wedge B\,\{\mathtt{b},\mathtt{m}\}\,C)$
$\Rightarrow A\,\{\mathtt{b}\}\,C \vee (A\,\{\mathtt{f},\mathtt{fi},\mathtt{e}\}\,B \wedge B\,\{\mathtt{b},\mathtt{m}\}\,C)$ (Lemma 10)
$\Rightarrow A\,\{\mathtt{b}\}\,C \vee (A\,\{\mathtt{b},\mathtt{m}\}\,C)$ (Trans. Table)
$\Rightarrow A\,\{\mathtt{b},\mathtt{m}\}\,C$. ∎

Lemma 12 $A\,\{\mathtt{b},\mathtt{m}\}\,B \wedge B\,\{\mathtt{d},\mathtt{di},\mathtt{s},\mathtt{si},\mathtt{f},\mathtt{fi},\mathtt{o},\mathtt{oi},\mathtt{e}\}\,C \Rightarrow A\,\{\mathtt{b},\mathtt{m},\mathtt{o},\mathtt{d},\mathtt{s}\}\,C$.

Proof: $A\,\{\mathtt{b},\mathtt{m}\}\,B \wedge B\,\{\mathtt{d},\mathtt{di},\mathtt{s},\mathtt{si},\mathtt{f},\mathtt{fi},\mathtt{o},\mathtt{oi},\mathtt{e}\}\,C$

$\Leftrightarrow (A\,\{\mathtt{b}\}\,B \wedge B\,\{\mathtt{d},\mathtt{di},\mathtt{s},\mathtt{si},\mathtt{f},\mathtt{fi},\mathtt{o},\mathtt{oi},\mathtt{e}\}\,C) \vee (A\,\{\mathtt{m}\}\,B \wedge B\,\{\mathtt{d},\mathtt{di},\mathtt{s},\mathtt{si},\mathtt{f},\mathtt{fi},\mathtt{o},\mathtt{oi},\mathtt{e}\}\,C)$

$\Rightarrow (A\,\{\mathtt{b},\mathtt{m},\mathtt{o},\mathtt{d},\mathtt{s}\}\,C) \vee (A\,\{\mathtt{m}\}\,B \wedge B\,\{\mathtt{d},\mathtt{di},\mathtt{s},\mathtt{si},\mathtt{f},\mathtt{fi},\mathtt{o},\mathtt{oi},\mathtt{e}\}\,C)$ (Trans. Table)

$\Rightarrow A\,\{\mathtt{b},\mathtt{m},\mathtt{o},\mathtt{d},\mathtt{s}\}\,C \vee (A\,\{\mathtt{b},\mathtt{m},\mathtt{o},\mathtt{d},\mathtt{s}\}\,C)$ (Trans. Table)

$\Rightarrow A\,\{\mathtt{b},\mathtt{m},\mathtt{o},\mathtt{d},\mathtt{s}\}\,C$ ■

Lemma 13 $A\,\{\mathtt{b},\mathtt{m}\}\,B \wedge B\,\{\mathtt{d},\mathtt{di},\mathtt{s},\mathtt{si},\mathtt{f},\mathtt{fi},\mathtt{o},\mathtt{oi},\mathtt{m},\mathtt{mi},\mathtt{e}\}\,C \Rightarrow A\,\{\mathtt{b},\mathtt{m},\mathtt{o},\mathtt{d},\mathtt{s},\mathtt{f},\mathtt{fi},\mathtt{e}\}\,C$.

Proof: $A\,\{\mathtt{b},\mathtt{m}\}\,B \wedge B\,\{\mathtt{d},\mathtt{di},\mathtt{s},\mathtt{si},\mathtt{f},\mathtt{fi},\mathtt{o},\mathtt{oi},\mathtt{m},\mathtt{mi},\mathtt{e}\}\,C$

$\Leftrightarrow (A\,\{\mathtt{b},\mathtt{m}\}\,B \wedge B\,\{\mathtt{d},\mathtt{di},\mathtt{s},\mathtt{si},\mathtt{f},\mathtt{fi},\mathtt{o},\mathtt{oi},\mathtt{e}\}\,C) \vee (A\,\{\mathtt{b},\mathtt{m}\}\,B \wedge B\,\{\mathtt{m},\mathtt{mi}\}\,C)$

$\Rightarrow (A\,\{\mathtt{b},\mathtt{m},\mathtt{o},\mathtt{d},\mathtt{s}\}\,C) \vee (A\,\{\mathtt{b},\mathtt{m}\}\,B \wedge B\,\{\mathtt{m},\mathtt{mi}\}\,C)$ (Lemma 12)

$\Rightarrow A\,\{\mathtt{b},\mathtt{m},\mathtt{o},\mathtt{d},\mathtt{s}\}\,C \vee (A\,\{\mathtt{b}\}\,B \wedge B\,\{\mathtt{m},\mathtt{mi}\}\,C) \vee (A\,\{\mathtt{m}\}\,B \wedge B\,\{\mathtt{m},\mathtt{mi}\}\,C)$

$\Rightarrow A\,\{\mathtt{b},\mathtt{m},\mathtt{o},\mathtt{d},\mathtt{s}\}\,C \vee (A\,\{\mathtt{b},\mathtt{m},\mathtt{o},\mathtt{d},\mathtt{s}\}\,C) \vee (A\,\{\mathtt{f},\mathtt{fi},\mathtt{e}\}\,C)$ (Trans. Table)

$\Rightarrow A\,\{\mathtt{b},\mathtt{m},\mathtt{o},\mathtt{d},\mathtt{s}\}\,C \vee A\,\{\mathtt{f},\mathtt{fi},\mathtt{e}\}\,C$

$\Rightarrow A\,\{\mathtt{b},\mathtt{m},\mathtt{o},\mathtt{d},\mathtt{s},\mathtt{f},\mathtt{fi},\mathtt{e}\}\,C$. ■

Theorem 7 **Rules 1-15** are correct.

Proof:

Rule 1

Since $A_x\,\{\mathtt{e}\}\,A_x$ and $A_y\,\{\mathtt{e}\}\,A_y$, from the definition of $\mathtt{EQ}$, $A\,\mathtt{EQ}\,A$ can be derived.

Rule 3

$A\,\mathtt{NT}\,B \Leftrightarrow A_x\,\{\mathtt{d},\mathtt{di},\mathtt{s},\mathtt{si},\mathtt{f},\mathtt{fi},\mathtt{e}\}\,B_x \wedge A_y\,\{\mathtt{bi},\mathtt{mi}\}\,B_y$

$\Leftrightarrow B_x\,\{\mathtt{d},\mathtt{di},\mathtt{s},\mathtt{si},\mathtt{f},\mathtt{fi},\mathtt{e}\}\,A_x \wedge A_y\,\{\mathtt{bi},\mathtt{mi}\}\,B_y$ (Lemma 5)

$\Leftrightarrow B_x\,\{\mathtt{d},\mathtt{di},\mathtt{s},\mathtt{si},\mathtt{f},\mathtt{fi},\mathtt{e}\}\,A_x \wedge B_y\,\{\mathtt{b},\mathtt{m}\}\,A_y$ (Inverse property)

$\Leftrightarrow B\,\mathtt{ST}\,A$.

It is similar to other strict directional relations. As for the positional directional relations:

$A\,\mathtt{LT}\,B \Leftrightarrow A_x\,\{\mathtt{b},\mathtt{m}\}\,B_x \Leftrightarrow B_x\,\{\mathtt{bi},\mathtt{mi}\}\,A_x \Leftrightarrow B\,\mathtt{RT}\,A$

$A\,\mathtt{AB}\,B \Leftrightarrow A_y\,\{\mathtt{bi},\mathtt{mi}\}\,B_y \Leftrightarrow B_y\,\{\mathtt{b},\mathtt{m}\}\,A_y \Leftrightarrow B\,\mathtt{BL}\,A$.

Rule 4

$$A\,\mathtt{NW}\,B \wedge B\,\mathtt{NW}\,C \Leftrightarrow [(A_x\,\{\mathtt{b},\mathtt{m}\}\,B_x \wedge A_y\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,B_y) \vee (A_x\,\{\mathtt{o}\}\,B_x \wedge A_y\,\{\mathtt{bi},\mathtt{mi}\}\,B_y)] \wedge \\ [(B_x\,\{\mathtt{b},\mathtt{m}\}\,C_x \wedge B_y\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,C_y) \vee (B_x\,\{\mathtt{o}\}\,C_x \wedge B_y\,\{\mathtt{bi},\mathtt{mi}\}\,C_y)]$$

There are four cases to consider:

1. $A_x\,\{\mathtt{b},\mathtt{m}\}\,B_x \wedge A_y\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,B_y \wedge B_x\,\{\mathtt{b},\mathtt{m}\}\,C_x \wedge B_y\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,C_y$

   $\Leftrightarrow A_x\,\{\mathtt{b},\mathtt{m}\}\,B_x \wedge B_x\,\{\mathtt{b},\mathtt{m}\}\,C_x \wedge A_y\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,B_y \wedge B_y\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,C_y$

   $\Rightarrow A_x\,\{\mathtt{b}\}\,C_x \wedge A_y\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,B_y \wedge B_y\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,C_y$ (Lemma 10)

   $\Rightarrow A_x\,\{\mathtt{b}\}\,C_x \wedge A_y\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,C_y$ (Lemma 9)

   $\Rightarrow A\,\mathtt{NW}\,C$

2. $A_x\,\{\mathtt{b},\mathtt{m}\}\,B_x \wedge A_y\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,B_y \wedge B_x\,\{\mathtt{o}\}\,C_x \wedge B_y\,\{\mathtt{bi},\mathtt{mi}\}\,C_y$

   $\Leftrightarrow A_x\,\{\mathtt{b},\mathtt{m}\}\,B_x \wedge B_x\,\{\mathtt{o}\}\,C_x \wedge A_y\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,B_y \wedge B_y\,\{\mathtt{bi},\mathtt{mi}\}\,C_y$

   $\Rightarrow A_x\,\{\mathtt{b}\}\,C_x \wedge A_y\,\{\mathtt{bi},\mathtt{mi},\mathtt{oi}\}\,B_y \wedge B_y\,\{\mathtt{bi},\mathtt{mi}\}\,C_y$ (Lemma 7)

   $\Rightarrow A_x\,\{\mathtt{b}\}\,C_x \wedge A_y\,\{\mathtt{bi}\}\,C_y$ (Lemma 8)

   $\Rightarrow A\,\mathtt{NW}\,C$

3. $A_x\,\{\texttt{o}\}\,B_x \wedge A_y\,\{\texttt{bi},\texttt{mi}\}\,B_y \wedge B_x\,\{\texttt{b},\texttt{m}\}\,C_x \wedge B_y\,\{\texttt{bi},\texttt{mi},\texttt{oi}\}\,C_y$
   $\Leftrightarrow A_x\,\{\texttt{o}\}\,B_x \wedge B_x\,\{\texttt{b},\texttt{m}\}\,C_x \wedge A_y\,\{\texttt{bi},\texttt{mi}\}\,B_y \wedge B_y\,\{\texttt{bi},\texttt{mi},\texttt{oi}\}\,C_y$
   $\Rightarrow A_x\,\{\texttt{o}\}\,B_x \wedge B_x\,\{\texttt{b},\texttt{m}\}\,C_x \wedge A_y\,\{\texttt{bi}\}\,C_y$ (Lemma 8)
   $\Rightarrow A_x\,\{\texttt{b}\}\,C_x \wedge A_y\,\{\texttt{bi}\}\,C_y$ (Lemma 7)
   $\Rightarrow A\,\texttt{NW}\,C$

4. $A_x\,\{\texttt{o}\}\,B_x \wedge A_y\,\{\texttt{bi},\texttt{mi}\}\,B_y \wedge B_x\,\{\texttt{o}\}\,C_x \wedge B_y\,\{\texttt{bi},\texttt{mi}\}\,C_y$
   $\Leftrightarrow A_x\,\{\texttt{o}\}\,B_x \wedge B_x\,\{\texttt{o}\}\,C_x \wedge A_y\,\{\texttt{bi},\texttt{mi}\}\,B_y \wedge B_y\,\{\texttt{bi},\texttt{mi}\}\,C_y$
   $\Rightarrow A_x\,\{\texttt{o}\}\,B_x \wedge B_x\,\{\texttt{o}\}\,C_x \wedge A_y\,\{\texttt{bi}\}\,C_y$ (Lemma 8)
   $\Rightarrow A_x\,\{\texttt{b},\texttt{m},\texttt{o}\}\,C_x \wedge A_y\,\{\texttt{bi}\}\,C_y$ (Lemma 6)
   $\Rightarrow A\,\texttt{NW}\,C$

Similar proof can be constructed for other mixed directional relations.

Rule 9

$A\,\texttt{LT}\,B \wedge B\,\texttt{OL}\,C \wedge C\,\texttt{LT}\,D$
$\Leftrightarrow A_x\,\{\texttt{b},\texttt{m}\}\,B_x \wedge B_x\,\{\texttt{d},\texttt{di},\texttt{s},\texttt{si},\texttt{f},\texttt{fi},\texttt{o},\texttt{oi},\texttt{e}\}\,C_x \wedge B_y\,\{\texttt{d},\texttt{di},\texttt{s},\texttt{si},\texttt{f},\texttt{fi},\texttt{o},\texttt{oi},\texttt{e}\}\,C_y \wedge C_x\,\{\texttt{b},\texttt{m}\}\,D_x$
$\Rightarrow A_x\,\{\texttt{b},\texttt{m}\}\,B_x \wedge B_x\,\{\texttt{d},\texttt{di},\texttt{s},\texttt{si},\texttt{f},\texttt{fi},\texttt{o},\texttt{oi},\texttt{e}\}\,C_x \wedge C_x\,\{\texttt{b},\texttt{m}\}\,D_x$ (drop y-interval)
$\Rightarrow A_x\,\{\texttt{b},\texttt{m},\texttt{o},\texttt{d},\texttt{s}\}\,C_x \wedge C_x\,\{\texttt{b},\texttt{m}\}\,D_x$ (Lemma 12)
$\Rightarrow A_x\,\{\texttt{b}\}\,D_x$ (Lemma 10)
$\Rightarrow A\,\texttt{LT}\,D$
Others are similar.

Rule 13

$A\,\texttt{LT}\,B \wedge B\,\{\texttt{WT},\texttt{NW},\texttt{SW}\}\,C \Rightarrow A\,\texttt{LT}\,C$. There are three cases to consider:

1. $A\,\texttt{LT}\,B \wedge B\,\texttt{WT}\,C$
   $\Leftrightarrow A_x\,\{\texttt{b},\texttt{m}\}\,B_x \wedge B_x\,\{\texttt{b},\texttt{m}\}\,C_x \wedge B_y\,\{\texttt{d},\texttt{di},\texttt{s},\texttt{si},\texttt{f},\texttt{fi},\texttt{e}\}\,C_y$
   $\Rightarrow A_x\,\{\texttt{b},\texttt{m}\}\,B_x \wedge B_x\,\{\texttt{b},\texttt{m}\}\,C_x$ (drop y-interval)
   $\Rightarrow A_x\,\{\texttt{b}\}\,C_x$ (Lemma 10)
   $\Rightarrow A\,\texttt{LT}\,C$

2. $A\,\texttt{LT}\,B \wedge B\,\texttt{NW}\,C$
   $\Leftrightarrow A_x\,\{\texttt{b},\texttt{m}\}\,B_x \wedge [(B_x\,\{\texttt{b},\texttt{m}\}\,C_x \wedge B_y\,\{\texttt{bi},\texttt{mi},\texttt{oi}\}\,C_y) \vee (B_x\,\{\texttt{o}\}\,C_x \wedge B_y\,\{\texttt{bi},\texttt{mi}\}\,C_y]$
   $\Rightarrow A_x\,\{\texttt{b},\texttt{m}\}\,B_x \wedge (B_x\,\{\texttt{b},\texttt{m}\}\,C_x \vee B_x\,\{\texttt{o}\}\,C_x)$ (drop y-interval)
   $\Rightarrow (A_x\,\{\texttt{b},\texttt{m}\}\,B_x \wedge B_x\,\{\texttt{b},\texttt{m}\}\,C_x) \vee (A_x\,\{\texttt{b},\texttt{m}\}\,B_x \wedge B_x\,\{\texttt{o}\}\,C_x)$
   $\Rightarrow (A_x\,\{\texttt{b}\}\,C_x) \vee (A_x\,\{\texttt{b},\texttt{m}\}\,B_x \wedge B_x\,\{\texttt{o}\}\,C_x)$ (Lemma 10)
   $\Rightarrow A_x\,\{\texttt{b}\}\,C_x \vee A_x\,\{\texttt{b}\}\,C_x$ (Lemma 7)
   $\Rightarrow A_x\,\{\texttt{b}\}\,C_x$
   $\Rightarrow A\,\texttt{LT}\,C$

3. $A\,\texttt{LT}\,B \wedge B\,\texttt{SW}\,C$
   $\Leftrightarrow A_x\,\{\texttt{b},\texttt{m}\}\,B_x \wedge [(B_x\,\{\texttt{b},\texttt{m}\}\,C_x \wedge B_y\,\{\texttt{b},\texttt{m},\texttt{o}\}\,C_y) \vee (B_x\,\{\texttt{o}\}\,C_x \wedge B_y\,\{\texttt{b},\texttt{m}\}\,C_y]$
   $\Rightarrow A_x\,\{\texttt{b},\texttt{m}\}\,B_x \wedge (B_x\,\{\texttt{b},\texttt{m}\}\,C_x \vee B_x\,\{\texttt{o}\}\,C_x)$ (drop y-interval)
   $\Rightarrow A_x\,\{\texttt{b}\}\,C_x$ (proof of case 2 of Rule 13)
   $\Rightarrow A\,\texttt{LT}\,C$

Rule 14

$A\,\texttt{LT}\,B \wedge B\,\texttt{TC}\,C \wedge C\,\texttt{LT}\,D$

$\Leftrightarrow A_x\,\{\mathtt{b},\mathtt{m}\}\,B_x \wedge [(B_x\,\{\mathtt{m},\mathtt{mi}\}\,C_x \wedge B_y\,\{\mathtt{d},\mathtt{di},\mathtt{s},\mathtt{si},\mathtt{f},\mathtt{fi},\mathtt{o},\mathtt{oi},\mathtt{m},\mathtt{mi},\mathtt{e}\}\,C_y)\vee$
$(B_x\,\{\mathtt{d},\mathtt{di},\mathtt{s},\mathtt{si},\mathtt{f},\mathtt{fi},\mathtt{o},\mathtt{oi},\mathtt{m},\mathtt{mi},\mathtt{e}\}\,C_x \wedge B_y\,\{\mathtt{m},\mathtt{mi}\}\,C_y)] \wedge C_x\,\{\mathtt{b},\mathtt{m}\}\,D_x$
$\Rightarrow A_x\,\{\mathtt{b},\mathtt{m}\}\,B_x \wedge (B_x\,\{\mathtt{m},\mathtt{mi}\}\,C_x \vee B_x\,\{\mathtt{d},\mathtt{di},\mathtt{s},\mathtt{si},\mathtt{f},\mathtt{fi},\mathtt{o},\mathtt{oi},\mathtt{m},\mathtt{mi},\mathtt{e}\}\,C_x)\wedge$
$C_x\,\{\mathtt{b},\mathtt{m}\}\,D_x$ (Drop y-interval)
$\Leftrightarrow (A_x\,\{\mathtt{b},\mathtt{m}\}\,B_x \wedge B_x\,\{\mathtt{m},\mathtt{mi}\}\,C_x \wedge C_x\,\{\mathtt{b},\mathtt{m}\}\,D_x)\vee$
$(A_x\,\{\mathtt{b},\mathtt{m}\}\,B_x \wedge B_x\,\{\mathtt{d},\mathtt{di},\mathtt{s},\mathtt{si},\mathtt{f},\mathtt{fi},\mathtt{o},\mathtt{oi},\mathtt{m},\mathtt{mi},\mathtt{e}\}\,C_x \wedge C_x\,\{\mathtt{b},\mathtt{m}\}\,D_x)$
$\Rightarrow (A_x\,\{\mathtt{b},\mathtt{m}\}\,B_x \wedge B_x\,\{\mathtt{m},\mathtt{mi}\}\,C_x \wedge C_x\,\{\mathtt{b},\mathtt{m}\}\,D_x)\vee$
$(A_x\,\{\mathtt{b},\mathtt{m},\mathtt{o},\mathtt{d},\mathtt{s},\mathtt{f},\mathtt{fi},\mathtt{e}\}\,C_x \wedge C_x\,\{\mathtt{b},\mathtt{m}\}\,D_x)$ (Lemma 13)
$\Rightarrow (A_x\,\{\mathtt{b},\mathtt{m}\}\,B_x \wedge B_x\,\{\mathtt{m},\mathtt{mi}\}\,C_x \wedge C_x\,\{\mathtt{b},\mathtt{m}\}\,D_x) \vee (A_x\,\{\mathtt{b},\mathtt{m}\}\,D_x)$ (Lemma 11)
$\Rightarrow (A_x\,\{\mathtt{b},\mathtt{m},\mathtt{o},\mathtt{d},\mathtt{s},\mathtt{f},\mathtt{fi},\mathtt{e}\}\,C_x \wedge C_x\,\{\mathtt{b},\mathtt{m}\}\,D_x) \vee (A_x\,\{\mathtt{b},\mathtt{m}\}\,D_x)$ (Trans. Table)
$\Rightarrow (A_x\,\{\mathtt{b},\mathtt{m}\}\,D_x) \vee A_x\,\{\mathtt{b},\mathtt{m}\}\,D_x$ (Lemma 11)
$\Rightarrow A_x\,\{\mathtt{b},\mathtt{m}\}\,D_x$
$\Rightarrow A\,\mathtt{LT}\,C$

The proofs of **Rules 2**, **5 — 8** are trivial, the proof of **Rule 10** is similar to the proof of **Rule 9**, and the proofs of **Rules 11** and **12** are similar to the proof of **Rule 13**. They all are omitted. ■