

6-1-1995

An Object-Oriented Query Language for Multimedia Database Systems

Young Francis Day

Purdue University School of Electrical and Computer Engineering

Arif Ghafoor

Purdue University School of Electrical and Computer Engineering

Follow this and additional works at: <http://docs.lib.purdue.edu/ecetr>

Day, Young Francis and Ghafoor, Arif, "An Object-Oriented Query Language for Multimedia Database Systems" (1995). *ECE Technical Reports*. Paper 137.

<http://docs.lib.purdue.edu/ecetr/137>

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

AN OBJECT-ORIENTED QUERY
LANGUAGE FOR MULTIMEDIA
DATABASE SYSTEMS

YOUNG FRANCIS DAY
ARIF GHAFOOR

TR-ECE 95-16
JUNE 1995



SCHOOL OF ELECTRICAL
AND COMPUTER ENGINEERING
PURDUE UNIVERSITY
WEST LAFAYETTE, INDIANA 47907-1285

An Object-Oriented Query Language for Multimedia Database Systems

Young Francis Day and Arif Ghafoor
School of Electrical and Computer Engineering;
1285 Electrical Engineering Building
Purdue University
West Lafayette, IN 47907-1285

Contents

1	Introduction	1
2	Background	3
2.1	Object-Oriented Model for Multimedia Data	4
2.2	Generalized n-ary Relations	5
3	The Proposed Language and Schema Declaration	9
3.1	Object-Oriented Schema Declaration	9
3.2	Queries for Multimedia Document Retrieval	11
3.3	Events and Content-Based Queries for Image/Video Data	15
3.4	Queries for Playout Control	23
4	System Architecture	25
5	Conclusion	27

List of Tables

1	<i>n</i> -ary relations	6
2	Spatial Object	19
3	Temporal event	20

List of Figures

1	Evolution of design approaches of multimedia	3
2	Media class and its subclasses	4
3	n -ary relations	7
4	Inter-media synchronization class and its subclasses	8
5	Example OCPN, timeline, and spatial layout at t_1	13
6	The user's perception of the document for imprecise query formulation . . .	16
7	The architecture of the system	25

Abstract

In this paper, we propose a general-purpose multimedia query language that is built upon a set of generalized n-ary spatio-temporal relations and object-oriented modelling paradigm for **multimedia** data. We present a grammar for the query language and elaborate how various **functionalities** such as declaration of multimedia data, specification of spatio-temporal logic, **expression** of spatio-temporal semantics for content-based retrieval of **image/video** data, composition of multimedia documents, and orchestration of presentations can be supported through this language. Currently, the language is being implemented using object-oriented concepts and the Postgres database management system.

1 Introduction

Most of the emerging multimedia applications assume a backend database management system, which provides facilities for indexing, storing, and retrieving multimedia data including images, audio, graphics, video and text. Multimedia data possesses spatio-temporal characteristics. Temporal characteristics deal with the synchronization, which is the process of coordinating the real-time presentation of information and maintaining the time-varying ordered relations among component media [23]. Spatial characteristics deal with the process of presentation of each object at appropriate place on the screen and the relative foreground/background relationships among media objects as they are displayed concurrently [11]. Spatio-temporal based event characterization and semantic modelling of image/video data are needed for content-based retrieval of such data.

Management and retrieval of multimedia data is a complex problem and a number of researchers have addressed various aspects of this problem. Few attempts have been made to extend the relational model to manage such data. In one approach data is represented as BLOBs (Binary Large Object Blocks) but this approach does not allow any direct manipulation of information within a BLOB. This model is also unable to handle spatio-temporal requirements. The rich semantics of multimedia data need new data models. As a result, most multimedia applications adopt object-oriented approaches [1, 6, 8][10]-[17][20, 26].

In order to develop a general purpose multimedia database management system, a query language is needed that must be complete and should have strong expressive power to specify not only spatio-temporal concepts, but should also allow manipulation of complex multimedia objects. A database query language supports the definition and manipulation of data, which reflects the underlying data model. Many object-oriented query languages [2, 3, 5, 7, 22, 30] have been proposed in the literature. Most of them emphasize the classification and inheritance properties of object-oriented concepts in the domain of textual data. Very few languages have been proposed for multimedia database application [21, 28]. In [29], a temporal structure for multimedia composition is proposed. In this structure, state-

ments have been proposed to assign raw data and temporal durations to objects. However, the language does not have any provision for spatial layouts. Furthermore, only compositional aspects of the language are discussed. The querying aspects, especially content-based retrievals, are not considered. In [26], a script-based language for multimedia presentations is proposed. Both spatial and temporal aspects of presentations are addressed. However, this approach does not provide any querying facility. A spatial and symbolic query language for 3-D image data is presented in [6]. It is a special purpose language for retrieving 3-D anatomical structures. An SQL-like query language for querying medical image data has been proposed in [10]. This language uses object-oriented concepts to handle queries concerning the evolution of objects in time. Again, it is a special-purpose temporal query language.

The evolution of multimedia DBMS is shown in Figure 1. The leftmost approach is built on relational DBMS with object-oriented interface on top, and the top level is the multimedia interface. The middle approach is built on an object-oriented DBMS with extensions for multimedia data. The rightmost approach is an integrated multimedia object-oriented approach. Each approach has its own pros and cons. The language proposed by us can be used for each of these approaches.

In this paper we propose a query language which is based on predicate-logic and the notion of generalized n-ary spatio-temporal relations. The language is highly expressive and allows users to specify complex multimedia structures and generate content-based queries. The various functionalities supported by this language are listed below:

- It provides the necessary data definition functionalities.
- It allows composition of multimedia document using a Petri-Net based model.
- It supports retrieval of *image/video* data based on content and spatio-temporal characteristics.
- It provides facilities for retrieving multimedia documents, based on their spatio-temporal structures and content of component media.

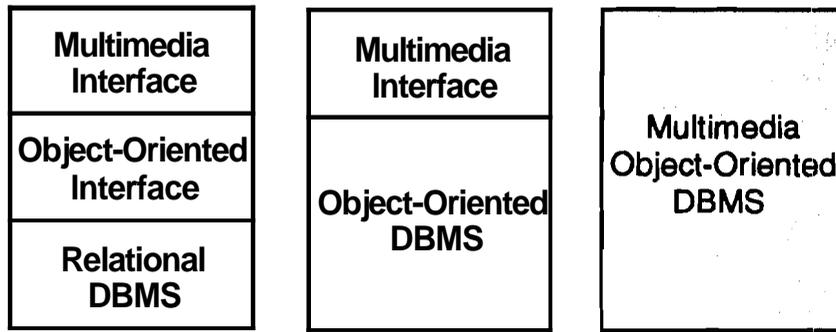


Figure 1: Evolution of design approaches of multimedia

- It supports various statements for controlling **playouts** during multimedia presentations.

The organization of this paper is as follows. Section 2 briefly reviews the underlying data model used for defining the language. Section 3 describes the main features of the language with some examples. Database architecture for processing queries based on the proposed language is presented in Section 4. The paper is concluded in Section 5.

2 Background

In this section we discuss the underlying data model used for the proposed language. The model is based on object-oriented paradigm. An object has a system-defined object identifier (*oid*), a set of attributes, and possibly various methods. *oid* of an object is unique in the sense that no two objects have the same *oid*. An attribute may contain data, meta-data, object(\sim) or reference(s) to object(s) [4, 17, 291]. Set-valued and tuple-valued (array) attributes are also allowed. A method is a function and/or a constraint applied to an object. Objects with the same attributes and methods are grouped in classes. Classes can be organized in the form of various generalization or aggregation hierarchies. Such **abstractions** can allow multiple inheritances.

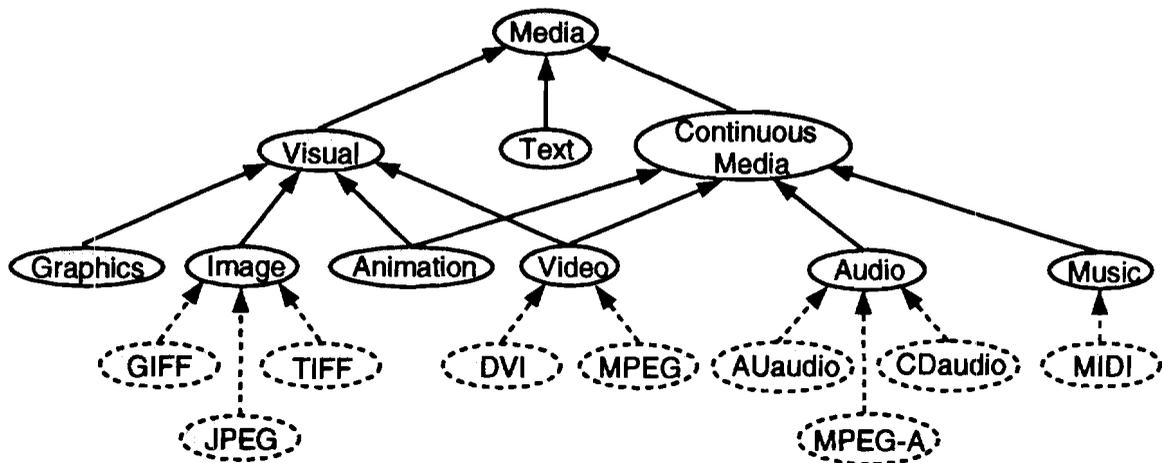


Figure 2: Media class and its subclasses

2.1 Object-Oriented Model for Multimedia Data

Various media data in a multimedia database environment can be organized in a generalization hierarchy shown in Figure 2 [13]. The *media* class, which is the super most class, has three subclasses, *Text*, *Visual* and *Continuous Media*. *Visual* class is the collection of objects having a rectangular display, while continuous media class is the collection of objects having temporal dimension. *Text*, *Image*, *Animation*, *Video*, *Audio*, and *Music* are the so-called *generic media* classes and will be used throughout the paper.

The *media* class and its subclasses not only serve as abstract data types [32], but they also control the presentations of different media in a unified and hardware/firmware independent manner as discussed in [18]. Each media is responsible for meeting intra-media synchronization requirements (if needed.) as well as for resolving hardware constraints. Each media class has attributes for specifying meta data information and an attribute called signature [17], which is an abstraction or representation of the content of the media data. Examples of signatures are the R-tree based indexing for image data and VSDG (Video Semantic Directed Graph) model [12] for video data. The formats of various allowable media are stored in the generic media class definitions. Each generic media class, as shown in Figure 2, is the root

of the hierarchy of its subclasses which are drawn in dotted ellipses. The methods are the functions to manipulate the raw data and the meta data. These include displaying the data itself, creation of indexing, etc. Each specific data type may also have a set of methods for compression/decompression.

The continuous media type has some temporal attributes. A data type of continuous media is a sequence $S = \langle e_i \rangle_n, 1 \leq i \leq n$ [25, 16]. Each element e_i has a value v_i from a domain set V , a start time π_i , and a duration τ_i , where π_i and τ_i are from time instant set and interval set. Additionally, $\pi_{i+1} = \pi_i + \tau_i, 1 \leq i < n$. For audio data, at the time of the creation of the data, this value is a number of digital samples. For video or animation, this value is an image or a graphics, respectively. Upon the creation of the object O_i , the original π_i and τ_i should be recorded.

We adopt the concept of segmentation and promotion introduced in [29]. A continuous media data object is divided into discrete units called segments, and each segment (a number of elements) is the smallest unit of data interesting to the user. The process is called promotion. Various playback speed can be supported by assigning different durations to each segment.

Some operations that can be applied to sequences (before or after promotion) [29], including the follow:

- $\text{concatenate}(S_1, S_2)$: S_2 follows S_1 . S_1 and S_2 must have the same value domain.
- $\text{subsequence}(S, i, j)$: A subsequence S' is generated by extracting from S a contiguous elements (segments) starting at i -th element (segment) and ending at j -th element (segment).
- $\text{insert}(S_1, i, S_2)$: Insert S_2 into S_1 at i -th element.

Indexing to each element of a sequence is also provided.

2.2 Generalized n-ary Relations

In order to model spatio-temporal semantics of image/video data and to formally express composition schema for multimedia document we first discuss the generalized n-ary relations,

Relation name	Symbol	constraints, $\forall i, 1 \leq i < n$
before	B	$\tau_i^e < \tau_{i+1}^s$
meets	M	$\tau_i^e = \tau_{i+1}^s$
overlaps	O	$\tau_i^s < \tau_{i+1}^s < \tau_i^e < \tau_{i+1}^e$
contains	C	$\tau_i^s < \tau_{i+1}^s < \tau_{i+1}^e < \tau_i^e$
starts	S	$\tau_i^s = \tau_{i+1}^s \wedge \tau_i^e < \tau_{i+1}^e$
completes	CO	$\tau_i^s < \tau_{i+1}^s \wedge \tau_i^e = \tau_{i+1}^e$
equals	E	$\tau_i^s = \tau_{i+1}^s \wedge \tau_i^e = \tau_{i+1}^e$

τ_s^i = starting coordinate of object τ^i , τ_e^i = ending coordinate of object τ^i

Table 1: n-ary relations

that we have proposed earlier in [14] and serves as the constructors for data model. The generalized relations are listed in Table 1 and their graphical representations are shown in Figure 3. A generalized n -ary relation can be defined as follows:

Generalized n-ary relation : A generalized n-ary relation $R(\tau^1, \dots, \tau^n)$ is a relation among n intervals, $\tau^i, i = 1, \dots, n$ which are located on a single axis with an origin and satisfy one of the conditions in Table 1 with respect to each other.

For this definition, we can notice that the same relations can be used both in space and time domains, since a one dimensional spatial axis is equivalent to the time:axis which itself is one dimensional by definition. The difference between the spatial and temporal n-ary operations is in the meaning of the operands rather than the operators.

In order to simplify the notation, we adopt the following convention to represent the generalized operators and their operands. Each interval object τ^i embodies three components and has the following format.

$$\tau^i = (oid : \ell^i : \tau_\Delta^i)$$

Here *oid* is the object id to which τ^i is associated; ℓ^i is the length of the interval τ^i , and τ_Δ^i is the inter-interval offset between interval τ^i and the next interval in the relation as shown in Figure 3. Note that τ_Δ^i is equal to zero by default for the operators meets, starts and equals and nonzero for before, overlaps, contains, and completes. Accordingly, a generalized n-ary operator R can be expressed as

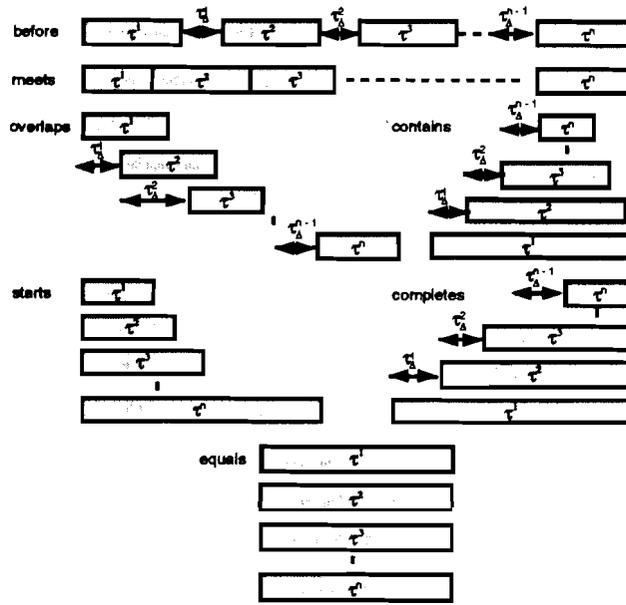


Figure 3: n-ary relations

$$R(\tau^1, \tau^2, \dots, \tau^n) \equiv R(O_1 : \ell^1 : \tau_\Delta^1, O_2 : \ell^2 : \tau_\Delta^2, \dots, O_n : \ell^n : \tau_\Delta^n)$$

for which detailed information on intervals can be provided.

The **generalized** relations are treated as classes and they can be applied to either spatial or temporal domain. They can be used to describe the temporal relationships among components of a document [11] and the relative position of media that are concurrently displayed. When applied to temporal synchronization, we have a IS-A class hierarchy, shown in Figure 4. These classes can be grouped into two super classes, namely, the sequential and parallel temporal operation classes, which correspond to the sequential and parallel nature of operations, respectively. Based on their simultaneous and non-simultaneous starting times, the parallel operations can be further divided to various classes. These classes have pointers to component objects and methods for calculating the necessary temporal information [11].

Additionally, generalized n-ary relations can be used to define spatial and temporal events in video data modeling [14]. These events allow user to specify content-based queries for the retrieval of video data. A spatial event depicts the relative positions of objects in a

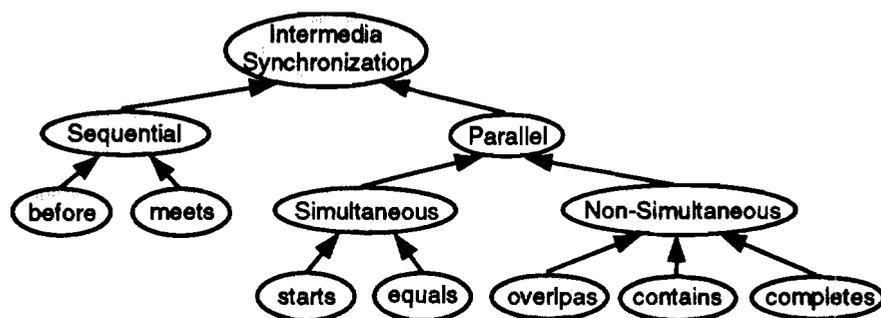


Figure 4: Inter-media synchronization class and its subclasses

frame. Formally, it is a logical expression consisting of various generalized n-ary spatial operations of projections of objects in a 3-D system. Temporal events are defined by relating spatial/temporal events using n-ary temporal operations. The formal definitions of these events are given in the next section.

The presentation control class provides the facility for starting, stopping, suspending, resuming, and repetitive playback of multimedia real-time presentation. An instance of this class is instantiated whenever a multimedia presentation is started. This class has methods for starting, terminating, suspending and resuming a presentation. Generally, the objects of presentation class receive messages from buttons [17] or sensors and perform appropriate pre-defined tasks before or during the presentations. This class supports an automatic spatio-temporal layout [1] by invoking inter-media synchronization objects and by calculating the necessary information for playback. During the presentation, an instance of this class sends playback messages with parameters to individual time-dependent media objects that are involved in the presentation at appropriate time. However, various delays such as network delays, device delays, etc, have to be taken into account. The respective playback methods of component media objects are then executed. These messages also specify the directions such as forward or backward, and speed (slow, normal, or fast) of playback. Repetitive (loop) and partial playback are supported through the methods specified within the classes.

3 The Proposed Language and Schema Declaration

The proposed language use an object-oriented modeling approach of multimedia data and is based on predicate calculus. The BNF of this language is given in Appendix A. Generally, a multimedia query can consist of one of the following four expressions; (i) data declaration declaratian for generating an object-oriented meta schema for multimedia database including declaratian of object, class, and meta-class, (ii) a composition-expression used for composing/accessing an OCPN-based document, (iii) a retrieve-expression used for content-based retrieval, and (iv) a play-expression for playing back individual multimedia object or a document. These expressions in turn consist of of sub-expressions containing object ids, logical operators AND and OR, spatio-temporal operators, and event specification. In the following section we elaborate on the main features of each type of these query expressions.

3.1 Object-Oriented Schema Declaration

Declaration/creation of multimedia objects is the first step in generating the meta-schema so that the support for our indexing mechanism of objects can be provided. As pointed out in [29], multimedia is not only presentation-driven, but also data-driven. In an object-oriented environment, a class is defined by specifying its name, superclass(es), lists of attributes and methods. Various classes in the database can form a generalization/aggregation hierarchy [12]. For example, a class X with superclasses Y and Z, attributes a_1 (domain class C_1) and a_2 (domain class C_2), and methods m_1 can be declared as :

```
class X subclass-of Y,Z
```

```
attributes
```

```
     $C_1$   $a_1$ ;
```

```
     $C_2$   $a_2$ ;
```

```
methods
```

```
     $m_1$ ;
```

Attributes and methods are treated uniformly [5, 22], i.e., they return a value. Here,

only the names of the methods are defined. Methods have signatures and they return a result class. The proposed language similar to well known programming languages like C, C++ can be used to define these methods. To instantiate an instance of a class C , we use a command called **insert** $C(\text{attribute_name} = \text{value}, \dots)$ (see Appendix A). Through this command, an instance can be assigned to a variable v by the assignment statement $v = C(\text{attribute_name} = \text{value}, \dots)$. The command **destroy** classname removes the definition of the specified class and all its instances. All the associated subclasses and their instances are also deleted.

Unary spatial operations like crop, scale, rotate, paint can be specified for displayable objects using the path expression methods. For example, y is an id of an image an object x can be formed using assignment statement $x = y.\text{crop}(0, 0, 100, 100).\text{scale}(2)$. As a result of this statement, the image y is cropped from upper left corner with 100 pixel by 100 pixel and scaled up with a factor two and the new object is x .

The command **create index** index-type on *class_name.attribute_name* creates index of type index-type on the attributename of class class-name. For an image data, such an index can, be a segmentation of image which is represented by a B-tree, R-tree, etc. For video data, it can be VSDG [12] model.

Using such a data definition approach, an object can have attributes with different media types. For example, a play class may have attributes name (text string), birth-date (text string), picture (image), interview-video (video), and interview-talk (audio). This class can have additional annotated textual attributes, if necessary. However, each media type has some built-in textual attributes, e.g., creation-date, creator, etc. To reiterate, the media attributes can be mixed with the textual attributes, and the media attributes can have built-in textual attributes. An example of classes is as follows.

class <i>text-example</i> attributes string title; text context;	class image-example attributes string classification; image body;
class graphics-example attributes string title; graphics content;	class audio-example attributes string title; audio body;
class video-example attributes string subject; video body;	class person attributes string name; image picture;

An instance of any of the classes, say video-example, can be created by using the query expression:

```
insert video-example(subject = ' Great Basketball Players', body = ' video1.mpeg').
```

Similarly, we can populate other classes by using the insert command.

3.2 Queries for Multimedia Document Retrieval

There are many interpretations of the term multimedia document. In this paper we assume that a document is a pre-orchestrated piece of information with the spatio-temporal model based on OCPN [11, 23]. For composition queries, the generalized n -ary relations in the temporal domain can be used.

As discussed earlier, a n -ary temporal relation among m objects can be specified by

$$T(o_1, \delta_1 : o_2, \delta_2 : o_3, \dots, \delta_{m-1} : o_m) \quad ,$$

where T can be B (before), O (overlaps), C (contains), CO (*completes*), M (meets), S (starts), or E (equals). Each o_i is either a single media object, or the result of a temporal operation. It can also be a surrogate variable representing a multimedia, data object promoted with its duration and layout. δ_i is the start time delay between o_{i-1} and o_i , i.e., τ_{Δ}^i shown in Figure 3. For M (meets), S (starts), and E (equals), there is no δ_i associated with o_i , $\forall i$, $2 \leq i \leq m - 1$. We can assign a temporal operation to a variable such as lecture := $O(\dots)$, so that the variable (lecture in this case) can be used elsewhere.

The OCPN model uses the binary temporal relations and can be extended by incorpo-

rating *n*-ary relations as follows. A **subnet** corresponds to a certain temporal relation (T) is replaced by a single place of type T. This **subnet** replacement process can be repeated until there is only one place left in the OCPN. Hence, the composition of the document is specified recursively, using the *n*-ary relations.

For display layout, we can describe a construct, *layout*, for the proposed language. This construct can specify the display position and dimension of single multimedia data object as follows:

layout(D_j , α_i , P_i , x_origin , y_origin , *width*, *height*)

Here D_j is the surrogate name or **object_id** where the media object α_i assigned to. P_i is the priority vector [11], which specifies the relative **foreground/background** information related to other media objects in the same document as presentation time evolves. (x_origin , y_origin) is the x-y coordinate of the upper left corner of the display area. Parameters *width* and *height* specifies the dimension of the display area. For audio data, the last four terms are replaced by ‘**channel_number**’, which specifies the output channel.

As an example of using the *n*-ary operator and *layout* command for constructing multimedia query, consider the OCPN shown in Figure 5 (a), with the timeline: shown in Figure 5 (b). The following query specifies the temporal aspect of this document.

$q_example_1 := E(M(C(A1, D2 : G1, (D1 - D2) : T2)), E(V1, A2), A3), I1, T1)$,

where $q_example_1$ is the surrogate of the document, and $A1$, $D2$, etc, represent variables for the **component** multimedia data. The composition process is elaborated as follows. $A1$, $T2$, $G1$, $D1$, and $D2$ are replaced by a place ($S1$) of type *contains*. Subsequently, $V1$ and $A2$ are substituted by a place $S2$ of class *equals*. Then, $S1$, $S2$, and $A3$ form a place $S3$ of *meets* type. Finally, $S3$, $I1$, and $T1$ are represented by a place $S4$ of type *equals* [11].

To support reusability, an OCPN can be specified using surrogates (e.g., $G1$). A surrogate can be assigned different mediaobjects, and a media object can appear in different documents or different places in a document. A surrogate needs to be assigned a **raw** media object, a duration, and relative spatial information. $q_example_1$ can be assigned different set of data

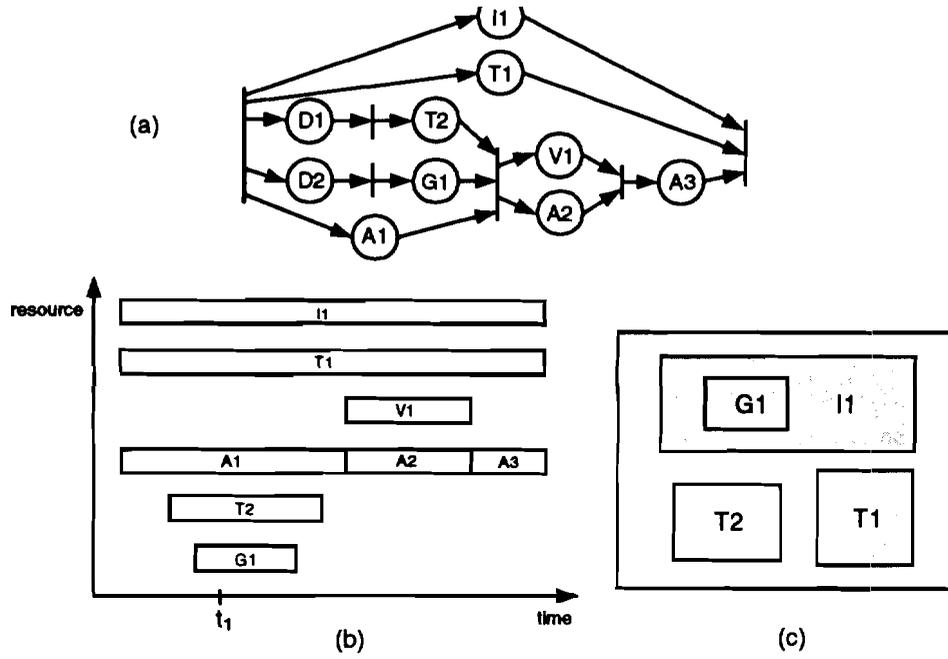


Figure 5: Example OCPN, timeline, and spatial layout at t_1

as long as these data follow the temporal specification. Note that by assigning different durations to variables (places in the OCPN) D1 and D2, A1, G1, and T2 may no longer be related to each other by the contains relation. Also, Notice that in this case, $q_example_1$ is actually a class definition of a number of documents that have similar temporal structures.

For $q_example_1$, we can now assign values to different variables and parameters as follows. For example, $G1 = (\mathcal{G}1, \tau_1)$ represents a video clip ($\mathcal{G}1$) specified below and assigned a duration τ_1 . Using the command $layout(G1, \mathcal{G}1, P_1, 20, 30, 200, 200)$, we set up the display area of the image on the screen. Similarly, we can initialize variables as follows.

$$V1 = (\mathcal{V}1, \tau_2)$$

$$layout(V1, \mathcal{V}1, P_2, 300, 300)$$

$$A1 = (\mathcal{A}1, \tau_3)$$

$$layout(A1, \mathcal{A}1, P_3, 1)$$

$$\mathcal{G}1 = \{x.content; x/graphics_example; x.subject = '...'\}$$

$$\mathcal{V}1 = (x.body; x/video_example; \exists y/person(y.name = '...' \wedge IN(y.picture, x.body)))$$

$\mathcal{A1} = \{x.body(10000 : 50000 :); x/audio_example; x.title = ' \dots' \}$

Here, τ_i is the duration in seconds. Note for variable $V1$, only the upper-left corner of the display area is specified since the frame dimension is chosen by default. For $\mathcal{A1}$, we only need to specify the audio output channel number. Note that $\mathcal{A1}$ takes part of an audio file to form a new audio object. The rest of the places in OCPN can be defined in the similar way. Notice that the various media object in the OCPN can be obtained from content-based retrieval queries mentioned in the next section.

Irrespective of the contents of multimedia documents, we can always group documents with shared semantics into a single class. These classes can form a generalization/aggregation hierarchy. For example, we can define a class called multimedia system manual with subclasses *user's manual* and *repair manual*. Each class is a collection of multimedia documents with the similar topic. Note that there can be more manual documents which are neither user's manual nor repair manuals. We can retrieve these documents based on the associated semantics.

Queries for retrieval of multimedia documents from a database can be multi-dimensional. In general, there are four possible predicates that can be specified in a document retrieval query. These predicates can be based either on spatial and/or temporal relations among the media components, or the logical structure of the document, or contents present in the component media of the document. A predicate may be a combination of any of these four dimensions. Occasionally, it is possible that a user may or may not have an a priori knowledge of the document. In that case, it may be desirable to allow the user to construct an imprecise query in order to find a document based on the above mentioned predicates. In this section we discuss the first three types of predicates. Predicates based on contents/events are discussed in the next section. The general format of this type of query can be as follows.

$x; x/D; x = (temporal_condition; spatial_condition; logical_condition; content-condition)$

The user needs to specify the search scope (D) (classes) in which the specified conditions may exist. D can be set to all, which means all the multimedia documents are searched. We can

also specify which document will not be searched by $\neg D$. The temporal condition specifies the temporal relations that exist among some component media objects during a certain period **time** of the document presentation. This condition can be easily specified by the proposed set of n-ary temporal relations. The condition represents an **imprecise/incomplete knowledge** the user perceives about the document. The spatial condition describes how the spatial **layout** looks like during the same period of time. The condition can also be specified using the n-ary spatial relations [11]. The logical-structure condition utilizes statements to delineate the possible logic-relationship of the component media objects present during the same **time** interval. Such logic-relation can be of the type similar to Hypermedia [11] model. The **query** may return the whole document or just part of the document corresponding to the specified conditions.

As an **example**, suppose there exist a number of documents in the database. Also suppose the user would like to retrieve a document that contains the spatio-temporal structure shown in Figure 5. It can be verified that the following query can specify the desired predicate.

$x; x/all;$

$\exists i/image \exists v/video \exists t/text;$

$x = (C(v, E(i, t)); (B_s(v_x, t_x) \wedge B_s(v_y, i_y));)$

3.3 Events and Content-Based Queries for Image/Video Data

Generally, most of the worldly knowledge can be expressed by describing the interplay among physical objects in the course of time and their relationship in space. Physical objects may include persons, buildings, vehicles, etc. A video database, is a typical replica of this worldly environment. In conceptual modeling of **image/video** data, it is therefore important that we identify physical objects and their relationship in time and space. For image database, several possible types of queries can be represented using the proposed language. These queries **use** the concept of spatial event which is given below [14].

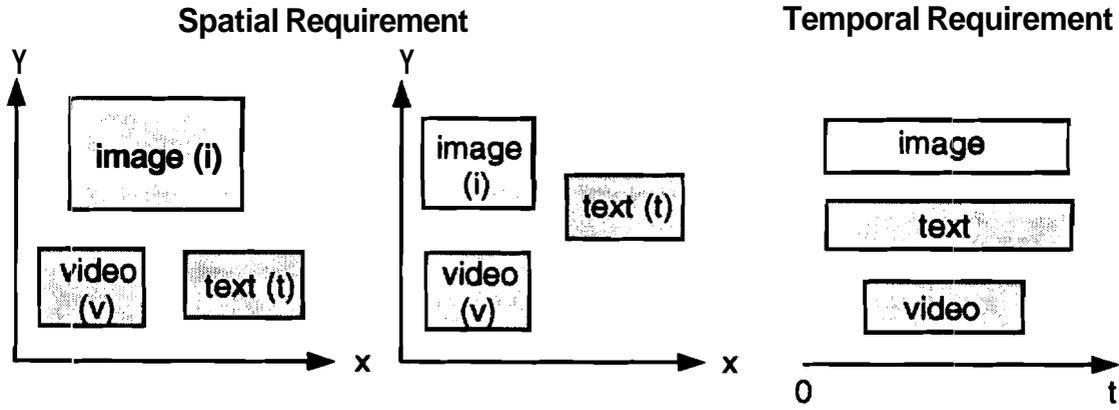


Figure 6: The user's perception of the document for imprecise query formulation

Spatial Event : A spatial event (E_s) is a logical expression involving various generalized n-ary spatio-temporal operations on positions of a group of objects. Formally,

$$E_s = R_1(\tau_1^1, \dots, \tau_1^{n_1}) \diamond_1 R_2(\tau_2^1, \dots, \tau_2^{n_2}) \diamond_2 \dots \diamond_{m-1} R_m(\tau_m^1, \dots, \tau_m^{n_m}),$$

where R_j , $j = 1, \dots, m$ is a generalized n-ary relation, \diamond_k , $k = 1, \dots, m - 1$ is one of the logical operators (A or \vee) and τ_j^i is the projection of object i in relation j on x, y , or z axis.

Similarly, for temporal event, we provide the following definition:

Temporal Event : A temporal event (E_t) is a logical expression involving; various generalized n-ary spatio-temporal operations on durations of a group of events. Formally,

$$E_t = R_1(\tau_1^1, \dots, \tau_1^{n_1}) \diamond_1 R_2(\tau_2^1, \dots, \tau_2^{n_2}) \diamond_2 \dots \diamond_{m-1} R_m(\tau_m^1, \dots, \tau_m^{n_m})$$

where τ_i^j is the interval for the j th temporal event in relation i .

As an example of a spatial event, consider a player holding the ball in a basketball game. To simplify the characterization of this situation, we assume that when the bounding rectangles of the objects player and ball are in contact with each other, that movement (or frame) marks event "player holding the ball". This particular movement is characterized by a spatial event E_s consisting of six n-ary relations between τ_x^1 (τ_y^1), the projection of the bounding rectangular associate with object player 1 on the x (y) axis and τ_x^b (τ_y^b), which is the projection of the bounding rectangular associated with the object ball on the x (y) axis.

Their event E_s is as follows:

$$E_s = (M(\tau_x^1, \tau_x^b) \vee O(\tau_x^1, \tau_x^b) \vee C(\tau_x^1, \tau_x^b) \vee S(\tau_x^1, \tau_x^b) \vee CO(\tau_x^1, \tau_x^b) \vee E(\tau_x^1, \tau_x^b)) \wedge \\ (M(\tau_y^1, \tau_y^b) \vee O(\tau_y^1, \tau_y^b) \vee C(\tau_y^1, \tau_y^b) \vee S(\tau_y^1, \tau_y^b) \vee CO(\tau_y^1, \tau_y^b) \vee E(\tau_y^1, \tau_y^b))$$

If the specified condition is satisfied for a specific frame, the event function E_s is said to be present in that frame.

Spatial events can serve as the low level (fine-grain) indexing mechanisms for video data where information contents at the frame-level are generated. The next level of video data modeling involves the temporal dimension. At the lowest level, **temporal** events are first constructed from spatial events using the above definition with a special conditioning that the n-ary operators are of type *meets* and all operands of a certain operation belong to the same spatial event. This allows us to represent the "persistence" of a specified spatial event over a sequence of frames which corresponds to a temporal event that is valid on the corresponding range of frames with duration ℓ_t . If the event starts at frame # a and ends at frame # β then $\ell_t = \beta - a + 1$. At higher levels where operands themselves are also temporal events, the duration of an *n*-ary/logical operator is the aggregate duration of its operators τ^j s, that are associated with corresponding temporal events.

An example for a temporal event consisting of two spatial events is "passing of a ball between two players". This event can be characterized by relating two similar spatial events E_s^X , "holding of the ball by player X" and E_s^Y , "holding of the ball by player Y" which can be described as in the previous section.

The pass event is composed of these events joined with two predicates. The first predicate is that both E_s^X and E_s^Y should persist for a finite duration. In other words the ball should be in contact with each player for a period of time for each event to be considered "holding". The second predicate specifies that these events should follow each other with a certain delay bounded by some specified value. The first predicate regarding persistence can be formally described as a temporal event that uses a *meets* operation with occurrence of E_s^X or E_s^Y over ℓ_t number of frames as its operands:

$$E_t^X = M(\tau_s^X, \dots, \tau_s^X) \equiv M(E_s^X : \ell_s^X : \tau_{\Delta_s}^X, \dots, E_s^X : \ell_s^X : \tau_{\Delta_s}^X)$$

$$E_t^Y = M(\tau_s^Y, \dots, \tau_s^Y) \equiv M(E_s^Y : \ell_s^Y : \tau_{\Delta_s}^Y, \dots, E_s^Y : \ell_s^Y : \tau_{\Delta_s}^Y)$$

where the number of arguments in each expression corresponds to the number of frames for which spatial events E_s^X and E_s^Y persist and are denoted by ℓ_t^X and ℓ_t^Y .

Finally, we can express the *pass* event using *before* n-ary operation between E_t^X and E_t^Y as

$$E_t^{XY} = B(\tau_t^X, \tau_t^Y) \equiv B(E_t^X : \ell_t^X : \tau_{\Delta_t}^X, E_t^Y : \ell_t^Y : \tau_{\Delta_t}^Y)$$

Here $\tau_{\Delta_t}^X$ and $\tau_{\Delta_t}^Y$ are the inter-interval offsets of the temporal events and correspond to the second predicate.

In the schema definition, we can define *spatial event class* and *temporal event class* as templates for spatial events and temporal events, respectively. The class *spatial event* has attributes for event definition (using the expressions discussed previously), for recording the clip and frame number of an instance of the event, object ids of participating objects in the event, etc. Its methods include the identification procedure which works on the signature (e.g., VSDG [12]) of video data and others. The class *temporal event* is similar to spatial event. Their definitions are shown in Table 2 and Table 3, respectively.

Each spatial event class (e.g., 'a player holding a ball') is a *specialization* of the spatial event class. Similarly, each temporal event class (e.g., 'pass') is a *specialization* of the temporal event class. A spatial event class is populated by processing the video clips in the database. In fact, whenever a video clip is archived, the identification procedures of a number of spatial event classes are applied to it. Similarly, the identification procedures of a number of temporal event classes are applied to a video clip during *archiving* stage after instances of spatial event classes have been identified. We assume that the video database stores a number of such processed video clips. Note that an instance of a *temporal* event class is always identified within the scope of a video clip. However, the *collection* of a temporal event class has instances identified across many clips.

For the purpose of object management and querying, we can maintain two views of the

CLASS <i>Generic Spatial Event</i>
ATTRIBUTE
oid objectidentifier (oid) event-definition event-definition-expression integer clip#, frame# oid list participating-object
METHOD
identification-procedure() /* class method */ (For each VSDG representation of a video clip in the video DB For each segment For each sampled frame Apply the projections (by accessing bounding volume) of the physical objects (circular nodes) defined in the event definition to evaluate the definition expression If the result of evaluation is true generate an oid for the spatial event instance record the current frame number and clip number record the partipicating physical objects' oids)

Table 2: Spatial Object

CLASS <i>Generic Temporal Event</i>
ATTRIBUTE
oid objectidentifier (oid) event-definition <i>event_definition_expression</i> BOOLEAN Spatial-Component integer clip#, <i>starting_frame#</i> , <i>ending_frame#</i> , duration oid list-of-component_event
METHOD
get-component_id() calculate-duration() (duration = <i>ending_frame#</i> - <i>starting_frame#</i> + 1) identification-procedure() /* class method */ (If Spatial_Component /* a persistent spatial event */ Search the corresponding spatial event class collection If there exist spatial event instances from frame a to frame b in a clip c Generate an oid for an instance of the temporal event starting-frame# = a, <i>ending_frame#</i> = b, clip# = c else For each clip For each term (a n-ary relation) Find all events (<i>e_{first}s</i>) corresponding to τ_1 For each <i>e_{first}</i> Identify components τ_2 to τ_n If all are identified If τ_i s satisfy the temporal relation defined Generate an oid of this instance of the temporal event Calculate and record necessary information of this instance Logically combine all the terms))

Table 3: Temporal event

video data. First, *image/video* are stored in the conventional way as instances of classes. A number of classes can exist in the database system. Each class is a collection of *images/video* clips based on certain criteria given by the users. For example, a class called 'Michael Jordan' is defined for all video clips in our video database with topic related to Michael Jordan. These classes can form a generalization/aggregation hierarchy. The classes in Section 3.1 belong to this category. On the other hand, there are *spatial/temporal* event classes existing in the system. A *spatial/temporal* event class can be related to another class not belonging to the previous category by the object-oriented abstractions like generalization, aggregation. For handling queries targeted for a video clip, we need a dictionary to record all the *spatial/temporal* event instances identified within a clip with necessary information.

For an image database, the following types of queries can be posed and are supported by the proposed language.

- Existence of physical objects. A system-defined function $IN(x, image.id)$ returns **TRUE** if the physical object represented by the variable x appears within the image represented by $image-id$.
- The system can be queried to return images which satisfy some spatial relations specified either in the form of n -ary generalized relations or some parameterized functions defined by the users. These users-defined functions are in turn can be expressed through the generalized n -ary relations. For example, the proposed function $ABOVE(x, y)$ can either return True or False value depending upon whether or not x is above y . Although this can be a spatial event, but due to its extensive use, it can be coded as a function.
- a Existence of spatial events. An image database can be searched for the existence a certain type of spatial event.

For video data, the above three types of queries can be applied at the frame level. Since video data has the temporal dimension, queries involving temporal events should be sup-

ported, which is also the features of the proposed language.

As the first example, consider the following content-based query for an image database, "Find an image where a person is to the right of a vehicle". The following expression constructs this query:

$x; x/image_example; \exists y/person \exists z/vehicle(B_s(z_x, y))$

The subscript s associated with operator B indicates that it is a spatial operation. Variable subscript i represent the bounding volume projection of the object on the i axis [12]. This query operates on the collection of a class of image data called *image_example*.

Next, we give an example of a content-based query for video database using the proposed language. Suppose we are using a sport video database with temporal event class pass (identifying the 'pass' event). A query to identify the video clip numbers of video data that contains "Michael Jordan passes a ball to Scotty Pippen" can be expressed. as:

$v.clip\#; v/pass; v.player_name_1 = 'MichaelJordan' \wedge v.player_name_2 = 'ScottPippen'$.

Here we assume that in the class definition of pass, two attributes *player_name_1* and *player_name_2* are defined for the players involved in passing and receiving the ball, respectively. For this query to be processed, we need to define a temporal event class called pass [14].

An example of a query aimed at checking the presence of objects in a video clips is given next. Suppose a class *player* is defined as the subclass of the class person. Similarly, suppose a subclass *NBA_video* of video-example class is also defined. Then, a query to check the existence of Michael Jordan can be expressed as:

$v; v/NBA_video; \exists x/player(x.name = 'Michael Jordan' \wedge IN(x, v))$

As a result of this query, multiple clips may be returned. This query is another example of querying on the collection of a class called 'NBA_video'.

The following query can be used to return all the video clips in the class video-example such that these temporal events e_1 (an instance of temporal event class t_{e1}), e_2 (an instance of temporal event class t_{e2}), and e_3 (an instance of temporal event class t_{e1}) appear in the

clip according to the temporal relation **B** (before) (see Gihure 3).

$v; v/video_example; \exists e_1, e_2/t.e_1 \exists e_3/t.e_2 (B(e_1, (10 : 30) : e_3, (0 : 50) : e_2))$

The tuple (10:30) represents the range of the interval between events e_1 and e_2 . Similarly, the tuple (0:50) represents the range of the interval between events e_3 and e_2 .

3.4 Queries for Playout Control

In the proposed language, we differentiate between the playback of a mono media and the playback of a complete document. The reason is that in some cases, a user would like to browse through the database prior to composing a document. The first type of query allows such browsing by retrieving a mono media as an attribute of an instance of a class. Such a query can be expressed as follows:

$x.multimedia_attribute.play_method(t_1 : t_2 : s); x/C; \text{qualification clause}$

Here x is a variable ranged over all the instances (collection) of a class. *multimedia_attribute* is the name of the attribute of the mono media to be played, and *play_method* is the system-defined method for displaying that media. For text, graphics, and image data, the appropriate methods are *display*, *draw*, and *display*, respectively. For audio and video data, the methods are *play-forward*, **play_forward* (repetitive play forward), *play_backward*, and **play_backward* (repetitive play backward). Parameters t_1 , t_2 , and s represent the beginning, ending, and playback rate of audio and video data. If s is omitted, a pre-defined standard playback rate can be used. For audio, the parameters ($t_1 : t_2$) can take one of the following forms.

(1) ($seg\# = n_1 : seg\# = n_2$). Here n_1 and n_2 represent the beginning and the end segment numbers of the audio clip, respectively. If t_1 is not specified, it means playing the audio data from the first segment. Similarly, if t_2 is unspecified, the playback can be started from segment n_2 and continue to the last segment.

(2) ($a : ?$). The playback starts from a and continues to β , where a and β are the time instants represented in seconds. Either a or β can be omitted as mentioned in (1).

(3) ($e : \mathbf{mid} \pm S$), ($\mathbf{mid} \pm \delta : e$), or ($\mathbf{mid} \pm \delta_1 : \mathbf{mid} \pm \delta_2$). Here \mathbf{mid} is the middle of the audio segment and ε represents the starting or ending point of the playback. ε takes the same format as the t_i in (1) and (2). δ and δ_i are offsets which must be compatible with ε .

(4) ($\mathbf{dur} \cdot \delta$): \mathbf{dur} represents the duration of the data, where as δ is an offset in seconds or segments. The playback is from $\mathbf{dur} - \delta$ to the end of the data.

For video, the same convention is adopted in the proposed language except that in (1), $\mathbf{seg\#}$ is replaced by $\mathbf{f\#}$, and if $n_1 = n_2$, a single frame is displayed.

For the playout of a complete document, the following query can be used.

P.play_method(t₁ : t₂ : s)

\mathbf{P} is either a variable that represents the label of a temporal event associated with a complete video clip, e.g., lecture, or it can be a temporal operation (e.g., $\mathbf{M}(\dots)$). *play_method* can be either *play*, *rplay*, *play (repetitive play), or *rplay (repetitive reverse play). Parameters ($t_1:t_2:s$) have the same format as *audio/video* except that *seg#* and *f#* are not allowed.

We can specify the type of interactions that are either allowed or prohibited. Also, we can suppress the playback of some component media of the document. Note that the latter technique is needed in those situations where some display devices are not available.

Some example queries based on the proposed language for various playback scenarios are given below.

- Play the whole presentation *q_example₁* at the regular speed where no user interaction is allowed.

q_example₁.play A \neg Interaction

- Partial reverse playback of representation *q_example₁* from the end to the instant 30 seconds into the presentation. Also playback at twice the regular speed.

q_example₁.rplay(: 30 : 2)

- Play a single frame of a video clip.

x.body.play_forward(100 : 100 :); x/video.example; x.title = '...'

- Play the middle 20 seconds of *q_example₁* at a regular speed.

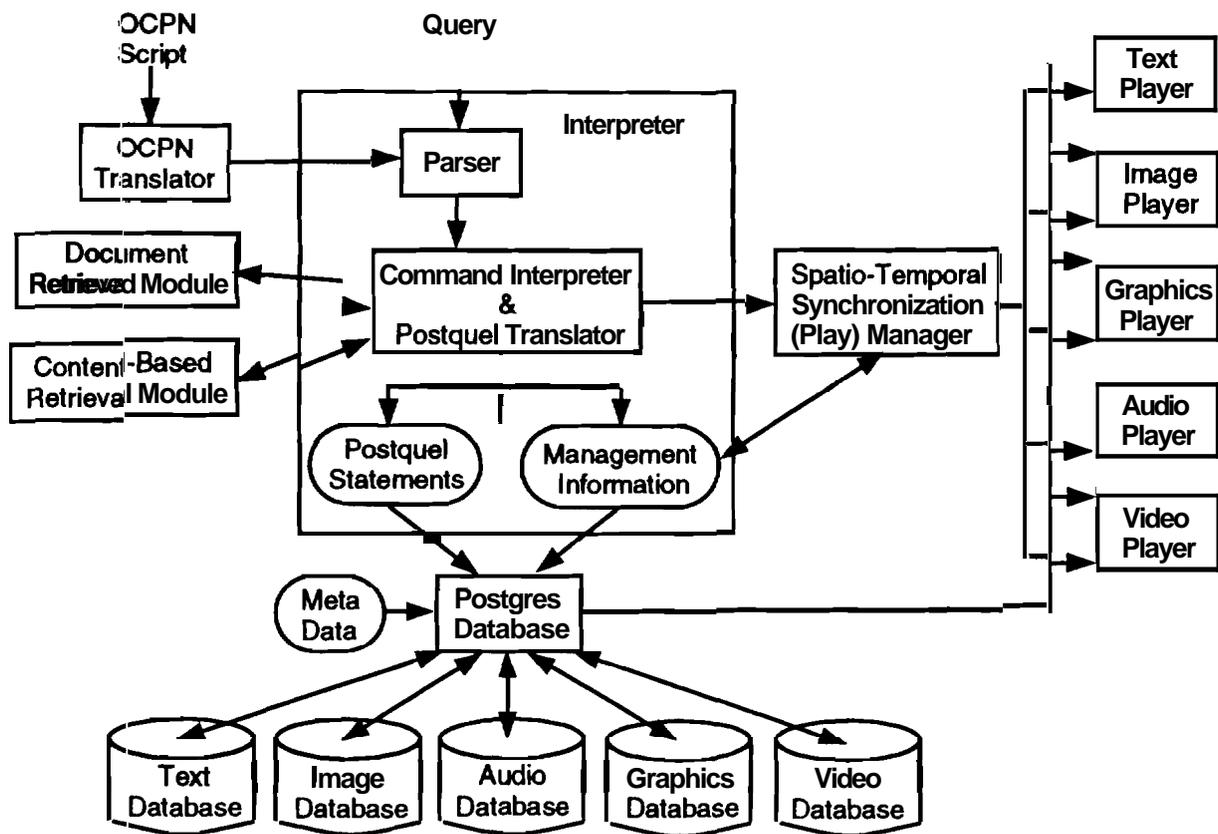


Figure 7: The architecture of the system

q_example1.play(mid - 10 : mid + 10 :)

- Play the: last 20 seconds of a video clip.

x.body.play_forward(dur - 20 ::); x/video_example; x.title = '...'

4 System Architecture

To complete our discussion, we present a database architecture for processing multimedia queries based on the the proposed language. Such an architecture is shown. in Figure 7 and is currently under development using object-oriented database techniques and the Postgres DBMS [31].

First, a query is processed by the parser, which decomposes the query into a format understood by the command interpreter & Postquel translator. This translittor converts the

parsed query into Postquel statements [27] (whenever possible) and/or generates temporal management information (e.g., playback table storing the information about a document), which is stored in the Postgres database [31]. The command interpreter also communicates with three other modules. The *content-based retrieval module* is responsible for image/video retrieval based on spatio-temporal characteristics; the *document retrieval module* maintains meta schema for documents; and finally the *spatio-temporal synchronization (play) manager* receives commands from the command interpreter and plays out multimedia documents or individual media data objects. After receiving a command for document playback, the manager retrieves the playback information from the management information. The manager then issues commands to individual media players (e.g. *video player*) which retrieve the raw data from the Postgres Database and playback the data at appropriate times. The document retrieval module communicates with the management information and content-based retrieval module to perform matching. The system also accepts OCPN scripts and composes corresponding queries using the algorithm given in [13]). Alternately, it translates the script; into an internal format for the management information. Meta-data information like definitions of media data (Appendix B) are also stored in the Postgres DBMS.

We have implemented a subset of our language, including data definition, spatio-temporal synchronization statements, and forward play statements using C and Postquel. Efficient algorithms for matching document structures are currently being developed.

Postgres, which acts as the underlying database engine, provides the necessary utilities to define new data types and operations on them.

5 Conclusion

In this paper, we have proposed a query language based on predicate calculus and generalized n-ary relations. Using an object-oriented approach, we have provided a formal BNF for the **grammar** of this language. The language not only allows specification of spatio-temporal composition of multimedia documents but can also be used to express complex spatio-temporal events related to video data. The latter functionality allows efficient content-based retrievals.

Appendix A. BNF for the Grammar for the Proposed Language

query ::= **basic_expression** | compositionxpression | retrieve-expression | **play_expression**

basic_expression ::= class-definition | **insert_expression** | destroy class-name

class-definition ::= class class-name [subclass-of class-name {, class-name}]

attributes class-name attribute-name; (class-name attribute-name; }

[methods method-name; (method-name;]]

insert-expression ::= [variable=] insert class-name

(attribute-name = value; (attribute-name = value; }

compositionxpression ::= **temporal_expression** | **layout_expression** | duration-expression

temporal_expression ::= [variable=] **spatial_temporal_op**

spatial_temporal_op ::= sp_op₁(operand_list₁) | sp_op₂(operand_list₂)

sp_op₁ ::= **B**_[s] | **O**_[s] | **C**_[s] | **CO**_[s]

sp_op₂ ::= **M**_[s] | **S**_[s] | **E**_[s]

operand_list₁ ::= spsurrogate, sp_surrogate_list

spsurrogatedist ::= [δ :]sp_surrogate | [δ :]sp_surrogate,sp_surrogate_list

operand_list₂ ::= spsurrogate | sp_surrogate, operand_list₂

spsurrogate ::= spatial-temporal~ $\#$ variable_expression

layout_expression ::= layout(*document_surrogate,object_surrogate,priority_vector,loc,[dimension]*)

dimension ::= width,height

$loc ::= (x.origin, y.origin) \mid \text{channel-num}$
 $\text{duration-expression} ::= \text{document-surrogate} = (object_surrogate, \tau)$
 $\text{retrieve-expression} ::= [\text{variable}=] \text{target_expression}; \text{range-expression}; \text{qualification_expression};$
 $\quad [object_surrogate = \text{target_item}(value)]$
 $\text{target_expression} ::= \text{target_item} \mid \text{target_item}, \text{target_expression}$
 $\text{target_item} ::= [\text{variable} \mid \text{variable_expression}][(\text{interval})]$
 $\text{variable-expression} ::= \text{variable.attribute_method_list}$
 $\text{attribute-methodlist} ::= \text{attribute_method_item} \mid \text{attribute_method_item.attribute_method_list}$
 $\text{attribute_method_item} ::= \text{attribute-name} \mid \text{method-name}$
 $\text{range-expression} ::= \text{range_item} \mid \text{range_item}, \text{range-expression}$
 $\text{range_item} ::= \text{variable} / [\text{all} \mid [\neg] \text{class-name}]$
 $\text{qualification_expression} ::= \{ \subseteq \text{range-expression} \} \text{qualification-term}$
 $\subseteq ::= \forall \mid \exists$
 $\text{qualification_term} ::= \text{predicate} \mid \text{predicate boolean-connector qualification-term}$
 $\text{boolean_connector} ::= \wedge \mid \vee \mid \neg$
 $\text{predicate} ::= \text{TRUE} \mid \text{FALSE} \mid \text{simple-condition} \mid \text{media-condition} \mid \text{document-condition}$
 $\text{simple-condition} ::= \text{item comparison-operator item}$
 $\text{item} ::= \text{constant} \mid \text{literal} \mid \text{variable} \mid \text{variable-expression}$
 $\text{comparison-operator} ::= = \mid != \mid > \mid < \mid >= \mid <=$
 $\text{media-condition} ::= \text{IN}(variable, object_surrogate) \mid \text{function_name}(parameter_list) \mid \text{spatial_temporal_op}$
 $\text{document.condition} ::= \text{variable} = [\text{temporal-condition};] [\text{spatial-condition};] [\text{media-condition};]$
 $\text{temporal_condition} ::= \{ \text{spatial_temporal_op} \}$
 $\text{spatial-condition} ::= \{ \text{spatial_temporal_op} \}$
 $\text{play-expression} ::= \text{sp_surrogate.play_method} [(\text{interval})] [\text{play-condition}]$
 $\quad [;\text{range_expression};\text{qualification_expression}]$
 $\text{play_method} ::= \text{play} \mid \text{rplay} \mid +\text{play} \mid * \text{rplay}$
 $\text{interval} ::= [\text{begin-end}]:[\text{speed}]$

begin-end ::= [point] :[point]

point ::= **seg#** = value | **f#** = value | value | **mid** ± value | **dur** - value | null

Appendix B. Postquel Definition of Some Data Types

define type **longtext** (input = lofilein, output = lo_fileout, internallength = variable)

```
define function textwin (language = "c", returntype = int4)
    arg is (longtext)
    as "/home/audio/a/postgres/rawdata/text/textwin.o"
define function textkill (language = "c", returntype = int4)
    arg is (int4)
    as "/home/audio/a/postgres/rawdata/text/textkill.o"
```

define type **image** (input = lofilein, output = lo_fileout, internallength = variable)

```
define function implay (language = "c", returntype = int4)
    arg is (image)
    as "/home/audio/a/postgres/rawdata/images/implay.o"
define function imkill (language = "c", returntype = int4)
    arg is (int4)
    as "/home/audio/a/postgres/rawdata/images/imkill.o"
```

define type **audio** (input = lofilein, output = lofileout, internallength = variable)

```
define function auplay (language = "c", returntype = int4)
    arg is (audio)
    as "/home/audio/a/postgres/rawdata/sounds/auplay.o"
define function aukill (language = "c", returntype = int4)
    arg is (int4)
    as "/home/audio/a/postgres/rawdata/sounds/aukill.o"
```

define type **video** (input = lofilein, output = lofileout, internallength = variable)

```
define function videoplay (language = "c", returntype = int4)
    arg is (audio)
    as "/home/audio/a/postgres/rawdata/video/videoplay.o"
define function videokill (language = "c", returntype = int4)
    arg is (int4)
    as "/home/audio/a/postgres/rawdata/video/videokill.o"
```

References

- [1] Philipp Ackermann, "Direct Manipulation of Temporal Structures in a Multimedia Application framework," Proc. ACM Multimedia 94, pp. 51-58.
- [2] Reda Alhajj and M. Erol Arkun, "A Query Model for Object-Oriented Databases," Proc. 1993 IEEE 9th International Conference on Data Engineering, pp. 163-172.

- [3] P. Atzeni, L. Cabibbo, and G. Mecca, "IsaLog: A Declarative Language for Complex Objects with Hierarchies," Proc. 1993 IEEE 9th International Conference on Data Engineering, pp. 219-228.
- [4] Jay Banejee, etl., "Data Model Issues for Object-Oriented Applications," ACM Trans. on *Office* Information Systems, Vol. 5, No. 1, January 1987, pp. 3-26.
- [5] Jay Banerjee, Won Kim, and Kyung-Chang Kim, "Queries in Object-Oriented Databases," Proc. 1988 IEEE 4th International Conference on Data Engineering, pp. 31-31'.
- [6] Dan Benson and Greg Zick, "Spatial and Symbolic Queries for 3-D image data," SPIE Vol. 1662 Image Storage and Retrieval Systems (1992), pp. 134-145.
- [7] Elisa Bertino, Mauro Negri, Giuseppe Pelagatti, and Licia Sbattella, "Object-Oriented Query Languages: The Notion and the Issues," IEEE Trans. on Knowledge and Data Engineering, Vol. 4, No. 3, June 1992, pp. 223-237.
- [8] Gerold Blakowski, Jens Hubel, Ulrike Langrehr, and Max Muhlhauser, "Tool support for the synchronization and presentation of distributed multimedia," Computer Communications, Vol. 15, No. 10, December 1992, pp. 611-618.
- [9] R. H. Campbell and A. N. Haberman, "The Specification of Process Synchronization by Path Expressions," Lecture Notes in Computer Sciences, No. 16, Operating Systems, Springer-Verlag, Berlin (1974), pp. 89-102.
- [10] W. W. Chu, I. T. Jeong, R. K. Taira, C. M. Breant, "A Temporal Evolutionary Object-oriented Data Model and Its Query language for Medical Image Management," Proc. of the 18th VLDB Conference, Vancouver, Canada 1992.
- [11] M. Iino, Y. F. Day, A. Ghafoor, "An Object-Oriented Model for Spatio-Temporal Synchronization of Multimedia Information," Proc. of IEEE ICMCS' 94, International Conference on Multimedia Computing and Systems, pp. 110-119.
- [12] Y. F. Day, S. D. Dagtas, M. Iino, A. Khokhar, and A. Ghafoor, "Object-Oriented Conceptual Modeling of Video Data," Proc. of IEEE International Conference on Data Engineering '95, Taipei, Taiwan, pp. 401-408.
- [13] Young Francis Day, "Semantic-Based Object-Oriented Modeling of Multimedia Data," PhD Thesis Proposal, Purdue Unviversity, March 20, 1995.
- [14] Y. F. Day, S. D. Dagtas, M. Iino, A. Khokhar, and A. Ghafoor, "Spatio-Temporal Modeling of Video Data for On-line Object-Oriented Query Processing," IEEE ICMCS' 95 *Proceeding*, pp. 98-105.
- [15] Simon Gibbs, "Composite Multimedia and Active Objects," Proc. OOPSLA '91, pp. 97-112.
- [16] Simon Gibbs, Christian Breiteneder, and Dennis Tsichritzis, "Audio/Video Databases: An O'bject-Oriented Approach," Proc. 1993 IEEE 9th International Conference on Data Engineering, pp. 381-390.

- [17] Sha Guo, Wei Sun, Yi Deng, Wei Li, Qing Liu, and Weiping Zhang, "Panther: An Inexpensive and Integrated Multimedia Environment", Proc. of *IEEE ICMCS' 94, International Conference on Multimedia Computing and Systems*, pp. 382-391.
- [18] I. Herman, G. J. Reynolds, and J. Davy, "MADE: A Multimedia Application Development Environment," Proc. of *IEEE ICMCS' 94, International Conference on Multimedia Computing and Systems*, pp. 184-193.
- [19] Petrit Hoepner, "Synchronizing the presentation of multimedia objects," *Computer Communications*, Vol. 15, No. 9, November 1992, pp. 557-564.
- [20] H. Ishikawa, etl., "The Model, Language, and Implementation of an Object-Oriented Multimedia Knowledge Base Management System," *ACM Trans. on Database Systems*, Vol. 18, No. 1, March 1993, pp.1-50.
- [21] Thomas Joseph, Alfonso F. Cardenas, "PICQUERY : A High Level Query Language for Pictorial Database Management," *IEEE Trans. on Software Engineering*, Vol. 14, No. 5, May 1988, pp. 630-638.
- [22] Michael Kifer, Won Kim, and Yehoshua Sagiv, "Querying Object-Oriented Databases," Proc. *ACM SIGMOD Management of Data Conference*, June 1992, pp. 393-402.
- [23] T. D. C. Little and A. Ghafoor, "Synchronization and Storage Models for Multimedia Objects," *IEEE Journal on Selected Areas In Communications*, Vol. 8, No. 3, April 1990, pp. 413-417.
- [24] T. D. C. Little, "Synchronization for Distributed Multimedia Database Systems," *Ph.D. dissertation*, Syracuse Univ., August 1991.
- [25] T. D. C. Little and A. Ghafoor, "Interval-Based Conceptual Models for Time-dependent Multimedia Data," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 5, No. 4, August 1993, pp. 551-563.
- [26] S. R. L. Meira and A. E. L. Moura, "A Scripting Language for Multimedia Presentations," Proc. of *IEEE ICMCS' 94, International Conference on Multimedia Computing and Systems*, pp. 484-489.
- [27] Postgres group, *Postquel User Reference Manual*, 1993.
- [28] Roussopoulos, N.; C. Faloutsos; et al, "An Efficient Pictorial Database System for PSQL," *IEEE Trans. on Software Engineering*, Vol. 14, No. 5, May 1988, pp. 639-650.
- [29] G. A. Schloss and M. J. Wynblatt, "Building Temporal Structures in a Layered Multimedia Data Model," Proc. *ACM Multimedia '94*, pp. 271-278.
- [30] Gail M. Shaw, and Stanley B. Zdonik, "A Query Algebra for Object-Oriented Databases," Proc. *1990 IEEE 6th International Conference on Data Engineering*, pp. 154-162.
- [31] Michael Stonebraker, Lawrence A. Rowe, "The Design of Postgres," Proc. *ACM SIGMOD Conf. Management of Data*, May 1986, pp. 340-355.
- [32] Michael Stonebraker and Michael Olson, "Large Object Support in POSTGRES," Proc. *1993 IEEE 9th International Conference on Data Engineering*, pp. 355-362.