A Multimedia Query Language for Handling Multi-Structure Information

Eunsook Ryu and Kyuchul Lee
Department of Computer Engineering
Chungnam National University
Taejon, KOREA 305-764
{esryu,kclee} @comeng.chungnam.ac.kr

Abstract

Compared with conventional query languages, more factors should be considered in developing multimedia query languages because multimedia information is composed of large unformatted media data and also has multiple complex structures such as hyperlink networks and spatial/temporal layout structures, as well as logical structures. In this paper, we propose an object-oriented multimedia query language for handling multi-structure information. The query language is specified in compliance with the SQL3 standard, and the type hierarchy and query expressions are enriched to support various query types (i.e., attribute, content, structure, hyperlink queries) and all the structures (i.e., logical, spatial/temporal layout, hyperlink network structures).

Keywords multimedia databases, object-oriented databases, query languages, hypertext.

1 Introduction

Multimedia database systems have received a lot of attention with the advent of newly emerging applications such as Office Information Systems, Digital Library, and World-Wide Web (WWW).

Compared with relational and object-oriented query languages, the following factors should be considered in developing multimedia query languages.

- New data types should be supported for storing and manipulating unformatted large-volume multimedia data. Most of DBMSs provide the blob (binary large object) data type. However, few DBMSs implement content-based retrieval operations and updating functions for various multimedia data.

- Multimedia information is usually delivered in a certain document format. There are several multimedia document standards such as SGML (Standard Generalized Markup Language) [11], HyTime (Hypermedia/Time-based Structuring Language) [12], ODA (Office Document Architecture) [10]. It is difficult to model and manage multimedia documents simply using record-based data models such as relational model because document structure information is quite complex [5]. Therefore, most of document standards based on object-oriented paradigm.

- Most previous studies on querying structured document [2, 7, 17, 18] have focused on only the logical structure of text-oriented documents. In querying multimedia information, more structures have to be considered. An ideal multimedia document representation must allow multiple structures including the layout (or physical) structure, as well as the logical structure. The layout structure specifies the layout of the components actually presented to the user on output devices, while the logical structure represents the hierarchical structure between logical elements such as title, sections, and paragraphs. Moreover, there can be more than one layout structure according to screen configuration or paper size at the presentation time.

- The layout structure can be further classified into the spatial layout structure and the temporal layout structure. The temporal layout structure is not required in text-oriented documents. However, it is essential for multimedia documents to present the components of a document in a coherent, synchronized manner according to the author's intention. For instance, audio must synchronously accompany the image presented in slide presentations. In case of HyTime documents, this temporal structure can be encoded in the logical structure [12].

- Related multimedia information can be interlinked to each other through hyperlinks. Mul-
timedia query languages should support the navigation of the hyperlink networks. It is more desirable that users can specify not only the navigation of one or more hops through the hyperlinks but also forward/backward traversal.

In this paper, we propose an object-oriented query language for defining and retrieving multimedia information which has multiple structures such as hyperlink networks and spatial/temporal layout structures, as well as logical structures.

This paper is organized as follows. In Section 2, a typical example of the multimedia information and its characteristics are described. The type hierarchy for multi-structural multimedia information is proposed based on the object-oriented data model in Section 3. Multimedia data types supported in this paper is described in Section 4. The query language and query examples are presented in Section 5. In Section 6, we compare our approach with previous work. Finally, Section 7 presents the conclusions and future work.

2 Multi-Structural Characteristic of Multimedia Information

This section gives a typical example of a multi-structured multimedia document and explains characteristics of each document structure.

2.1 Logical and Spatial Layout Structure

A typical multimedia document is composed of several logical elements, and is presented according to a spatial layout (Figure 1).

Figure 1: Logical elements and a spatial layout of a multimedia document

The logical structure, which forms a hierarchical representation of a document, represents how to construct a document by a set of logical elements such as titles, sections, and paragraphs. Figure 1 shows that the document consists of text portions such as a title, an author, an abstract, two sections, and several paragraphs. It can include other multimedia data such as images, video, audio, and graphics. The logical structure makes it easy for the reader to understand the contents. Therefore, most of the document standards (e.g., SGML, ODA) provide schemes for representing the logical structure.

The spatial layout structure describes the hierarchical aggregation of presentation components of a document. The contents of the document in Figure 1 is presented in two pages. Each page is composed of rectangular areas called frames, and they can be further divided into the lowest-level areas known as blocks. We use the ODA terminologies to represent spatial layout structure elements. However, it does not mean that our scheme is dependent on a specific document model. Spatial layout structure elements may vary according to applications.

Figure 2 represents composite objects for the logical and spatial layout structures of the document 'mml' in Figure 1. In Figure 2, the title, the author, an abstract, and two sections are aggregated to form a composite object 'mml' in the logical structure. The spatial layout structure information can be also modeled as a composite object. There is a part-of relationship between a child node and a parent node. In each composite object, the document itself is the root of the hierarchy, and only leaf nodes contain text or multimedia data.

Figure 2: Composite objects for logical and spatial layout structures

The contents of a leaf logical object does not always correspond to the contents of a layout block. The logical object section[1].para[2] is divided into two portions (i.e. section[1].para[2]' and section[1].para[2]"") and associated with two separate blocks, that is page[1].frame[2].block[1] and page[2].frame[1].block[1], belonging to two different pages as shown in Figure 1 and 2. On the other hand, several logical objects (i.e., image[1], image[2]) can fit into one layout block page[2].frame[2].block[1] in the spatial layout structure.
2.2 Temporal Layout Structure

In order to present two images and a video clip synchronously in page[2] of the document in Figure 1, we need another structure, called temporal layout structure. Let us assume that the author's intention of the presenting schedule of frame[2] and frame[3] in page[2] is shown in Figure 3(a). Note that the logical object video[1] is divided into three fragments (i.e., video[1'] , video[1''] , and video[1''']) which have the same presentation time, with two images and a 60-seconds delay between them.

Figure 3: (a) Example presentation schedule (b) A composite object for the temporal layout structure

To represent the temporal layout structure, we use the temporal relationship specification scheme developed by the second author of this paper [14]. Allen [1] has defined 13 temporal relations between two time intervals. In [14], the delay object, a special kind of object representing specified delay, is introduced to represent Allen's 13 temporal relations simply as a composition hierarchy of two types, namely the parallel (par) and sequential (seq) relationships, without any loss of semantics. For example, Allen's 'finishes(A,R)' relation can be interpreted as a parallel relationship between two objects. That is, one object is a sequential composition of a delay object d and an object A and the other is an object B. Therefore, this relation can be represented in as 'par(seq(d,A), B)'.

Figure 3(b) represents a composite object hierarchy for the presentation schedule in Figure 3(a). The 60-seconds delay between image blocks is treated as an object, and has a parallel relationship with the video[1''] fragment.

2.3 Hyperlink Network

One more structure, which should be considered, is the hyperlink network. It is formed by interlinking related multimedia information. For example, the document K is directly referred by the objects C and I, and indirectly by object E in Figure 4. Users can freely navigate in either direction and select any interested document or a group of documents.

Figure 4: Example composite object and hyperlink network

3 Defining Multi-Structure Information

Object-oriented DBMS technology can be used to define composite objects for each structure information described in Section 2. In this section, the type hierarchy for multi-structural multimedia information is designed based on the object-oriented data model.

3.1 Type Definition for Multi-Structure Information

Figure 5 illustrates the type hierarchy designed for multi structure information. There are three abstract subtypes of the type Doc_Structure - the first subtype Doc_Type modeling the different kinds of documents (e.g., memos, multimedia articles), the second subtype Structure_Type modeling the categories of structures (e.g., logical, spatial layout, temporal layout), and the third subtype Element modeling all the elements of three structures. The structure of a document is defined as a subtype of two subtypes of Doc_Type and Structure_Type. For example, the type Logical.MM_Article representing the logical structure of MM_Article has two supertypes MM_Article and Logical.

Figure 5: The type hierarchy for multimedia document structures

Every element of these structures is represented by a type. For instance, the section element in the logical structure is represented by the type Section, and the page element in the spatial layout structure is modeled as the type Page. Each element type, according to the structural aspects is classified into three categories as shown in Figure 5. The type Link is defined as a subtype of the type Logical.Element because an anchor of a link is set based on a portion of document contents.
Figure 6 presents the type definitions of Doc_Type and Logical.MM_Article corresponding to the logical elements in Figure 1. The type definition is specified in compliance with the SQL3 [8] standard. SQL3 has been developed to incorporate the advantages of the relational and the object-oriented database systems. The reason why we adopt SQL3 as the basis of our multimedia query language is to be independent of a specific DBMS implementation. Our query language can be easily implemented using any DBMS which follows the SQL3 standard.

```sql
CREATE OBJECT TYPE Doc_Type
( PROTECTED doc_id char(5),
text_content TEXT);

CREATE OBJECT TYPE Logical.MM_Article
UNDER MM_Article, Logical
( PUBLIC rev_date date,
title TextFragment,
authors LIST(TextFragment),
abstract TextFragment,
sections LIST(Section) );
```

Figure 6: Example type definitions

In the definition of the type Doc_Type, the attribute doc_id is declared for identifying a unique document, and the attribute text_content has an object identifier of the text object containing the entire text content of a document. These attributes are defined as PROTECTED mode, and are inherited by its subtypes.

In the definition of Logical.MM_Article, the attribute rev_date is a descriptive attribute, while the other four attributes — title, authors, abstract, and sections — represent the logical structure elements. The attribute authors has LIST of TextFragment as its domain. The attribute sections is declared based on LIST of Section. The type construct LIST supports a set of ordered values.

3.2 Efficient Retrieval of Structure Elements

In order to support efficient retrieval of structure elements, we have to maintain the hierarchical information about the composite object for each structure. Previously, database relations [6], parse trees [4], and element locators [2] have been used. However, they require considerable disk space and access time.

In this paper, we propose a new scheme which uses specially designed Hierarchical Element Identifiers (HIDs) for structure element access. By letting the HID carry information about the document structure, the HIDs of the ancestor or descendant nodes of a node can be obtained directly from the HID. A HID of a node in a composite hierarchy is generated by concatenation of the children identifiers in the path from the root to the node. For example, the HID of the node which is the second block of the first frame of the second page in the spatial layout structure in Figure 2 is '212'. By using it, the HIDs of the ancestor nodes at the i-th level backward can be obtained by shifting the HID of a child object i digits right as follows:

\[ \text{ancestor}(c, i) = \text{shift-right}(\text{HID}(c), i) \]

(Where, \( c \) is the child object and \( i \) is the level)

Thus, the HID of the ancestor node at the 2-level backward for the node page[2].frame[1].block[2] is calculated as follows:

```sql
ancestor(page[2].frame[1].block[2], 2)
= \text{shift-right}(\text{HID}(page[2].frame[1].block[2]), 2)
= \text{shift-right}('212', 2)
= \times (= \text{HID}(\text{page}[2]))
```

The HIDs of the child nodes at j-th level forward can be obtained by shifting the HID of a parent object \( j \) digits left. For instance, the child nodes at the 2-level forward of the node page[2] in Figure 2 have the HIDs of '2xx' form.

The operations ancestor and children are defined in the type Doc.Structure as methods, and inherited by all element types and used to locate ancestor and descendant objects fast without looking up additional information. Each digit of the HID should be implemented in order to accommodate the maximum number of child elements. For example, 256 child elements can be identified if we implement each digit of the HID to have one byte.

In general, the aggregation relationship is defined as a parent-child two-way relationship on a composite hierarchy in object-oriented database. However, it is required to access all intermediate objects if we want to retrieve an ancestor or a descendant object of an object when the difference of levels between them is greater than 1. By using our HID scheme and ancestor, children functions, we do not need to access all the objects between them. Therefore, it provides great advantages in minimizing the disk access time by calculation only without disk access and in enhancing the performance of accessing ancestors and descendants of an object at any level even though we do not know the type of node.

The number associated with each node in Figure 2 and 3 is the HID. We use three kinds of HIDs — lid(logical HID), sid(spatial layout HID), tid(temporal layout HID) — for each element of three structures. The object key of an element consists of its HID and the doc.id of the document which the element belongs to. Each of the attributes lid, sid, and tid is defined in the type Logical.Element, SpLayout.Element, and TempLayout.Element, respectively.
4 Defining Multimedia Data Types

Various kinds of multimedia data can be supported in our system (Figure 7). The supertype of all the media types is Media defined as follows:

CREATE OBJECT TYPE Media
(PUBLIC
  start Time,
  end Time,
PRIVATE
  content blob,
  position Point);

In type Media, start and end are attributes for specifying start time and end time of each media data, and used to represent the temporal relationship. The attribute Position is used for spatial presentation.

Figure 7: The type hierarchy for multimedia data

The actual data of each media type is stored in the attribute content as a blob(binary large object) in our framework.

4.1 Multimedia Data Types

The type Media have subtypes such as Text, Graphic, Image, Audio and Video (Figure 7). Each media type(e.g. Image) can also have subtypes(e.g. Jpeg, Gif) according to its data format.

Methods for editing and querying can be implemented in each media type. For simplicity, we assume that content-based queries on multimedia data are based on conventional information retrieval techniques [19], which search given keywords in the text description or captions or annotations associated with multimedia data. However, it is possible to use media specific search methods implemented in each media type to pose queries. The following is the type Image defined in this paper.

CREATE OBJECT TYPE Image UNDER Media
PRIVATE
  FUNCTION contains(p1 Image,
    search-expr varchar(max_pattern_length))
  RETURNS BOOLEAN;
FUNCTION zoomin(p1 Image, r Ratio) RETURNS Image;
FUNCTION zoomout(p1 Image, r Ratio) RETURNS Image;
FUNCTION rotatecpl Image, a Angle) RETURNS Image;
FUNCTION equal(p1 Image, p2 Image) RETURNS BOOLEAN;
FUNCTION shift(p1 Image, d Direction)
  RETURNS Image;
FUNCTION superimpose(p1 Image, p2 SET(Image))
  RETURNS SET(Image);
FUNCTION color_transform(p1 Image, c Color)
  RETURNS Image;
FUNCTION union(p1 Image, p2 SET(Image))
  RETURNS BOOLEAN;
FUNCTION intersect(p1 Image, p2 SET(Image))

As shown above, the attribute caption is the description data for content-based retrieval. The method contains searches whether the value of caption contains given keywords. Other methods are media-specific methods. For example, the method shift is used for shifting an image according to a given direction. We have also defined other media types(e.g. Graphic, Audio, Video) with the content-based method contains and media specific methods [15].

The method contains is defined in the type Text as follows:

CREATE OBJECT TYPE Text UNDER Media
PRIVATE
  search_expr varchar(max_pattern_length))
  RETURNS BOOLEAN,
FUNCTION contains(text Text,
  search_expr varchar(max_pattern_length))
  RETURNS BOOLEAN;
FUNCTION contains-any(text Text,
  tl LIST(Text_Token)) RETURNS BOOLEAN;
FUNCTION contains-all(text Text,
  tl LIST(Text_Token)) RETURNS BOOLEAN;
FUNCTION contains_phrase(text Text, phr Text)
  RETURNS BOOLEAN;
FUNCTION contains_forward_trunc(text Text,
  t Text_Token) RETURNS LIST(Text_Token);
FUNCTION contains_backward_trunc(text Text,
  t Text_Token) RETURNS LIST(Text_Token);
FUNCTION contains_double_trunc(text Text,
  t Text_Token) RETURNS LIST(Text_Token);
);

In the definition of the method contains, the argument search_expr is the search expression. The method contains returns true if the argument text satisfies the search expression, and otherwise returns false. It is possible for users to make search expression by using not only AND, OR and NOT operators but also PRIVATE methods defined in the type Text. The syntax of search expression is defined as follows:

<search expression> ::= <search term>
| <search expression> OR <search term>
| <search term> AND <search factor>
| [NOT] <search primary>
| <text literal>
| <asterisk> <text literal> <asterisk>
| <text literal> <plus sign>
| <text function invocation>

<text literal> ::= <character string literal>
| <text function invocation> . .
| <text function name> <argument list>
| <text function name> ::= Phrase | Double_Trunc
| Forward_Trunc | Backward_Trunc

Example usage of the method contains are given below.

contains(t1, 'X AND Y')
contains(t1, 'X OR Y')
contains(t1, 'NOT X')
contains(t1, Phrase('X Y'))

337
If X is ‘Chungnam’ and Y is ‘University’, the condition \( \text{contains}(t1, \text{‘Chungnam AND University’}) \) is true when \( t1 \) contains both words, while the condition \( \text{contains}(t1, \text{Phrase(‘Chungnam University’)}) \) is true when \( t1 \) includes a phrase ‘Chungnam University’.

4.2 Sharing Multimedia Data

The actual multimedia should be contained in each leaf object such as paragraph, image and block in composite objects of Figure 2 and 3. However, it may cause two problems.

- Multimedia data are duplicated if we store the actual data in every leaf node of each structure. For example, the leaf node \text{section}[1].\text{para}[2] of the logical structure in Figure 2 has the same data with two blocks \text{page}[1].\text{frame}[2].\text{block}[2] and \text{page}[2].\text{frame}[1].\text{block}[1]. It is required to avoid data duplication and to share the same data content between multiple structures.

- Retrieving the whole content of a document can degrade the performance seriously since we have to access every leaf objects.

In order to solve the above problems, each leaf object is not supposed to contain actual data in our framework. Instead, the actual data of each media type is stored as a blob, and each leaf object contains the pointer to the original source data (i.e. blob data) and the list of offset denoting the start position within the source data and length in bytes. This type of object is modeled as the type \text{Fragment}, which is a subtype of the type \text{Element} in Figure 7. The type \text{Fragment} is defined to have the six attributes: \text{doc.id} of a document, HIDs (i.e. \text{lid}, \text{sid}, \text{tid}) of each structure, \text{source} and \text{offset}. For example, the second paragraph of the first section of the multimedia document ‘mm1’ in Figure 1 consists of two text fragments because the paragraph is divided into two blocks after the layout processing is completed. The two text fragment objects are shown below:

- \text{text-fragment4} [doc.id: ‘mm1’, lid:42, sid:122, tid:11, source:text1, offset:<(120,60)>]
- \text{text-fragment5} [doc.id: ‘mm1’, lid:42, sid:211, tid:21, source:text1, offset:<(180,20)>]

The system can know that two objects belong to the same logical paragraph since they have the same \text{tid}. However, they are presented in different pages and synchronized with other data as they have different \text{sids} and \text{tids}.

By treating multimedia data in this manner, we can share multimedia data between multiple structures and enhance the performance in retrieving the whole content of multimedia information.

5 The Multimedia Query Language

We provide the following four types of queries which should be supported for multi-structured multimedia information.

- Attribute-based queries: queries based on attributes (e.g., \text{doc.id}, \text{rev.date}).
- Content-based queries: queries based on the document contents which include text and multimedia data
- Structure queries: queries based on the logical, spatial/temporal layout structural elements
- Hyperlink queries: queries based on the hyperlink networks

We can also make more powerful mixed queries of the above four types. The syntax of the query language is specified in compliance with SQL3 standard. The query expressions are extended to support efficient description of multiple structures of multimedia information and hyperlink traversal.

5.1 Basic Expressions

The following shows a typical example query based on attributes.

\[
\text{SELECT m.rev.date} \\
\text{FROM MM-Article m} \\
\text{WHERE m.doc.id = ‘mm1’}
\]

The above query is that “Select revision date of a MM-Article whose \text{doc.id} is ‘mm1’.", where \( m \) is an object variable for the type MM-Article. The dot notation is used to invoke a method in SQL3. In SQL3, the \text{Observer} methods are automatically created, which have the same name with the attribute definition, and are used to retrieve attribute values.

In this paper, we extended the three query expressions. The extended query expressions are \text{component} expression, \text{follow} expression and \text{list} expression.

The \text{component} expression allows navigating to ancestor or descendant elements of an element through a document’s specific structure. The syntax of \text{component} expression is \text{component} \text{parameter}, where \text{parameter} specifies the number of levels to navigate forwards or backwards. The \text{parameters} are defined as follows:

- \( */(-) \) : 0 or more levels forward(backward)
- \( +(-) \) : 1 or more levels forward(backward)
- \( $(-) \) : levels to leaf(root) node(s)
- \( !(-) \) : i levels forward(backward)
- \( i^j(-i \ ^j) \) : i \ j levels forward(backward)

The following show some examples of the \text{component} expression for the composite object in Figure 4.
The *component* expression is implemented by using *ancestor* and *children* functions described in Section 3.2. It is defined as the method of type *Element* of Figure 5 and inherited by its subtypes.

The *follow* expression makes the hypertext navigation possible. We can specify not only the navigation of 0 or more levels through the hyperlinks but also forward/backward traversal. The *follow* expression is defined similarly as the *component* expression like *follow(parameter)*. It is implemented as the method of type *Link* of Figure 5.

The following show some examples of the *follow* expression for the hyperlink network in Figure 4.

E..follow(+) = {I,J,K,L,M}
E..follow(2) = {K,L,M}
K..follow(-1) = {I,E,L}
K..follow(-2) = {G,E}

The *list* expression is designed to locate a specific portion of a list of elements. It is defined as follows:

- Either $L[i]$ or $L[i:j]$ is a list expression, where $L$ is an element name, component expression, or follow expression, and $i$ and $j$ are integers or character '$$'. $L[i]$ returns the $i$-th element, or the last element in the case of '$$', from the list. $L[i:j]$ extracts the sublist of $L$ starting at position $i$ and ending at position $j$.

Examples of the list expression are given below.

sections[2]..paragraphs[8]
: the last paragraph of the second section
sections[1:3]
: the first, the second, and the third sections
E..component(2)[3]
: the third child at 2 levels forward

SQL3 supports the *head* operation to extract the first element and the *tail* operation to extract the last element for the attribute of type *LIST*. The *list* expression can be implemented by recursively applying the parameter value of head and tail operation. It is also defined as the method of type *Element*.

The *component* expression, *follow* expression, and *list* expression can be combined to make more powerful expressions. The following examples show combined expressions for the example in Figure 4.

A..component(2)[2]..follow(1) = {I,J}
E..follow(1)[2]..follow(-1) = {E}
K..follow(-1)[1]..component(1) = {D,E}
J..follow(-1)..component(+) = {F,G,H}

It is not needed to modify existing database language processor for implementing these expressions since they are defined as methods.

### 5.2 Queries Based on the Logical Structure

Using our extended expressions, users can access each element of the logical structures. For example, the query, "Select all memos which refer to MM_Article whose titles contain 'Chungnam' and 'University'.", is expressed as follows:

```sql
select m from MM_Article a, Memo m
where contains(a..title, 'Chungnam'
and 'University') and m in a..follow(-)
```

Note that hyperlink query is combined with logical element access in the above query.

The query, "Select the titles of MM_Article that are referenced by the last child of the first section of the MM_Article 'mml'.", can be posed as follows:

```sql
select b..title
from MM_Article a
where a..doc-id='mml'
and b in a..sections[1]..component[1][9]..follow(1)
```

### 5.3 Queries Based on the Spatial/ Temporal Layout Structure

Queries about the spatial layout structure are similar to queries about the logical structure except the spatial layout elements are used. One of advantages of our query language is that queries can be posed without a precise knowledge of the spatial layout structure.

For example, the query, "Select all blocks presented on the same page with a video block.", can be expressed as follows:

```sql
select a..pages..component($)
from MM_Article a
where for some a..pages..component($)
as b (type(b) in (Video))
```

In the above query, the expression 'type(b) in (Video)' asserts that $b$ is an object of the type *Video*. If it turns out that it is not true, an exception is raised at runtime. This query returns the set of blocks which are presented on the same page with the video block. In our example of Figure 1, two text blocks, two image blocks, and a video block are returned as a result. This kind of query cannot be processed if we only maintain the logical structure information.

We can make a query with the spatial layout information and hyperlinks. For example, the query, "Select the documents referenced directly or indirectly by the page two of a MM_Article whose doc_id is 'mml'.", is expressed as follows:

```sql
select d from MM_Article a, Doc_Type d
where a..doc_id='mml'
and d in a..pages[2]..follow(+)
```
Queries based on the temporal layout structure are focusing on the temporal relationships between presented objects. As shown in Figure 3(b), the synchronization relationships are represented by a hierarchical structure between the media instances.

The following example shows a typical temporal structure query, "Select the blocks of MM_Article presented in parallel and equal duration with the second image of the second section of the 'mm1'."

```sql
select b
from MM_Article a, (a..component($) $) b
where a..docid='mm1' and
      equal(b, a..sections[2]..image[2])
```

The above query uses `equal` operation defined in the type `Media`. It is inherited by each media fragment type because they are subtypes of the type `Media`. The expression `equal(b, a..sections[2]..image[2])` tests whether objects `b` and `a..sections[2]..image[2]` have an equal temporal relationship. If it holds, then it returns `true`. The answer of the query will be the object `video[1]" as shown in Figure 3. The method `equal` is actually implemented like the following: "image[2]..start=b..start and image[2]..end=b.end". The attributes `start` and `end` denote the starting and ending coordinates in presenting a media object.

We need to implement operations for all Allen's thirteen temporal relationships [1] between two intervals of objects in order to support temporal structure queries. However, only seven operations (i.e., `before`, `meets`, `during`, `overlaps`, `starts`, `finishes`, and `equal`) are implemented because the thirteen relationships can be represented by seven cases since six of them are inverses. For example, `after` is the inverse relationship of `before` (before(a, b) is the same as after(b, a)).

The query, "Select the blocks of MM_Article presented just before an image containing the keyword 'SGML'.", is expressed as follows:

```sql
select c
from MM_Article a, (a..component($) $) b c
where contains(b, 'SGML') and
      contains(c, 'SGML') and
      meets(c, b)
```

The condition `meets(c, b)` in the above query is satisfied when there is a `meets` relationship between `c` and `b` objects. In other words, `c` is presented just before `b` without any gap. If we assume that the image object containing the keyword 'SGML' is `image[2]` in Figure 3, the result will be `video[1]".

5.4 Queries Based on Multi-structure Information

Using the multimedia query language proposed in this paper, we can make queries based on not only each structure but also the combination of several structures.

The example query, "Select the pages presenting sections whose elements have the keyword 'institute'.", is to retrieve the spatial layout elements based on the logical elements.

```sql
select p
from MM_Article a, a..pages p, a..sections s
where for some s..component($) $
      as x(contains(x, 'institute'))
      and p in x..component(-)
```

In the above query, object variable `x` is given in order to find the common leaf nodes of logical elements and spatial layout blocks. This query shows that it is possible to search the layout elements related to a certain logical element. We can also search the logical elements based on the layout elements.

Queries can be posed based on any combination of logical, spatial layout, temporal layout, and hyperlink network structures. To our knowledge, this functionality cannot be supported by previous studies.

5.5 Querying Structured Multimedia Data

Some multimedia data such as graphics, animation, video, and music have inherent hierarchical structures [9]. We can consider the application of our query language to such structured multimedia data because they have structures like documents. For example, a video clip can consist of one or more segments, a segment can consist of one or more events, and an event can consist of one or more shots. Our query language can be used to manage and handle the structure information of structured multimedia data.

In order to obtain keywords from video structure elements, we can use user annotations or captions. Users can freely annotate video elements with text data and keywords can be extracted from the text data. Other methods can also be used. For example, automated video analysis tools can be used to extract keywords from video frames.

Queries can be posed based on the structure of video data. Example queries on the structure of video data are given below.

- Select segments that have an event about 'waterfall'.
  ```sql
  select v.segments
  from Video v
  where contains(v.segments, 'waterfall')
  ```

- Select all audio nodes referenced by the memos which have a link to a video node.
  ```sql
  select a
  from Audio a, Memo m, Video v
  where a in m..follow(1) and m in v..follow(-1)
  ```

6 Related Work

Much research has been performed on modeling structures of documents [2, 3, 5, 7, 17, 18]. Most
of the models are based on the object-oriented modeling techniques since the structure information has hierarchical characteristics and current structured document standards such as SGML, ODA, HyTime, and HTML also use the object-oriented concept. However, most previous studies have focused on only the logical structure of text-oriented documents [2, 7, 17, 18].

Query languages for structured documents should support content-based access and structure-based access [18]. Our query language supports four types of queries: attribute-based query, content-based query, structure-based query, and hyperlink-based query. We can also make more powerful mixed queries of the four types. Furthermore, we have proposed queries based on any combination of logical, spatial layout, temporal layout, and hyperlink network structures. To our knowledge, this functionality cannot be supported by previous work.

Arnold-Moore et al. [2] have presented a data model and query language for the SGML documents. They have defined a functional query language, called SGQL, which supports structure queries, as well as content queries. However, the SGQL is not user-friendly for complex structure queries because it requires many functions to be combined in a method similar to the low-level functional programming language. Moreover, users should know the procedure of query evaluation since the query language is procedural. Kimber [13] have proposed a query language, called HyQ, for the HyTime documents. The HyQ is also a functional query language which is similar to the SGQL. The HyQ provides basic functions for simple queries, and they are combined to generate a complex function. It is also a procedural language. Macleod [17] has defined a query language based on a tree-structured document model. His language has used the concept of path in a document hierarchy. In order to make a structure query, users must know the element names and their hierarchy defined in the DTD. Christophides et al. [7] have extended this path concept and presented a query language, called Extended O2SQL, which uses path expressions for structure queries. However, their path expression supports only forward directional navigation.

We have extended the path expression to represent backward paths from lower levels to higher levels in the document structure and backward hyperlinks. Thus, users can navigate in either direction freely and select any interested portion of a document or a group of documents. We also have extended the expression to be able to represent the number of hops by a method similar to the regular expression in automata theory. By using our path expression, users can easily control the number of hops of branches or links in the structure hierarchies or hyperlink networks, and express powerful structure queries. In order to locate ancestors or descendants of an element directly, we also have proposed a novel scheme which uses hierarchical element identifiers (HIDs). Thus, we can perform rapid structure query processing by calculation only without disk access.

Compared with the previous research, one more advantage is that our language is not dependent on any specific document standard, and is also not closely related to a specific DBMS implementation.

Implementing our language on an object-oriented DBMS, it is not needed to modify existing database query processor because all extended query expressions (i.e., component, follow, link expressions) can be coded as methods in each appropriate type.

7 Conclusions

Multimedia information disseminated in document style has gained popularity with the advent of World-Wide Web. Structure information is useful to retrieve the portions in multimedia information which users are interested in. The main contribution of this paper is that we design a query language which can fully utilize multi-structure information including hyperlink networks and spatial/temporal layout structures, as well as the logical structure in multimedia information.

We are currently implementing the multimedia query language using the UniSQL/X object-relational DBMS [20]. Even though there is no DBMS which can support full-level of SQL3 standard, the UniSQL/X DBMS supports fundamental language primitives which can be used in implementing our query language. In order to support full-text content-based retrieval, we have developed a prototype system which integrates the UniSQL/X DBMS and an information retrieval engine [16].

In this paper, we do not take into account update operations on multimedia information. The contents and structures of the multimedia information can be changed over time. Currently, we are studying the versioning mechanism.

References


341
tem,” ACM Transactions on Office Information Systems, 6(1), 1988.


