Knowledge-Assisted Content-Based Retrieval for Multimedia Databases

Atsuo Yoshitaka, Setsuko Kishida, Masahito Hirakawa, and Tadao Ichikawa
Hiroshima University, Japan

Unlike conventional databases, which manage only text and numerical data, multimedia databases must evaluate audio and visual properties of data. We propose a system of querying and content-based retrieval that considers audio or visual properties of multimedia data.

The need to manage various types of data in a database environment has drastically increased recently, in accordance with the development of a variety of media like CD-ROM. Multimedia database systems try to uniformly manage voice, image, and video data, as well as text and numerical data. Retrieval in these systems is characterized by several qualities, including the following:

- Because information consists of several kinds of data, a system should in some cases evaluate a query condition by referring to several different kinds of data at once.
- The type of query condition given by a user may differ from the target data to be retrieved.
- Multimedia data has rich contents from which we can derive several concepts included implicitly in it.

In managing queries of multimedia data, we think it's important to provide a facility that takes the above-mentioned characteristics into account to achieve better human-computer interaction, as well as to broaden the range of acceptable queries.

In particular, content-based retrieval is a major issue in processing queries in multimedia databases.

There are several approaches to content-based retrieval of image data. One approach is to attach textual and/or numerical information describing the contents or features of an image to the image data. These systems retrieve data by referring to the attached information rather than to the image data themselves. Another approach is to evaluate image data directly for an image segment given as a query condition.

In the first approach, it is difficult to retrieve image data properly, since textual description cannot represent the rich contents of image data. It is unrealistic to try to provide text and/or numerical descriptions to all the possible features that might be useful in content-based retrieval of image data. Further, when the standard of judgment changes, we must update all the attached textual descriptions. In addition, this approach makes it difficult to adjust the result of query evaluation to each user's standard of judgement. The second approach specifically implements content-based retrieval of image data for a specific application, resulting in application dependency. Meanwhile, from a querying point of view, in both approaches the query usually specifies the same type of condition as the type of data stored in a database. We call this direct querying.

Multimedia systems may evaluate heterogeneous types of data at the same time. The condition in a query, however, may need to be represented in a single media, such as text, for ease of specifying queries. We call this indirect querying, where the type of specified condition differs from that of target data. Indirect querying provides more freedom in specifying a query for content-based retrieval in a multimedia database. So far as we know, no system can evaluate multimedia information, that is, the information observable when the system refers to heterogeneous types of data at the same time.

Our proposed system enables indirect querying to retrieve the contents of heterogeneous types of data stored in a multimedia database. The system holds domain knowledge, which describes how the system views the target multimedia data for content-based retrieval. Domain knowledge can be structured in accordance with the database schema, which is the structure of data in the database. Providing domain knowledge also allows us to realize application independence, as well as media independence. We adopted an object-
oriented approach throughout the system's implementation.

Object-oriented data model
Tsuda et al. proposed the data model MORE® (multiple objects relationship), which has fundamental features similar to other object-oriented data models, such as the one proposed by Lecluse, Richard, and Velez. In MORE, every entity in an application domain is represented as an object. An object's behavior is prescribed with a method, which activates the object by receiving a certain message. Objects bearing the same characteristics (for example, structure) are managed as a single class. Generally, a class contains structural definitions, methods, or values that the objects in the class commonly possess.

MORE considers two kinds of hierarchies to manage the relationships among classes: class hierarchy and aggregation hierarchy. The class hierarchy represents an "is-a" relationship among classes. Generalizing a class yields a superclass, and specializing a class gives a subclass. A class's properties, such as instance variables and methods, are inherited from a class to its subclasses. The aggregation hierarchy represents the "part_of" relationship.

Figure 1 shows a database schema with example objects based on MORE. Ellipses represent classes, and small circles represent objects. A black circle represents a primitive object, which contains a certain value. A white circle represents a complex object, defined by the aggregation of primitive or complex objects. The object hierarchy (that is, the aggregation hierarchy) is shown by the arrows from the complex objects to their component objects. Class relationships (the class hierarchy) are illustrated as a Venn diagram.

For example, a General object (an object that belongs to the General class) is defined as the composition of an Integer object, a Name object, and a Student ID object, which are referred to by instance variables of age, name, and student ID. The Map and Portrait objects are the subclasses of the Image class and, therefore, these two classes inherit properties such as instance variables, methods, and others defined in the Image class.

Queries for multimedia databases
As noted, we classify queries into two categories, direct and indirect. Conventional databases can process direct queries, but need assistance to interpret indirect queries. We model a query Q as

\[ Q = dm( rf [ta, c] ) \]

Terms
Condition
Specified for an attribute. It takes a certain value that is included in the domain of the attribute value.

Database schema
A framework describing the structure of data stored in a database, including the names and types of attributes. Other information inherent to each data model is also described.

Domain knowledge
Knowledge defined in a class, to derive information which is implicitly included in the value of an instance of the class. For our purposes, domain knowledge conveys information in order to derive the contents of multimedia data.

Feature item
An attribute possessed by the Feature component of domain knowledge.

Pseudo attribute
Whereas an ordinary attribute in a database is associated with a specific value, a pseudo attribute is associated with a value derived from domain knowledge.

Query attribute
The part of the query that corresponds to an attribute in a database, or a pseudo attribute in domain knowledge.

Figure 1. The MORE® (multiple objects relationship) schema.

In the equation, \( dm \) denotes a domain or a class of objects to be retrieved, \( rf \) is an attribute for which a condition is specified, and \( c \) is a condition value. The retrieval function \( rf \) obtains data items under the restriction specified by a set of pairs of \( ta \) and \( c \), \( [ta, c] \).

For example, in Figure 1 a query asking for 20-year-old students is denoted as "student
(find_equal(student.gender, 'male')). We regard the query as a primitive query. A more complicated query could be described by combining primitive queries. Yoshitaka, Hirakawa, and Ichikawa offer a detailed discussion on the query model elsewhere.\textsuperscript{6}

Under this query model, direct queries are ones in which \( \alpha \) corresponds to an attribute existing in the database schema, and the condition \( \beta \) corresponds to the value(s) of \( \alpha \). In indirect queries, on the other hand, \( \alpha \) has not been defined in the database, and/or the condition value \( \beta \) is not comparable with the values of \( \alpha \).

We further classify indirect queries into three types, as follows.

In Type 1, \( \alpha \) has not been defined in the database, but there exists an attribute \( \alpha' \) equivalent to \( \alpha \). The condition value is included in the domain of \( \alpha' \). In other words, this type of query is feasible simply by replacing \( \alpha \) with \( \alpha' \).

In Type 2, \( \alpha \) has been defined in the database but the condition value \( \beta \) is out of the domain of \( \alpha \). That is, the type of condition differs from that of the domain of \( \alpha \). Thus, comparison between \( \beta \) and the values of \( \alpha \) is not feasible. You can make this query feasible by providing a function translating \( \beta \) into \( \bar{\beta} \), where \( \bar{\beta} \) is equivalent to \( \beta \), but of a type of data included in the domain of attribute value \( \alpha \).

With reference to Figure 1, consider the example

\[
Q = \text{student ( find_near[student.additio nal_information.map, 'school'] )}
\]

This is a query to find a student whose house is near the school. Since the map attribute is of an image class, it is impossible to compare its value (map image) with the text string ‘school.’ To make the comparison feasible, you need to provide a way to interpret the map.

In Type 3, the query attribute \( \alpha \) has not been defined in the database and the condition \( \beta \) does not correspond to any domains of attribute value in \( \alpha \). These queries are feasible if the domain represented by \( \alpha \) (and not found in \( \alpha \)) is derived from the domains of existing attributes in \( \alpha \), and if \( \beta \) is either included in the derived domain or translated into an equivalent condition \( \bar{\beta} \) included in the derived domain. An example is “find the student(s) whose hair is long.”

\[
Q = \text{student ( find_equal [student.hair, 'long'] )}
\]

Obviously, this query is not feasible, since neither the attribute “hair” nor any equivalent attribute exists in the schema, nor is there a domain of hair length containing the value “long.” However, under the assumption that the domain of the length of a student’s hair can be derived from the contents of “portrait” image data, and the degree of the hair length can be mapped into the domain of words containing, for example, “long” or “short” through an additionally provided attribute for hair length, the query becomes feasible.

Queries for multimedia databases may be characterized as either direct or indirect, especially Type 2 or Type 3. A direct query’s condition may be an image or a sound, as well as text or numerical data. For indirect queries, the condition may be specified as text or numbers, even if it is for image or sound data (or vice versa). We are most interested in Type 2 and Type 3 indirect queries.

Domain knowledge

Let’s assume that a multimedia database is constructed on an object-oriented data model such as MORE.\textsuperscript{6} The framework described below is based on that described in Yoshitaka, Hirakawa, and Ichikawa,\textsuperscript{6} but is slightly modified to adapt to object-oriented data models.

Domain knowledge \( Dk \) is a way for a class to represent knowledge regarding a certain concept held by objects in the class. It is defined as a triple,

\[
Dk(C) = <Fi [fi, exp], Op [op [fi, mf]], Cm [cd, fi, v]>
\]

In the description, \( C \) denotes a concept representing a pseudo object. This pseudo object derives from the objects in a class by providing the domain knowledge. \( Fi \), \( Op \), and \( Cm \) denote a feature, an operator semantic, and a condition mapper, and a pair of brackets represents a set.

Feature. \( Fi \) represents the features constituting a concept \( C \), such as “color” and “length” for the concept “hair.” A feature \( Fi \) consists of a feature item \( fi \) and a procedure \( exp \) to extract the information from objects in the associated class. A feature item is the instance variable (the query attribute) of a pseudo object representing the concept \( C \). For example, if two feature items named color and length are defined for a piece of domain knowledge whose concept is hair, a pseudo object derived through the domain knowledge has two attributes (pseudo component objects) called color and length.

Operator semantics. \( Op \) defines the semantics of operators appearing in a query and how the oper-
ator is evaluated during the retrieval. The semantic behavior of an operator may change depending on the class of objects to be evaluated. For example, the behavior of an operator "=" for objects in an integer class differs from that in a color class. A member of $Op$ thus consists of an operator $op$ and a set of descriptions of semantic behaviors corresponding to the operator. That is, the description of a semantic behavior is given by the combination of a specification of a feature item $fi$ and a function $nf$ for evaluating the fitness between the extracted value of feature item $fi$ and a data value $v$ (which is a part of the description of Condition Mapper). $Op$ itself possesses a formalization function that takes the result of one or more functions and returns an evaluation value that is normalized to take from 0.0 to 1.0. The higher the value, the more the object satisfies the query condition.

Condition mapper. $Cm$ converts a condition value $cd$ specified in a query into a certain data value (or a certain range of values) $v$ whose data type is the same as that of the data values of $fi$. Therefore, both $fi$ and $v$ are the same type and are processed through $nf$. $v$ can be a certain function $f(cd)$ that returns a certain data value (or a certain range of values).

Figure 2 shows an example of a domain knowledge description for the concept Hair in the Portrait class. The feature items “length” and “color” are specified with the feature extraction functions “ext_length_proc” and “ext_color_proc.” To derive the “length” feature of “hair,” the system extracts the hair part from a portrait image by applying the ext_length_proc function, then calculates the length of the hair. The resultant data value is compared with a threshold value, for example, “140” when the hair is long.

Pseudo attribute
A pseudo attribute’s values are not stored physically in the database but derived from existing attribute values by referring to domain knowledge. The pseudo attribute is similar to a view in relational database systems but is different in that, since the pseudo attribute is provided for the object-oriented model, we can define a new attribute through the inheritance facility. Furthermore, separation of the structural description (the pseudo attribute) from the functional description (domain knowledge) allows us to design a database schema or the domain knowledge itself in a more flexible manner. We discuss this in more detail later.

A pseudo attribute is defined as a part of the class definition and is described by the triple

\[
<\text{Att}_p, \text{Po}_\text{set}, \text{C}> \]

Here, $\text{Att}_p$ is the name of a pseudo attribute. $\text{Po}_\text{set}$ is the path from the class where $\text{Att}_p$ is defined to the existing class of objects being associated with $\text{Att}_p$, where the domain knowledge to be associated is defined. The path is described by a chain of instance variable names. The concept $\text{C}$ indicates one of the pieces of domain knowledge in the class referred to through $\text{Po}_\text{set}$. It is identified by the concept name.

The pseudo attribute description looks like

\[
\text{<hair Student.addtional_info.portrait Hair>}
\]

The name of the pseudo attribute being defined in the Student class is “hair,” and by defining it, we can specify the hair attribute followed by a feature item name (defined in the domain knowledge whose concept is Hair) and a condition about feature items of “Hair” in a query, just as we specify an ordinary query. When a user makes a query such as

\[
Q = \text{Student(} = \text{[Student.hair.length, ’long’]} \)
\]

the system refers to the “Hair” domain knowledge in the “Portrait” class to interpret the condition and map the pseudo attribute to the actual attribute.

We separate the definition of a pseudo attribute from that of domain knowledge mainly to enhance the reusability of the latter. Consider the alternative, where domain knowledge is specified together with the pseudo attribute definition.
“home” is defined for the Student class, which is connected to the domain knowledge “Home” in the Map class. The definition of the pseudo attribute is

<home, Student.additional.info.map, Home>

Now consider a query to get the students whose residence is near school.

Q = Student (near to [Student.home.location, school])

Assume here that in the domain knowledge “Home” a feature item (a query attribute) “location” is defined for identifying the location of each student’s residence. To process the query, the system must identify the school’s location. The domain knowledge of “school” should be provided beforehand with a feature item “location” (in the Map class), which shares the derived values of “location” in the domain knowledge “Home.” The notion of nested reference allows “location” in “school” to refer to “location” in “Home.”

Figure 3 describes this example. In the figure, the domain knowledge “Home” has a feature item “location” but has no description concerning the condition “school” in its condition mapper. In the Map class where the domain knowledge “Home” is defined, another domain knowledge “school” is defined, also with a feature item “location.” Here we assume that the derived values associated with “location” of “Home” are the same as those of “location” of “school.” A pseudo attribute “home” is defined in the Student class, which is associated with the domain knowledge “Home” in the Map class.

When a user makes this query, the pseudo attribute “home” is associated with the “Home” domain knowledge in the Map class. Then the feature item “location” in “Home” is identified as the query attribute.

In evaluating the condition “school,” the domain knowledge “Home” refers to another domain knowledge “school” defined in the Map class. It does so because there is no description about “school” in the condition mapper of “Home” but a feature item “location” is defined in “school.” Then the domain knowledge “Home” requests “school” to send the derived value corresponding to “location” of “school” and it extracts the values representing the location of the student’s home so as to evaluate the query with the derived value of the location of school. The exam-
ple query is processed by referring to another domain knowledge.

**Composition of domain knowledge**

A new domain knowledge can be defined based on existing ones located at descendant classes along the aggregation hierarchy. Thus we can define a new concept by combining pieces of predefined domain knowledge in the component classes.

Basically, composition of multiple pieces of domain knowledge can be done in the same way as the specification of a domain knowledge. However, in the composition, the feature items of a piece of domain knowledge defined in the descendant classes are specified at the extraction procedure part of the domain knowledge being defined. Then the feature items work as pointers to the extraction procedures of pieces of domain knowledge in the descendant classes, so they are substituted for an extraction procedure of the domain knowledge being composed.

Figure 4 illustrates the composition of a domain knowledge. In the figure, a Scene object is composed of a Video_with_annotation object and a Sound object. The Video_with_annotation object is composed of a String object and a Video object. The Video object contains scenery of mountains, a train station, a main street, and so on, and the Sound object is associated with the corresponding Video_with_annotation object. Meanwhile, assume that there is domain knowledge describing the concepts of mountains, sea, and buildings in the Video class and domain knowledge about waves, birds singing, cars, and trains in the Sound class.

The query, “Get the scene objects which include a mountain train,” can be represented as

\[ Q = \text{Scene} (\text{find} [\text{Scene.scene-features.object, 'mountain train'])} \]

This query is only feasible if the system understands the existence of mountains and trains. Thus, it is difficult to derive the existence of a mountain train only by evaluating video data in the database. Meanwhile, it’s easy to extract the existence of mountains from video data and that of a train from sound data, and the existence of a mountain train is then derived based on the extraction of both of them. So, we assume that the existence of a mountain is derived through the feature item “existence” in the domain knowledge “mountains,” and the existence of a train is done through the feature item “existence” in the domain knowledge “trains.”

To make the query feasible, the domain knowledge for a concept “included_object” is defined in the Scene class shown in Figure 4. It contains a feature item named “object” in the description. A pseudo attribute “scene_features” is defined in the Scene class, which associates itself with the domain knowledge “included_object.” The two path specifications,

“Scene.va.video.mountains.existence” and “Scene.sound.trains.existence”

are specified to point to extraction procedures for the feature item “object.”

The contents describing whether or not a mountain exists are defined in the “mountains” concept in the Video class, and the knowledge for deriving the sound of a train is defined at the “trains” concept in the Sound class. In the domain knowledge of “included_object,” the condition mapper that corresponds to the feature item “object” is specified as the combination of return values of the feature item “existence” in the domain knowledge “mountains” and the “exist-
Figure 6. The database schema in our prototype system.

This, we believe, provides an advanced way to process queries for multimedia data. The domain knowledge composition facility allows us to define a new concept that needs two or more data types. It thus makes it feasible to evaluate queries that refer to interrelated, heterogeneous types of data.

System organization

Programming in C, we implemented a prototype system supporting the query-processing facility explained above on an IBM PC-compatible computer running Microsoft Windows. The system operates in our research environment.

Figure 5 illustrates the system's organization. The system consists of four components: a query analyzer, a query translator, a query processor, and a domain knowledge base. When a query is given, the query analyzer analyzes it and interprets whether it is direct or indirect. If it is a direct query, the system transfers it directly to the query processor, which activates the database retrieval. Otherwise, the indirect query is transferred to the query translator, which translates it into a query the system can execute. Then the translated query components (Att, Opr, and Cond) are transferred to the query processor and executed in the same way as a direct query.

System behavior

As an example, let's consider the system's application to student information management. Figure 6 shows the database schema of the example database, and a pseudo attribute “hair” is defined in the Student class. In addition, we assume that the domain knowledge “Hair” associated with the hair attribute is provided, as shown in Figure 2.

In the domain knowledge, a feature item “length” is defined with the extraction procedure ext_hair_region, to extract the region of hair from portrait image data. In the operator semantics, we define the function to evaluate hair length. For simplicity, we calculate hair length as the proportion between vertical and horizontal length of the extract-
ed hair region (the feature item length). In the condition mapper, a value that is replaced with the condition 'long' is defined as a possible condition in the query.

Users specify a query by using a querying box (Figure 7). Example face objects (data) are shown in Figure 8. The query in Figure 7 asks to find students with long hair. The system examines the portrait image and obtains the hair region by calling the extraction procedure defined in the feature item. The comparison function returns a value between 0 and 1 by calculating the proportion between the height and width of the extracted hair region. Figure 9 shows the intermediate result of extracting the hair region and the subsequent evaluation. (The user doesn’t see this step.) The value at the bottom is the return value of the comparison function. Here, the higher the value, the higher the certainty for the condition being satisfied.

The system then displays the results, shown in Figure 10. Here the students with long hair are linearly ordered, according to the values given by the comparison function. Of course, we can see detailed information about the students. Figure 11 displays the data for the student whom the system decided has the longest hair.

**Conclusion and future direction**

Our system for query processing of multimedia databases can extract the contents of multimedia data, such as image, video, sound, or their arbitrary composition, in terms of domain knowledge. The domain knowledge for a concept prescribes a way of viewing multimedia data in order to extract the information contained implicitly among the heterogeneous types of data. It also
provides the mechanism to translate an operator or a condition appearing in a query into equivalent ones whose types are suited to internal query evaluation.

We introduced the concept of a pseudo attribute, which relates the domain knowledge defined in the class whose objects implicitly possess the concept to the class where the pseudo attribute is required to process the content-based retrieval.

Domain knowledge in the classes can be structured in accordance with the database schema. In other words, a piece of domain knowledge is referred to by another, or examples of domain knowledge in the classes are composed in their common parent class along the aggregation hierarchy to prescribe a new concept. The structure of data matches that of domain knowledge; both data and knowledge are represented in a single structure.

This system has several advantages over other methods of retrieval, which require supplementing multimedia data with descriptive text or numerical values. Adding descriptive data can be expensive, and it can cause trouble if values are set improperly or if the standard for judging relatedness changes. Furthermore, implementing indirect querying in our proposed system provides a wider view of multimedia data, reflecting the data's rich content and using it to interpret queries.

This approach requires more time to evaluate a query, compared to approaches that utilize text or numerical data to identify multimedia contents. In the example database, it takes 0.48 seconds on the average to evaluate the length of hair in an image of 140 × 128 pixels and 64 color degrees, using a 33-MHz IBM PC-compatible 486DX machine. Despite this disadvantage, we took this approach because of the above-mentioned reasons.

Our experiments with the prototype proved that the system behaves properly. However, we still have some issues to investigate. First, the standard for judging whether a person's hair is long or short, for example, differs from person to person. Therefore, we need an interactive way to feed back the user's standard to the system, to satisfy each user's sense of recognition. To provide this facility, we are implementing a domain knowledge editor, which helps define or modify the definition of domain knowledge.

Second, the definition of domain knowledge, with regard to feature item extraction procedures and comparison function, is currently described at a comparatively low level. We need to further abstract this definition so that a designer of domain knowledge is not required to specify detailed description.

Acknowledgments

We would like to thank Masanori Yoshimitsu for his implementation work of the prototype system, and Masaki Kurihara for his discussion. This research is partially supported by Bedin Shop Systems, Cornuda, Italy.

References

Atsuo Yoshitaka is a research associate at Hiroshima University. His research interests include object-oriented databases, multimedia databases, and visual user interfaces. He received BS and MS degrees in engineering from Hiroshima University in 1989 and 1991.

Setsuko Kishida works at Yokogawa Hewlett-Packard in Kobe, Japan. Her research interests include image processing and multimedia databases. She received BS and MS degrees in engineering from Hiroshima University in 1992 and 1994.

Tadao Ichikawa is a professor in the Faculty of Engineering at Hiroshima University, Hiroshima, Japan. His research interests include visual programming, databases, and multimedia computing. He founded the Technical Committee on Multimedia Computing at the IEEE Computer Society and currently serves as chair of the committee. Ichikawa received his Doctor of Engineering from Waseda University. He is a fellow of the IEEE.

Readers can contact Atsuo Yoshitaka at the Faculty of Engineering, Hiroshima University, Kagamiyama 1-4-1, Higashi-Hiroshima, 724, Japan, or by e-mail at yoshi@huis.hiroshima-u.ac.jp.

Masahito Hirakawa is an associate professor in the Faculty of Engineering at Hiroshima University. His current research interests include visual programming, multimedia database systems, multimedia software engineering, and computer-augmented environments. He received his BE from Hiroshima Institute of Technology in 1979 and his ME and Doctor of Engineering from Hiroshima University in 1981 and 1984.

---

**Orlando Multimedia '95**

**conference and expo**

February 22-24, 1995

Hyatt Orlando Hotel

Kissimmee, Florida

For managers and senior professionals in training and education. Technical sessions feature application descriptions in interactive:

- Education and Technical Training
- Distance Learning
- Multimedia Materials Development
- Performance Support Systems

Pre-conference tutorials cover introduction to multimedia learning systems, designing multimedia programs, digital technologies design and development, and more.

For more information

Call 800-457-6812

or 703-347-0055

SALT®