This paper has been accepted by the 3rd IEEE Pacific-Rim Conf. on Multimedia, Taiwan, Dec., 2002 for publication.

MediaView: A Semantic View Mechanism for Multimedia Modeling

Qing Li [1], *Jun Yang* [1,2], *Yueting Zhuang* [2]

[1] City University of Hong Kong, HKSAR, China
{itqli, itjyang}@cityu.edu.hk

[2] Zhejiang University, Hangzhou, China
yzhuang@cs.zju.edu.cn

ABSTRACT

The semantics of multimedia data, which features context-dependency and media-independency, is of vital importance to multimedia applications but inadequately supported by the state-of-the-art database technology. In this paper, we address this problem by proposing *MediaView* as an extended object-oriented view mechanism to bridge the "semantic gap" between conventional databases and semantics-intensive multimedia applications. This mechanism captures the dynamic semantics of multimedia using a modeling construct named *media view*, which formulates a customized context where *heterogeneous* media objects with similar/related semantics are characterized by *additional properties* and user-defined *semantic relationships*. View operators are proposed for the manipulation and derivation of media views. The feasibility of *MediaView* is validated by an experimental implementation, and its usefulness and elegancy is demonstrated by its application in a multimodal information retrieval system.

1. INTRODUCTION

Owning to the expansion of the Web, recent years witness a phenomenal growth of multimedia information in a variety of types, such as images, videos, animations, etc. The huge volume of multimedia data creates the challenge of manipulating the data in an organized, efficient, and scalable way, preferably, using a database approach. In the database community, however, although a large number of publications have been devoted to the data model, presentation, indexing, and query of multimedia data (e.g., [2], [4]), relatively small progress has been achieved regarding the semantic modeling of multimedia, which is of primary importance to various multimedia applications and systems. A typical multimedia application, say, authoring of electronic lecture notes, is more likely to query against the semantic content of data, e.g., "*find an illustration of the ANSI/SPARC three-schema database architecture*", rather than to query against the primitive features of data, e.g., "*find all the images in JPEG format with size above 200KB*". Therefore, it is critical for a database to model the semantics of multimedia data in order to effectively support the functionality of semantics-intensive multimedia applications. Unfortunately, most existing data models fail to capture precisely the semantic aspect of multimedia data, which features the following two unique properties:

- **Context-dependency**. Semantics is not a static and inherent property of a media object[1]. Rather, the semantic meaning of a media object is influenced by the application (or user) that manipulates the object, the role it plays, and other objects that interact with it, which constitutes a specific context around this object. As an example, consider the interpretations of van Gogh's famous painting "Sunflower", the leftmost image in both Fig.1 and Fig.2. When it is placed with the other two images in Fig.1, which are also the paintings of van Gogh, the meaning of "van Gogh's paintings" is suggested. When the same image is interpreted in the context of Fig.2, however, the meaning of "flower" is manifest. Moreover, a media object may acquire context-specific properties when interpreted in a certain context. For example, as a painting, the picture "Sunflower" can be described by the "artist" and "year", whereas as a flower it can have attribute like "category".

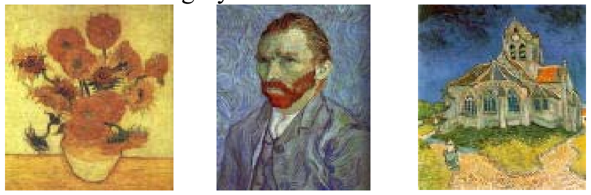
Fig.1: "Sunflower" in the context of *van Gogh's paintings*

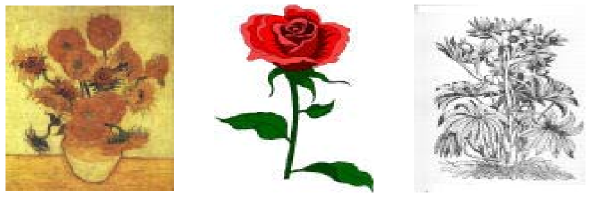
Fig.2: "Sunflower in the context of *flower*

- **Media-independency**. Media objects of different types of modality (i.e., multimodal objects) may suggest

[1]In this paper, a media object refers to an object of any type of modality, such as an image, a video clip, or a textual document.

the similar/related semantic meaning. For instance, the concept of "ANSI/SPARC three-schema architecture" can be expressed by a textual document, an image illustration, a PowerPoint slide, or a combination of them.

The dynamic nature of multimedia is fundamentally different from that of the traditional alphanumeric data, whose semantics is explicit, unique, and self-contained. This large distinction explains the failing of applying traditional data models to characterize the semantic aspect of multimedia data. For example, in a conventional (strongly typed) object-oriented model, each object statically belongs to exactly one type, which prescribes the attributes and behaviors of the object. This obviously conflicts with the context-dependent nature of a media object, which needs to switch dynamically among various types depending on specific contexts. Moreover, a conventional object model can hardly model the media-independency nature, which requires media objects of different types to have some attributes and methods defined in common.

The incapability of semantic multimedia modeling severely undermines the usefulness of a database in supporting semantics-intensive multimedia applications. This problem, referred to as the "semantic gap" between databases and multimedia applications, constitutes the major motivation of *MediaView* as an extended object-oriented view mechanism to be presented in this paper. As illustrated in Fig.3, *MediaView* bridges this "semantic gap" by introducing above the traditional three-schema database architecture an additional layer constituted by a set of modeling constructs named *media views*. Each media view, defined as an extended object view, formulates a customized context in which the dynamic and elusive semantics of media objects are properly interpreted.

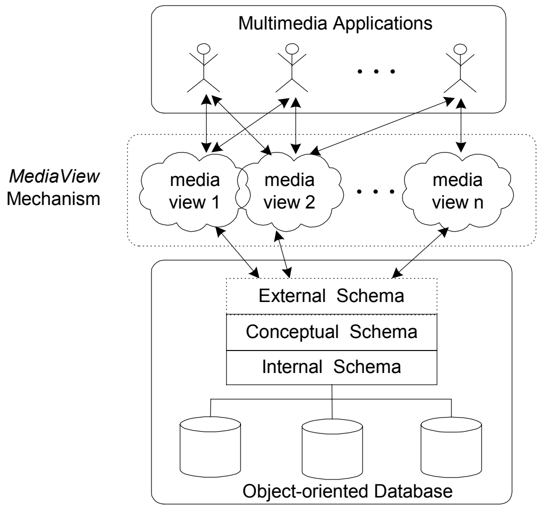

Fig.3: *MediaView* as a "semantic bridge"

To cope with the dynamic semantics of multimedia, *MediaView* builds the following extensions to the traditional object-oriented view mechanisms (e.g., [1], [6], [7]): (1) A media view can accommodate *heterogeneous* media objects (i.e., objects belonging to different classes) as its members. (2) Objects included as the members of a media view are endowed with *additional properties* that are specific to that media view. (3) Objects in a media view are interconnected by user-defined *semantic relationships*. A media view serves as a container that accommodates semantically related objects (typically heterogeneous) and describe them by additional properties and semantic relationships. The basic operations of media views, such as creation, deletion, and manipulation, are provided as a set of *view operators*. We demonstrate a real-world application, namely multimodal information retrieval, can be elegantly modeled by media views with its functionality adequately supported by view operators. An experimental implementation of *MediaView* is developed on top of an object-oriented database system.

The rest of this paper is organized as follows. In Section 2, we describe the fundamentals of the *MediaView* mechanism and compare it with other related works. We demonstrate the application of *MediaView* in a multimodal information retrieval system in Section 3. Section 4 describes the experimental implementation strategy of *MediaView*. We conclude the paper and discuss the future work in Section 5.

2. FUNDAMENTALS OF *MEDIAVIEW*

In this section, we introduce the basic concepts of media view as well as the view operators devised for its manipulation. The relationships between media view and existing modeling constructs are discussed as well.

2.1 Basic concepts

MediaView is essentially an extension built on top of a standard object-oriented data model. In an object model, real-world entities are modeled as objects. Each object is identified by a system-assigned identifier, and has a set of attributes and methods that describe the structural and behavioral properties of the corresponding entity. Objects with the same attributes and methods are clustered into classes. The definitions of class and other related concepts are given below:

Definition 1. *A **class** named as C_i is represented as a tuple of two elements:*

$$C_i = <O_i, P_i>$$

1. *O_i is the extent of C_i, which is a set of objects that belong to C_i. Each object $o \in O_i$ is called an instance of C_i.*
2. *P_i is a set of properties defined by C_i. Each property $p \in P_i$ is an attribute or a method that can be applied to all the instances of C_i.*

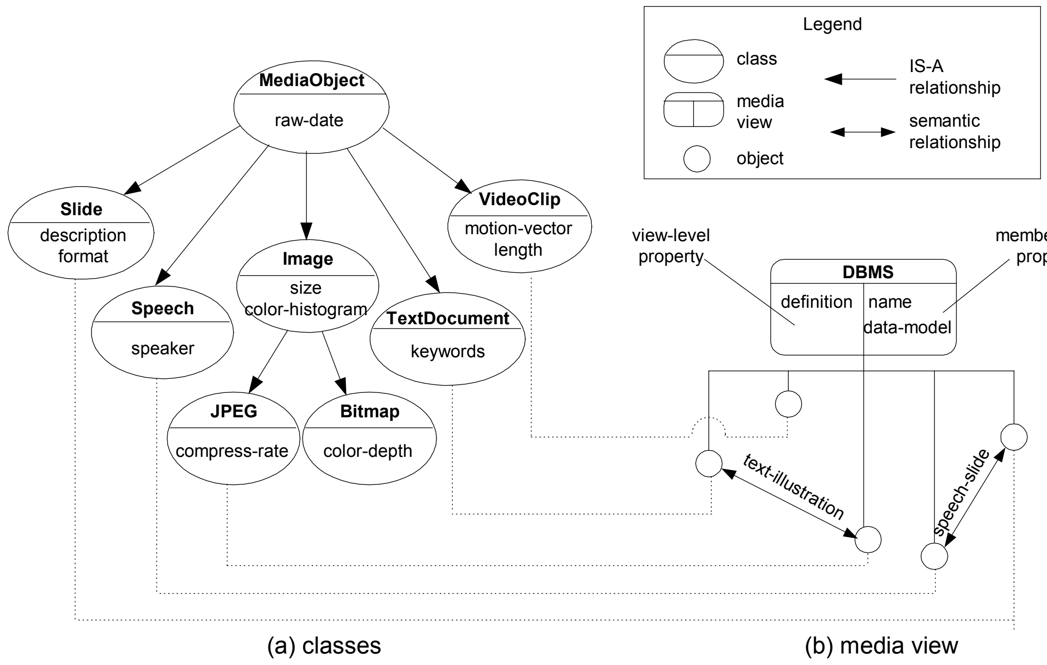

Fig.4: Examples of classes and a media view

3. *For two class C_1 and C_2, C_1 is called a* ***subclass*** *of C_2 (or C_2 is a* ***superclass*** *of C_1) if* (1) *$P_1 \supseteq P_2$ and* (2) *$O_1 \subseteq O_2$. If C_1 is a subclass of C_2, we also say that there is an* ***IS-A relationship*** *from C_1 to C_2 .*

Adopting object model has two advantages from the perspective of our work: (1) compared with other models (e.g., relational model), an object model has better modeling capability by capturing not only the structural but also the behavioral properties of an object; (2) each object has a unique identifier, such that its identity can be maintained when it is included into multiple contexts. Please note that this paper does not intend to propose a more powerful object model. Actually, the object model defined above is a basic subset (the core) of most existing object models [3], and therefore our *MediaView* mechanism is implementable on most existing object-oriented database systems.

The formal definition of media view as an extended object-oriented view is given as follows:

Definition 2: *A* ***media view*** *named as MV_i is represented as a tuple of four elements:*

$$MV_i = <M_i, P_i^v, P_i^m, R_i,>$$

1. *M_i is a set of objects that are included into MV_i as its members. Each object $o \in M_i$ belongs to a certain source class, and different members of MV_i may belong to different source classes.*
2. *P_i^v is a set of view-level properties (attributes and methods), which are applied on MV_i itself.*
3. *P_i^m is a set of member-level properties (attributes and methods), which are applied on all the members of MV_i.*
4. *R_i is a set of* ***relationships****, and each $r \in R_i$ is in the form of $<o_j, o_k, t>$, which denotes a relationship of type t between member o_j and o_k in MV_i.*

The relationship between classes and a media view is exemplified in Fig.4. As shown in Fig.4(a), a set of classes is defined to model media objects of different types, such as *Image*, *VideoClip*, and *Speech*, which are connected into the conceptual schema through IS-A relationships. From the properties defined in these classes, one can see that they emphasize on the primitive features of media objects, such as the color histogram of images, keywords of text document, which have uniform interpretation irrespective of specific contexts. Although such emphasis (on primitive features) is not mandatory, by doing so the conceptual schema is able to provide an objective, context-independent foundation based on which a variety of customized contexts can be formulated.

Fig.4(b) illustrates an example media view called *DBMS*. Each member of this media view is a media object that is about a specific DBMS product, such as a JPEG image illustrating a DBMS, a textual document about a DBMS, a slide as the demonstration of a DBMS, etc. Note that all these objects are not created by this media view, but are selected from heterogeneous source classes in Fig.4(a). However, these objects obtain a set of new (member-level) properties when they become the members of *DBMS,* such as the name of the DBMS product each of them represents. Different from their properties defined in respective source classes, their properties in the media view focus on the semantics suggested by media objects. Moreover, a view-level property, *definition*, is used to describe the global property of the media view itself (i.e., the definition of a DBMS). Different types of semantic relationships exist between the view members. For example, there is a "speech-slide" relationship between the *Speech* object and the *Slide* object, denoting that the speech accompanies the slide.

3.2 Operators over media views

To support manipulations on media views, we devise a set of view operators, whose definitions[2] are presented as follows.

1. *CREATE-MV (N: mv-name, VP: set-of-property-ref, MP: set-of-property-ref): mv-ref*. This operator creates a media view (*MV*) named as *N*, which takes the properties in *VP* as its view-level properties, and those in *MP* as its member-level properties. When executed successfully, it returns the reference to the created media view, which has no members and relationships initially.
2. *DELETE-MV (MV: mv-ref)*. This operator deletes a media view specified by the reference *MV* from the database. Along with the deletion of *MV*, all its members are excluded from *MV* with their properties (value) defined in *MV* removed. All the relationships in MV are also deleted. Note that the member itself as an instance of its source class is not deleted from the database.
3. *GET-ALL-MV():set-of-mv-ref*. This operator retrieves all the media views currently in the database. The return value is a set of references to these media views.
4. *ADD-MEMBER(MV: mv-ref, O: object-ref)*. This operator adds the object referred by *O* as a member of the media view referred by *MV*. All the member-level properties for *O* are set to their default values.
5. *REMOVE-MEMBER(MV: mv-ref, O: object-ref)*. This operator excludes the object *O* from the media view *MV*. All the relationships and properties of *O* in *MV* are also deleted.
6. *ADD-RELATIONSHIP(MV: mv-ref, O1: object-ref, O2: object-ref, R: relationship-type): relationship-ref*. This operator establishes a relationship of type *R* between objects *O1* and *O2*, which are the members of the media view *MV*. (In fact, *R* is the name of a class and the relationship is an instance of this class, which refers to the two associated objects as its properties.) If the operator is applied successfully, the reference to the relationship object is returned.
7. *REMOVE-RELATIONSHIP(MV: mv-ref, O1: object-ref, O2: object-ref[, R: relationship])*. If the last argument is not specified, this operator removes all their relationship(s) between objects *O1* and *O2* in the media view *MV*. Otherwise, it only deletes the relationships of the type specified by *R*.
8. *GET-ALL-MEMBER (MV: mv-ref): set-of-object-ref*. This operator retrieves all the (heterogeneous) objects as the members of the media view *MV*.
9. *HAS-MEMBER(MV: mv-ref, O: object-ref)*: *boolean*. This operator tests if object *O* is a member of the media view *MV*.
10. *GET-RELATED-MEMBER (MV: mv-ref, O: object-ref[, R: relationship]): set-of-object-ref*. This operator returns all the objects that have relationship of any type (if the last argument is absent) or of type *R* (if the last argument is given) with object *O* in the media view *MV*.
11. *GET-ALL-RELATIONSHIP (MV: mv-ref): set-of-relationship-ref*. This operator retrieves all the relationships in the media view *MV*.
12. *GET-VIEW-PROP (MV: mv-ref, P: property-ref): value*. This operator retrieves the value of the view-level property *P* of media view *MV*.
13. *SET-VIEW-PROP (MV: mv-ref, P: property-ref, V: value)*. This operator sets the value of the view-level property *P* of media view *MV* to the value specified by *V*.
14. *GET-MEMBER-PROP (MV: mv-ref, O: object-ref, P: property-ref, V: value)*. This operator retrieves the value of the member-level property *P* of object *O* in media view *MV*.
15. *SET-MEMBER-PROP (MV: mv-ref, O: object-ref, P: property-ref, V: value)*. This operator sets the value of the member-level property *P* of object *O* in media view *MV* to the value specified by *V*.

The set of view operators defined above provides the basic functions of media views, while more sophisticated operations can be implemented as a combination of these basic operators. For example, a search for objects that are related with a specific object in any media view can be handled by applying *GET-ALL-MV()* and *GET-RELATED-MEMBER()* in combination.

[2] In the definition of view operators, the suffix "-ref" represents the reference to object, which is actually a variable holding the *Oid* of an object. For example, *mv-ref* is the reference to a media view, *relationship-ref* is the reference to a relationship, etc. As will be seen in Section 4, media views, properties, relationships are all implemented as objects.

2.3. Discussion and comparison to related work

From the above descriptions, one can easily see a resemblance between a media view and some existing constructs in an object model, namely *class*, *object view*, and *composite object*. In the following, we compare media views with each of these constructs in order to clarify the position of our work in the framework of object models.

- **Class.** Similar to the extent of a class, a media view also contains a set of objects as its members, and it can apply (member-level) properties on them to describe their structural and behavioral properties. However, a media view differs from a class in several aspects, particular in that (1) it can accommodate heterogeneous obejcts, whereas a class only holds a set of uniform objects; (2) a media view can only dynamically include/exclude objects that are instances of source class(es), and does not create new objects; (3) while an object must belong to exactly one class, it can be included into arbitrary number of media views; (4) a media view models the semantic relationships and consequently the interaction between its members, which is not supported by class; (5) the global feature of a media view is captured by its view-level properties, another feature not supported by class.
- **Object view.** In the past decade, there exist numerous proposals on object-oriented view mechanisms (e.g., [1], [6], and [7]). Generally, an object view can be regarded as a virtual class derived by a query over classes [1]. In fact, an object view is almost a class except that its instances are selected from the instances of other classes, and in this regard it is closer (compared with class) to our media view. However, except point (2), the rest statements on the difference between a media view and a class hold for a conventional object view as well. Furthermore, with the ability of assigning new properties to its members, a media view is more powerful than a conventional view, whose properties are inherited or derived from classes (e.g., deriving the area of a circle object from its diameter).

  Admittedly, with these new features added, a media view can be hardly classified as an object view (and *MediaView* is no longer just a view mechanism) from a conventional point of view, although our initial thought was to adapt an object view for multimedia data. In this paper, we stick to the term "view" on the ground that (a) structurally, media views sit in-between the conceptual schema and the applications, the position where views are used to be, and (b) functionally, they are used to provide customized view of the data for a certain application.
- **Composite object**. From another perspective, a media view can be regarded as an extended composite object, which maintains two lists of object references—one list keeps the members of the media view, and the other keeps all the relationships (which are implemented as objects) between members. As a composite object, a media view naturally allows dynamic insertion/removal of its members and relationships. The view-level properties correspond to the properties of the composite object. As the major difference between them, however, a media view can define properties for its members, whereas a composite object cannot.

Essentially, a media view can be regarded as a "*hybrid*" of a class (or an object view as a virtual class) and a composite object. Consequently, it benefits from the advantages of both constructs, i.e., the modeling power of a class, which allows it to endow the objects with new properties, as well as the flexibility of a composite object (e.g., heterogeneous membership) indispensable for modeling the dynamic nature of multimedia.

3. REAL-WORLD APPLICATION: MULTIMODAL INFORMATION RETRIEVAL

To show the usefulness and elegancy of *MediaView*, we introduce a real-world application in which media views are found to be a natural and suitable modeling construct. The example application comes from our on-going research project on a multimodal information retrieval system, *Octopus* [8]. In this section, we firstly describe several specific media views created as the data model of *Octopus*, and then demonstrate how a variety of retrieval functions can be implemented using view operators.

3.1. Data model

Octopus is proposed to provide search functionality in multimedia repositories ranging from web to digital libraries, where data are typically of multiple types of modality. The basic search paradigm supported by *Octopus* is query-by-example, that is, a user forms a query by designating a media object as the sample object and the system retrieves all the media objects relevant to it. For example, using the poster (as an image) of the movie "Harry Potter" as the sample, we expect to receive media objects such as a textual introduction of the movie, a "highlight" video clip, and the music of the movie. Essential to such a multimodal retrieval system is the relevance defined between any two media objects, which is evaluated from the following three perspectives:

1. **User perceptions**. Two media objects are regarded as relevant if users have the same/similar interpretation of them, e.g., annotating them with the same keywords.
2. **Contextual relationship**. Media objects that are spatially adjacent or connected by hyperlinks are usually relevant to each other.
3. **Low-level features**. Low-level features (e.g., color histogram for images) can be extracted from media objects to describe their visual/aural characteristics.

Intuitively, media objects are considered relevant if they possess highly similar low-level features.

As shown in Fig.5, a media view called *KB* is created to model the relevance between any two media objects in the database of *Octopus*. The members of *KB* are media objects such as images, videos, audios, which are modelled as instances of heterogeneous source classes (see Fig.4). Three types of relationships (perceptual, contextual, and feature) are defined to represent the inter-object relevance from the aforementioned three perspectives. A weight can be associated with each relationship as its property to indicate the strength of the relevance.

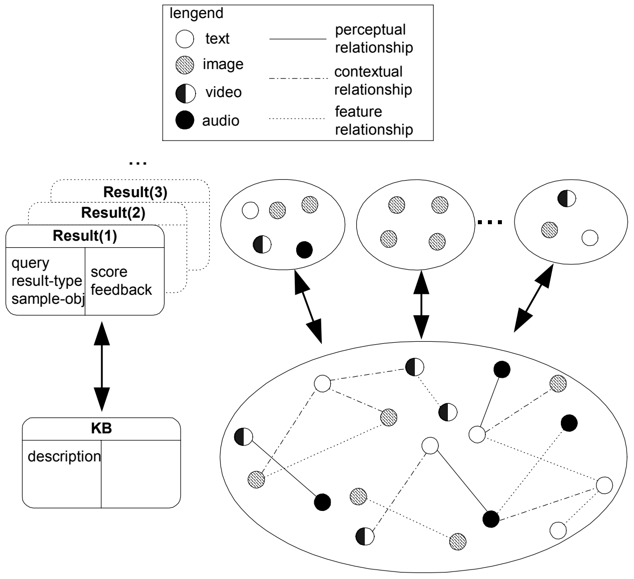

Fig.5: Media views created for *Octopus*

KB provides an integrated knowledge base for evaluating the relevance among media objects, based on which user queries can be processed by analysing the relationships contained in it (see below). For each query, a media view named *Result(n)* is created to accommodate the results of the query, where *n* is the serial number. The global aspect of the query is described by its view-level properties, such as the type of results, the sample object used, while member-level properties are assigned on each object to describe its characteristics as a query result, such as its relevance score, and users' feedback opinion towards it (relevant, neutral, or irrelevant).

3.2. Implementation of retrieval-related functions

Octopus provides a variety of retrieval-related functions, such as search, relevance feedback, navigation, learning, all of which are realized by applying view operators over the media view *KB* and *Result(n)*, as shown below.

- **Query-by-example (QBE)**. The media objects relevant to a sample object specified in a user query can be found by "propagation" via the relationships in the media view *KB*. Starting from the sample object, we traverse to other media objects in *KB* through relationships (up to a specific number of iterations) and identify these objects as relevant results. The pseudo-code describing this process is given below.

```
S: a set of objects as the query result
K: the number of iterations for propagation
$o_S$: the sample object
1.  S:= {$o_S$}
2.  For n= 1 to K
3.      T: = {}
4.      For each object o in S
5.          T := T $\cup$ GET-RELATED-MEMBER("KB", o)
6.      S := S $\cup$ T
```

We can designate the type(s) of relationship used in propagation by specifying it in *GET-RELATED-MEMBER* (Step 5). For example, if the feature relationship is unreliable under certain situations, we can specify the perceptual and contextual relationships for the search of relevant objects. Moreover, the modality of query results can be controlled by distinguishing the source class of each object (i.e., image, videos, etc). All the retrieval results, together with the user's possible feedback opinions towards them, are stored in the media view *Result(n)* created for the query.

- **Navigation**. As shown by the code below, navigation among the media objects can be facilitated by various relationships in *KB*, which serve as the natural routes for navigating from one media object to related objects.

```
1.  Set o as the object currently been viewed by the user
2.  S := GET-RELATED-MEMBER("KB", o)
3.  Present all the objects in S to the user, from which the
    user can choose an interested object and navigate to it
4.  Go to Step 1
```

- **Relevance feedback**. Relevance feedback is a mechanism used to refine the retrieval results by giving evaluations to the previously retrieved results, typically, by designating some of the results as relevant or irrelevant examples. The pseudo-code below presents a simple algorithm for relevance feedback. Similar to the algorithm for the search function, we perform propagation based on relevant and irrelevant examples respectively, resulting in a set of "positive" results and a set of "negative" results. The final results are obtained by removing the "negative" results from the "positive" ones.

```
S: a set of objects as the query results
R: a set of relevant examples
N: a set of irrelevant examples
K: the number of iterations for propagation
1.  For n=1 to K
2.      T: = {}
3.      For each object o in R
4.          T := T ∪ GET-RELATED-MEMBER("KB", o)
5.      R := R ∪ T
6.      T := {}
7.      For each object o in N
8.          T := T ∪ GET-RELATED-MEMBER("KB", o)
9.      N := N ∪ T
10. S := R-N
```

- **Learning from feedbacks**. In the previous two examples, retrieval results are obtained based on the knowledge contained in *KB*. Chances are that new knowledge can be derived from user feedback information recorded in *Result(n)* and incorporated into *KB*. The following algorithm shows a simple and intuitive way of doing that: if two objects are relevant examples for the same query (i.e., they appear in the same *Result(n)* with property *feedback* set as "relevant"), we add a perceptual relationship between them in *KB*. More sophisticated techniques (e.g., data mining) can be used for knowledge discovery based on media views, which are nevertheless out of the scope of this paper.

```
Result(n) (n=1,...,N): a set of media views for query results

1.  S: = GET-ALL-MEMBER("KB")
2.  For any two objects $o_i$ and $o_j$ in S
3.      For n = 1 to N
4.          mv := Result(n)
5.              If HAS-MEMBER(mv,$o_i$)
     and HAS-MEMBER(mv,$o_j$)
     and GET-MEMBER-PROP(mv, $o_i$, "feedback")="Relevant"
     and GET-MEMBER-PROP(mv, $o_j$, "feedback")="Relevant"
6.                  ADD-RELATIONSHIP("KB",$o_i$,$o_j$, "perceptual")
```

4. AN EXPERIMENTAL IMPLEMTATION

We have come up with an implementation strategy for *MediaView*, using which an experimental prototype has been developed on top of an object-oriented database system, *NeoAccess* [5]. Specifically, the concept of media view is implemented based on the notion of "meta view" and "shadow class".

NeoAccess is an object-oriented database management system with a C++ programming language interface. The library of *NeoAccess* introduces a *CNeoPersist* class, which defines all the basic properties and functions needed for persistent storage of objects. Persistent classes can be defined by inheriting from *CNeoPersist* using the standard grammar for C++ class definition.

As mentioned in Section 2.3, a media view has the characteristics of both a class and a composite object. Since a composite object does not support properties of its constituents, while a class cannot model the interactions (relationships) among its instances, a simple idea of implementing a media view as either of them does not work. Rather, a media view is implemented as a "hybrid" of the both constructs. On one hand, all the media views are created as instances of a common class called *Meta-View*, and on the other hand each of them is associated with a "shadow class" that models the properties of its members.

Fig.6 illustrates the implementation strategy of the media views used in the multimodal retrieval application discussed in Section 3. *Meta-View* is inherited from the class *CNeoPersist* as a built-in class, from which all the media views are created as its instances. *Meta-View* specifies the common data structures of media views, and provides all the basic functions in the form of view operators (cf. Section 2.2). Another built-in class, *Relationship*, is defined to represent the inter-object relationships and can be customized by inheritance for modeling different types of relationships. Each instance of *Relationship* or its subclasses refers to two object identifiers (i.e., *Oid*s) as its properties.

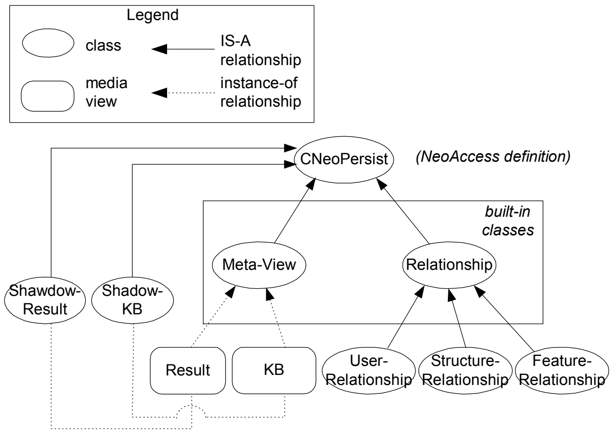

Fig.6: Implementation strategy of *MediaView*

Essential to the implementation of media view is the appropriate data structure chosen to meet its internal complexity (i.e., the data structure of *Meta-View*). In our approach, the members of a media view are maintained by a list of *Oid*s (of its member objects), and the relationships are kept in another list containing instances of the built-in class *Relationship* or its subclasses. Constraints are enforced to guarantee the data integrity during the evolution of members and relationships, e.g., when a member is removed, its involved relationships are also removed.

While the view-level properties of a media view is easily implemented as a list of <*Property, Value*> pairs, difficulty is encountered when it comes to the modeling of member-level properties, which vary from one media view to another. Since a composite object is not allowed to have properties for its members, we introduce a "shadow class" for each media view to model the properties of its members. Specifically, when a media view is created, a corresponding shadow class is defined implicitly, whose properties are set equal to the member-level properties of the media view. Furthermore, for each object identified as the member of the media view, a "shadow object" is created as the instance of the corresponding shadow class. All the accesses (read and write) to the properties of view members are automatically "forwarded" to their shadow objects. The removal of a view member will lead to the deletion of its shadow object. The mapping between an object as a view member and its shadow object (denoted by *shadow-Oid*) is realized by a list of <*Oid, shadow-Oid*> pairs maintained by the media view. All the manipulations of shadow classes and shadow objects are conducted in a user-transparent manner.

5. CONCLUSIONS AND FUTURE WORK

The *MediaView* mechanism presented in this paper aims at building a bridge across the "semantic gap" between conventional databases and multimedia applications, the former of which are inadequate to capture the dynamic semantics of multimedia, whereas data semantics plays a key role in the latter. This mechanism is based on the modeling construct of *media view*, which formulates a customized context where *heterogeneous* media objects with similar/related semantics are characterized by *additional properties* and *semantic relationships*. View operators have been developed for the derivation and manipulation of media views. The implementation strategy of *MediaView* as well as its application in a multimodal information retrieval system has been described to demonstrate its feasibility and usefulness.

As the current emphasis of *MediaView* is mainly on modeling capability, further issues regarding its efficiency need to be investigated. Efficient data structure and indexing strategy for media views will be developed in favour of the frequent operations, such as retrieving all the objects related to a specific one in a media view. Another promising research direction is to apply data mining technology. Despite the dynamic and subjective aspect of multimedia semantics, we believe that many informative patterns revealing the "common knowledge" of multimedia data can be discovered from media views through mining, such as the inter-object relevance derived from user feedbacks in the multimodal retrieval application (see Section 3.2). Such common knowledge can help improve the semantic characterization of multimedia and therefore better support multimedia applications.

6. REFERENCE

[1] S. Abiteboul and A. Bonner, "Objects and Views," *Proc. of ACM Conf. on Management of Data*, pp. 238-247, 1991.

[2] P. Apers, H. Blanken, and M. Houtsma, (eds.), *Multimedia Databases in Perspective*, Springer, London, 1997.

[3] W. Kim, "Object-Oriented Database Systems: Promises, Reality, and Future," *Proc. of 19th Very Large Database*, pp. 676-687, 1993.

[4] S.M. Chuang, (eds.) *Multimedia Information Storage and Management*, Kluwer Academic Publishers, USA, 1996.

[5] NeoAccess. http://www.neologic.com

[6] E.A. Rundensteiner, "MultiView: A Methodology for Supporting Multiple Views in Object-Oriented Databases," *Proc. of 18th Int. Conf. on Very Large Database*, pp. 187-198, 1992.

[7] M.H. Scholl, C. Lassch, and M. Tresch, "Updateable Views in Object-Oriented Databases," *Proc. of 2nd DOOD Conf*, Germany, 1991.

[8] J. Yang, Q. Li, and Y.T. Zhuang, "Octopus: Aggressive Search of Multi-Modality Data Using Multifaceted Knowledge Base," Proc. 11th Int. Conf. on World Wide Web, pp. 54-64, 2002.