The Roles and Views of Multimedia Objects

Raymond K. Wong*, H. Lewis Chau*, and Frederick H. Lochovsky*

Technical Report HKUST-CS96-11
April 25, 1996

*Department of Computer Science
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
Abstract

Although multimedia information technology has become popular, conventional database systems do not offer adequate support for the modeling, indexing, and manipulation of multimedia data. Database management systems need to be extended if they are to handle multimedia information like audio and video. In this paper we present an approach to extend the data model of an object-oriented database programming language by means of a meta-object protocol to implement the classes for multimedia objects. In particular, we model the different multimedia presentations (mostly with different characteristics) of a real-world entity as a single object which plays different roles. Moreover, we also propose different ways of using views to provide the right level of abstraction of the resources and data representation.
1 INTRODUCTION

The object-oriented paradigm provides several useful concepts which we can exploit to meet particular requirements [11, 17, 21] of modeling multimedia systems. For example, the object-oriented approach allows to model application specific data types and classes including associated operations. This is also the reason why most approaches for multimedia database systems follow object-oriented principles (e.g., [1, 5, 9, 13, 15, 17, 27]). However, the conventional object-oriented approach still lacks some features such as supporting time-dependent data, content-based query and retrieval techniques [16], etc. In particular, it does not support the concept of roles. In this paper, we propose the use of roles and views in a dynamic object-oriented programming language, called DOOR [30, 31, 32], to model multimedia objects.

The standard object-oriented data model is not satisfactory when entities, which change the class to which they belong and their behavior during their life, must be modeled. When an object is allowed to acquire new object types, the problem arises that different and inconsistent behaviors may have been defined for these different object types. This problem can be solved by preventing these inconsistent behaviors from interacting by using roles. In this approach, an object has a context dependent behavior and, in each context, it exploits only a consistent subset of its possible behaviors. This subset depends on the role that the object is playing in that context. A role mechanism addresses this problem and, more generally, allows one to model entities that can play several roles and behave according to the role being played [3, 22].

The concept of roles is becoming popular in advanced object-oriented database sys-
tems which provide powerful mechanisms for multi-faceted modeling and dynamic role changing. It has already proven its powerful modeling capabilities in many applications such as spatial and geographic applications, office information systems, manufacturing systems and robotics [18, 28, 29, 33]. However, no one has investigated the possibility of modeling the multifaceted and dynamic behavior of multimedia objects using roles.

The role mechanism of DOOR is implemented as a metaobject protocol which can be easily extended. In this paper, we show the extension of role modeling in DOOR to deal with advanced multimedia applications. We also describes the design of our prototype which can be easily extended to different multimedia devices.

Besides, views are important in multimedia applications because they provide the right level of abstraction of the resources, data distribution and data representation. Views are also important because they provide different views of the same data, e.g., to realize the often mentioned distinction between internal and external representation of multimedia data [16]. In DOOR, views are used to provide different views and abstractions of a multimedia object, and roles are used to model the specialization of a multimedia object. In other words, we regard objects and roles (but not views) as logical entities, and we regard views as presentations of these logical entities.

The organization of the rest of this paper is as follows. Section 2 introduces the fundamental of role modeling in DOOR, and presents the concept of roles for modeling the dynamic and multifaceted behavior of multimedia objects. Section 3 describes the applications of views on multimedia modeling. Views can provide different perspectives of a multimedia object. Section 4 lists some program code in DOOR for multimedia applications. The purpose of this section is to let the readers have some ideas about what
the DOOR code for multimedia applications looks like. In section 5, the architecture
of the DOOR prototype as well as the implementation of its multimedia support are
described. Section 6 is the related work section, and finally, section 7 concludes the paper
and describes our ongoing future research.

2 ROLES FOR MULTIMEDIA MODELING

2.1 Role Modeling for Entities in the Real World

In DOOR, a role is conceptually like an object, except that it has a special relationship
to other objects (or roles) which are said to play the role. A role can be played by an
object or by another role. We now describe roles informally with the following notation.
(Readers can refer to [30] for the formal definition of the role data model in DOOR.)

A class is a set of possible individuals, called class instances. If the instances are
objects, the class is called an object class. If the instances are roles, then the class is
called a role class. Let \( \text{inst}_\alpha(C) \) denote the set of all possible instances of class \( C \) with
the state of the world\(^1\) being \( \alpha \).

We assume there is a function called played-by in the model such that if \( OC \) is an
object class and \( RC_1, RC_2 \) are role classes, then in each state \( \alpha \) of the world, we have

\[
\text{played-by}: \text{inst}_\alpha(RC_1) \rightarrow \text{inst}_\alpha(OC) \cup \text{inst}_\alpha(RC_2)
\]

where \( \text{played-by}(r) \ (r \in \text{inst}_\alpha(RC_1)) \) is called the player of \( r \), and played-by has the

\(^1\)Here we use the notation state of the world [25] as a reference for time, to denote the time variant
properties of dynamic/evolving objects.
following properties:

1. Let $R$ be a role class. For any state $\alpha$ of the world, $r_1 \in inst_\alpha(R)$ and $r_2 =$
\[ \text{played-by}(r_1) \implies r_2 \neq r_1; \]

2. $\text{played-by}$ is neither a surjective function nor injective function.

The codomain of $\text{played-by}$ includes both the instances of object classes and role classes. Therefore, a role player can be an object, or even a role. However, by property (a) above, we eliminate the case that the player of a role instance is the role instance itself, although it is possible that both the player of a role instance and the role instance itself are of the same role class. For example, a person can be a club-member of a credit card club, and being a general club-member, he then can further join as a club-member of the privilege club of the credit card. The second property provides more information about the role playing characteristics. It implies that an object (or a role) can play multiple roles (i.e., $\text{played-by}$ is not injective), and also, an object (or a role) may not be a player of any roles at all (i.e., $\text{played-by}$ is not surjective).

Furthermore, it is possible to define delegation from roles to players. For example, suppose we model an employee $e$ as a role of a person $p$, and $\text{sex}$ is an attribute of persons but not of employees. Then $\text{sex}(e)$ would be a type error. We can correct this error by delegating the evaluation of $\text{sex}$ to $\text{played-by}(e)$ [19]. This amounts to replacing $\text{sex}(e)$ by $\text{sex}(\text{played-by}(e))$. Moreover, roles also provide data protection by partitioning the messages received by players. For example, suppose we model an employee $e$ and a student $s$ as two roles of a person $p$, and $\text{student-id}$ is an attribute of students but not of persons or employees. Then $\text{student-id}(e)$ would be a type error. Unless we know that
person \( p \) is a student and access his/her information from the perspective of accessing student information (by \( \text{studentid}(s) \)), \( \text{studentid}(p) \) would also be a type error.

By introducing the role class hierarchy into the object class hierarchy, our role model is formed. These two types of classes are orthogonal to each other, and each of them can be partitioned into subclasses. The difference between role classes and object classes lies in the fact that an instance of an object subclass is identical to (i.e., has the same identifier as) an instance of its superclass but an instance of a role class is different from any instance of its player class. This formalizes the difference with respect to the counting problem mentioned in [24].

Similar to the other properties of a class, the player relationships of a class will be inherited to all its subclasses. This inheritance property of player relationships also holds for role classes, not just object classes. However, since DOOR allows multiple inheritance, there are some cases where we need to conjunct the player constraints of the superclasses of a multiple inherited role class. Consider an interesting example shown in Figure 1\(^2\), where a PERSON (including a CHILD or an ADULT) can be a STUDENT, but only an ADULT can be an EMPLOYEE. Hence a CHILD cannot play the role EMPLOYEE. The role class STUDENT-WORKER is formed by multiple inheritance from both EMPLOYEE and STUDENT classes. The player constraint of this STUDENT-WORKER is computed simply by the conjunction of its superclasses’ player constraints, i.e., a STUDENT-WORKER has to be a PERSON (since both an EMPLOYEE and a STUDENT have to be a PERSON) as well as an ADULT (since an EMPLOYEE has to be an ADULT) and ADULT *is* a PERSON, so the player constraint for STUDENT-

\(^2\)Class names in bold denote object classes, and class names in italic denote role classes.
WORKER is PERSON $\land$ ADULT = ADULT. This means that ADULT can play the role STUDENT-WORKER and CHILD cannot.

Figure 1: An object-role schema which shows that PERSON (including CHILD and ADULT) can at any moment be a STUDENT, but only an ADULT can at any moment be an EMPLOYEE. Therefore, only an ADULT can be a STUDENT-WORKER.

2.2 Role Modeling for Multimedia Objects

Apart from role modeling of general objects in daily life, roles can be used to model the dynamic and multifaceted behavior of multimedia objects.

Images: A raster image object may require different effects or extra properties when it is used in different situations. For example, a bitmapped company logo used on the cover-page of a product manual (a colorful plain logo) may look different when it is used in the company’s letters (a black and white logo with company name and correspondence). Another common example is that a wallpaper raster image may have different properties from the image in iconic form or as a texture-mapped button.

Graphics: The term ‘graphics’ here refers to the concepts that allow the generation of drawings and other images based on formal descriptions, programs or data structures. One of the useful applications of roles to graphics is to model the thematic
role of geometric objects. For instance, a 3D graphical description of a video display can easily play a role as a computer monitor or as a role of a 16"-to-19" home TV depending on the add-on properties.

Audio: The tracks of an audio CD may be classified into different categories according to their content (e.g., soft, light, touchy, rock, etc). Further, segments of a track of a CD can then be further classified. In most of the cases, a track/segment may belong to different classes depending on the effect that has been set (e.g., bass, equalizer settings, volume, surround/theatre sound effect, etc.). With role modeling, we can easily classify or reclassify a song dynamically. This dynamic classification is indeed very useful to support audio data retrieval. For example, you can easily retrieve a track with segments which contain the sound of a canon firing.

Video: A video may have different characteristics in different situations. For example, a video segment may require different sound effects and/or different quality preferences to give an audience different impressions. In particular, an interactive video may have different interaction settings when targeting different kinds of audiences.

Generated Media: Generated media stands for particular computer generated presentations like animation and music. A 3D animation may have totally different effects if it plays different roles which customize its settings (e.g., a varying speed and different settings of quality preferences) and/or add extra properties (e.g., different sound effects). Similarly, MIDI music may have different settings (such as tempo, speed, channels, patches, keys, volume, etc.) For example, soft and touchy guitar folk song music can become rock and heavy with different settings (such as changing
a grand piano patch to an electric guitar patch) and perhaps extra properties (such as more channels or beats).

**Hypertext:** The links and structure of hypertext may be required to be different for different purposes or readers with different background/preferences. In fact, roles can be used to model the different versions of a structured document which allows the document to be targeted for different purposes or readers with different background/preferences.

Figure 2: An object-role schema which shows that a 2D- or 3D- graphic object can at any moment play a role as a 2DImage. For example, a 2D image (2DImage3) of a cube can play a role as a company logo.

To further illustrate the use of objects with roles for multimedia modeling, let us consider the example shown in Figure 2. In this figure, a class **Multimedia Object** is the root class of all multimedia object classes. In particular, it contains two classes, namely **Time-Based Media Object** class and **Graphic Object** class. There are also some
role classes with the player domain constraints [30] specified. The player constraints are used to specify which object(s) is/are qualified to play a particular role. For example, all objects of 3D-Graphic class or its subclasses are qualified to play roles of 2DImage3 class, where the 2DImage3 class captures the properties of the 2D snapshot of 3D-Graphic. With this example, a 2D image of a 3D graphic object can play roles of banner logo, letter logo, or trademark logo (with different characteristics and/or behavior).

Another advantage of having the role playing concept is that it prevents the number of multiple inherited classes from growing exponentially [12]. Moreover, with the schema-level specification of player domain constraints (as denoted by the dashed arrows in Figure 2), our role model supports the concept of intersection classes [25]. That is, DOOR users do not need to define the Company Logo class and its subclasses twice, as subclasses for both 2DImage2 and 2DImage3. This advantage does not exist in conventional object-oriented programming languages such as C++.

3 VIEWS FOR MULTIMEDIA MODELING

A view in conventional databases permits different perspectives on the database that are tailored to specific needs. It hides irrelevant information and restructures data that is retained. To further illustrate this concept, consider the top level of the typical three-level architecture for databases (e.g., [2]) which consists of user views (i.e., versions of the data that are restructured and possibly restricted images of the database as represented at the middle level, i.e., the logical level). In many cases these views are specified as queries (or query programs). These may be materialized (i.e., a physical copy of the view
is stored and maintained) or \textit{virtual} (i.e., relevant information about the view is computed as needed).

Since views are intended to increase the flexibility of relational database systems, it is natural to extend the notion of relational view to the OODB framework. However, unlike relational views, OODB views might redefine the behavior of objects in addition to restructuring their associated types. There are also significant issues raised by the presence of OIDs. For example, to maintain incrementally a materialized view with created OIDs, the linkage between the base data and the created OIDs must be maintained. Furthermore, if the view is virtual, then virtual OIDs need to be specified and manipulated in some way.

In order to apply the concept of views to multimedia data, one has to distinguish two problems: first, the hierarchical organization of interfaces and code in order to ensure certain access to objects, and second, the integration of heterogeneous interfaces of objects for users who are not interested in the detailed content of the objects. An important mechanism to deal with these two problems is the concept of views. Views are important because they provide the right level of abstraction of the resources, data distribution and data representation. Views are also important because they provide different views of the same data, e.g., to realize the often mentioned distinction between internal and external representation of data.

In DOOR, we take an object-oriented database view approach as a basis to multi-view modeling of multimedia objects. View can be created in schema level or instance level. In schema level, view definition is similar to class definition. A view in an object-oriented database allows the user to define new classes by modifying the definitions of existing
classes. The user can choose among attributes or derive new attributes by combining existing ones. The user can also define new methods for the view class. Note that views are different from inheritance in that views allow the user to neglect attributes and modify attributes as well as to add attributes.

For instance, suppose we define views to the object-role schema shown in Figure 2. The resultant schema (with views) is shown in Figure 3. In Figure 3, we can create a view definition (in the schema level) from object class 2D-Graphic, role class 2DImage, or from other view definitions, and so on. View definitions in schema level are in general used as the main content-independent and media-independent user interface of DOOR to present different perspectives or abstractions of multimedia data.

Alternatively, apart from defining views in schema level, views can also be defined at instance level for the sake of flexibility (which is useful for complex applications with large amount of heterogeneous data). For example, consider the example shown in Figure 4. In Figure 4, it shows the use of roles and views together to model the multimedia objects.
Figure 4: Instance Level: Multimedia objects or roles can be visualized from different perspectives or abstraction levels by different views.

In DOOR, multimedia data is modeled at three different layers. Firstly, multimedia data which is of heterogeneous types is stored in media files or device streams are modeled in physical layer (this approach is quite similar to the one mentioned in [26]). Then this data is modeled logical as objects or roles in logical layer. To decide whether the data should be modeled as objects or roles is content dependent. For example, in Figure 4, a 3D DOOR logo and a university bitmap logo are modeled as objects, while a university project group’s letter logo is modeled as a role, which consists of the university name, etc. and is played by a university bitmap logo. This is because a university logo itself
is a logical entity while the letter logo without being played by the university logo (i.e., without linking to the logo) is meaningless and not regarded as a letter logo. Therefore, the existence of a letter logo depends on its player (in this case, the university bitmap logo).

However, one may argue that this can be easily modeled by object composition or part-of relationship. With the specification of role-player domain constraints, we allow any objects to play the letter logo role if they satisfy the player constraints. For example, if the domain constraints is the 2D bitmap class. Then any objects who is of that class is qualified to play the letter logo role. Suppose we have a bitmap object which is in fact a project logo, then it can play the letter logo role anytime to acquire the properties of a letter logo. Finally, this letter logo can have different appearance by creating different views for it. This can be easily done with the help of a visual specification interface. For example, users can visually arrange the layout of the letter logo or add some fancy frame or border lines.

In summary, views are used to provide different views of a multimedia object, and roles are used to model the specialization of a multimedia object. In other words, we regard objects and roles (but not views) as logical entities, and we regard views as presentations of these logical entities.
4 SPECIFICATION OF MULTIMEDIA OBJECTS WITH ROLES AND VIEWS IN DOOR

The purpose of this section is to let the readers have some ideas about what the DOOR code for multimedia applications looks like. For clarity and simplicity, we illustrate only the key constructs related to this paper. These constructs include:

1: \((\text{make } \langle \text{class} \rangle \ '\text{subclass-of } \ldots)\)

2: \((\text{make } \langle \text{virtual-view} \rangle \ '\text{existing-instances } \ldots)\)

3: \((\text{make } \langle \text{materialized-view} \rangle \ '\text{existing-instances } \ldots)\)

4: \((\text{make } \langle \text{an-existing-object-class} \ '\text{slot}1 \ldots)\)

5: \((\text{make } \langle \text{an-existing-role-class} \ '\text{an-existing-instance} \ '\text{slot}1 \ldots)\)

The keyword make is used to create a class (either object class or role class depending on its superclass), a virtual view, a materialized view, an object or a role, as shown in line 1 to line 5 respectively. To illustrate the use of these five makes, let us consider the following DOOR program code which outlines part of the schema shown in Figure 2 and Figure 3.

\((\text{define } \langle 3D\text{-Character} \rangle)\)

\((\text{make } \langle \text{class} \rangle \ '\text{subclass-of } (\text{list } \langle 3D\text{-Graphic} \rangle))\)

'\text{slots } (\text{list } \langle \text{character } '\text{height } '\text{width } '\text{depth}\rangle)

'label "\langle 3D\text{-Character} \rangle")\)

(add-method initialize

(make-method (list \langle 3D\text{-Character} \rangle))

14
(lambda (call-next-method p initargs) (call-next-method)
  (initialize-slots p initargs)))

(define <3D-String>)

(make <class> 'subclass-of (list <3D-Graphic>))
  'slots (list 'string 'height 'width 'depth 'spacing)
  'label "<3D-String>")
(add-method initialize
  (make-method (list <3D-String>)
    (lambda (call-next-method p initargs) (call-next-method)
      (initialize-slots p initargs))))

(define <2D-Bitmap>)

(make <class> 'subclass-of (list <2D-Graphic>))
  'slots (list 'bmpcolors)
  'label "<2D-Bitmap>")
  ...

(define <2D-Image>)

(make <class> 'subclass-of (list <Role>))
  'slots (list 'level ...)
  'label "<2D-Image>")
(add-method initialize
  (make-method (list <2D-Image>)
    (lambda (call-next-method p initargs) (call-next-method)
(define <2D-Image3>)

(make <class> 'subclass-of (list <2D-Image>))

'player-domains (list <3D-Graphic>)

'slots (list 'coords 'ligits ...)

'label "(<2D-Image3>)")

(define <Company-Logo>)

(make <class> 'subclass-of (list <Role>))

'player-domains (list <2D-Image>)

'slots (list 'company-name 'address 'tel)

'label "(<Company-Logo>)")

... ... ...

(define door (make <3D-String> 'string "DOOR" 'height 50 'wide 40 'depth 100 'spacing 5))

(define logo (make <Company-Logo> door 'company-name "DOOR Project Group" ...))

(define mountains (make <2D-Bitmap> 'bmpcolors 256 'filename "MOUNTN.BMP"))

(define mountain-view (make <virtual-view> (list mountains) (list 'bmpcolors 16))

This DOOR program segment shows the definition of three object classes
(<3D-Character>, <3D-String> and <2D-Bitmap>) and three role classes (<2D-Image>,
<2D-Image3> and <Company-Logo>). It then creates an object instance of class
<3D-String> (the last fourth line). Then this created object plays a role which is of
role class <Company-Logo> (the last third line). After that, a 2D bitmap object is created
Besides, in class **Time-Based Media Object** (shown in Figure 2), a method called `mciStrCommand` which can send a string command to the MCI devices (i.e., multimedia devices conformed with the Microsoft MCI device interface) is provided. For example, users can easily play an audio CD by calling the method with argument "play cd audio".

## 5 PROTOTYPING EFFORT

The first DOOR prototype, equiped with multimedia extensions, has been implemented on the Microsoft Windows platform. The PC and Microsoft Windows are chosen because they provide a cost-effective solution and are widely available and accessible to general users. A PC with Microsoft Windows is flexible enough to handle many different formats of multimedia data given the wide variety of cost-effective multimedia accessories. Windows also provides an easy-to-learn and easy-to-use GUI for consistent viewing of multimedia data. Our prototype also makes extensively use of the two advanced features provided by Windows: the **Media Control Interface (MCI)** and **Object Linking and Embedding (OLE)** to handle the multimedia data.

**MCI**: MCI, which is a de facto standard agreed to by many hardware manufacturers, is a software layer which sits between Windows and the hardware devices. Through this interface, modules in DOOR can interact with multimedia devices such as sound cards, video cards and CD-ROMs, which have an MCI driver, through a set of software routines. Any hardware modifications would not affect the data flow of DOOR directly. The only change would be the new device driver installed to
replace the old, unwanted one.

**OLE:** OLE allows Windows applications to link and embed multimedia objects into a multimedia document. The multimedia objects are associated with the specialized media editors that are registered in Windows. For example, we can edit the WAVE data by a professional sound editor and then link it to DOOR as a sound object.

The architecture of the prototype is shown in Figure 5. The entire implementation is based on a Scheme-like interpreter which is written in a modular fashion, such that specific input/output routines or OS dependent calls can be grouped as individual modules and plugged into the core part of the interpreter. Currently three such modules have been implemented. They include the input/output module which handles general input/output such as console or file input/output, the Win API module, which includes the OLE calls and the basic Windows calls such as the manipulation of the Windows interface, and the Win MCI module which handles the system calls to the Windows multimedia extensions.

![Diagram](image.png)

**Figure 5:** Left: Architecture of DOOR with multimedia extensions. Right: A snapshot of a multimedia application prototyped with DOOR.
Each of the heterogeneous multimedia file types that can be handled by the Windows multimedia extensions – WAVE files for sampled sound, MIDI files for sequenced music, and AVI files for Video for Windows movies – is supported by various levels of system calls under the Windows multimedia extensions. In each case, the Windows media control interface (MCI) can treat these files as black boxes such that users can simply hand the path of a multimedia file to MCI, tell MCI to play it, and understand almost nothing about what is really going on in the process. In fact, it is rather trivial to integrate DOOR with the MCI devices. All communication from DOOR to the MCI interface is handled through the \texttt{mciSendCommand} function provided by the Windows multimedia extensions.

Figure 5 Right shows a snapshot of a multimedia application based on the prototype. With this example application, users can retrieve a WAVE object and mark a particular segment of the object as a particular role. Users may also retrieve a MIDI object and specialize it by playing a role with different tempo and perhaps different patches (e.g., change a piano patch to an organ patch) or by adding different background beats. Similarly, users can retrieve an image (no matter whether it is a 2D raster image or a 3D vector image), do some transformation or add extra properties, and then save it as a role. Users can then reuse these roles for multimedia document composition. For example, you may want to retrieve a movie about a waterfall as a role of a hypertext movie to highlight the wonderful nature of the earth. Users can then select a suitable (music) role as background music to match the theme.
6 RELATED WORK

Systems for storing and retrieving multimedia information have been proposed and developed for different application areas in the past decade. Most of these systems are prototypes tailored for specific application areas and so provide domain-specific functionalities and features. However, one can also identify certain features essential for multimedia databases in general [10].

One of the earliest uses of multimedia information systems comes from office automation, whose goal is to provide flexible retrieval from large repositories of multimedia documents. Three examples are MINOS [8], MULTOS [4], and DIAMOND [23]. The former two systems offer the following features: modeling of complex object structure, content-based retrieval, and differentiation of internal (logical) and external (layout) representations. The later one puts less emphasis on content-based retrieval. Instead, it uses simple file system-like folders for retrieval and stresses flexible handling of network communication (i.e., transmission of documents). Alternatively, there are other multimedia document systems, called image databases, which stress the importance of image processing. The extent to which image processing should be provided by multimedia databases in general is an open issue. However restricted content-based retrieval for images, e.g., by some form of similarity measure, is a possibility. For example, REDI [7] allows simple content retrieval for LANDSAT images. This system combines systems for image processing, image recognition, and database management.

Multimedia database systems replicate, to some extent, the features of multimedia document systems, e.g., modeling of complex object structure, content-based retrieval,
and differentiation of internal and external representation. The difference is that these systems also stress typical database system issues such as concurrency control, access control, recovery, versioning, and management of storage devices. An example of a multimedia database system is ORION’s Multimedia Information Manager (MIM) [27, 26].

One interesting aspect of the MIM is that all devices are represented as objects. For I/O devices these objects further specify how information is presented and captured. MIM support images, voice and analog video. The problem of version control has also been investigated.

Among multimedia databases, most of the work done up to now favors the object-oriented approach and suggests the use of an OODBMS, e.g., [1, 5, 9, 10, 13, 15, 17, 27]. With minor divergences, the constructs which should be supported by the multimedia data model, such as those in [11, 17, 27], are widely accepted. As storage requirements of multimedia data are large, techniques to minimize storage space on the physical level (e.g., compression techniques) and the logical level (e.g., data sharing through aggregation) are considered important. Querying and browsing of multimedia information as well as control of data capture and presentation must be provided. Additionally, multimedia database management systems provide traditional database functions such as support for multiple views, content-based retrieval (which is problematic for image and audio, but at least discussed in several lists of requirements), concurrency control, transaction management, security, backup and recovery. Finally, version control is also considered important.

A similar idea to the meta-object protocol approach used in DOOR has been employed in the VODAK system proposed by researchers in GMD-IPSI (e.g., [1, 15, 20]). While the main mission of the VODAK system is on tailoring the data model (also called an
"open" data model) by means of metaclasses, DOOR stresses its semantics cleanliness and extensibility from the programming language point of view. Therefore, we have chosen Scheme as our prototyping language (because of its semantic clarity and simplicity). The actual implementation of DOOR is heavily influenced by the implementation techniques described in [14].

View support in conventional databases is wellknown (e.g., [2]). However, no one has applied this concept extensively in multimedia applications although its importance has been realized and mentioned in [16]. Similarly, role modeling in the object-oriented approach has been used successfully in modeling dynamic, evolving, and multi-faceted applications [18, 29, 28, 30, 33], but no one has considered and applied this concept to multimedia modeling.

In summary, we have presented the ideas of using roles and views to model multimedia objects which have not been discussed before. Moreover, the extensible data model and architecture in DOOR are also different from most of the existing approaches mentioned above. We believe that our flexible approach is at least more useful in developing different data models or architectures for multimedia applications.

7 CONCLUSIONS AND FUTURE WORK

We have presented an approach of extending the data model of an object-oriented database programming language called DOOR, by means of meta-object protocol to implement the classes for multimedia objects and the Microsoft MCI interface for accessing the multimedia devices. We have described the fundamental of role modeling for daily-life entities and
proposed it for multimedia applications. In particular, the different multimedia presentations (mostly with different characteristics) of a real-world entity can be modeled as a single object which plays different roles. We discussed the role modeling of different types of multimedia objects such as audios or videos, etc. Moreover, we have also proposed different ways of using views to provide the right level of abstraction of the resources and data representation. Finally, we present the extensible architecture of the DOOR prototype for multimedia applications.

Our ongoing research work includes the indexing of time-dependent media data in DOOR. This involves the integration of the role retrieval mechanism with binary large objects (BLOBs) [6] (the preliminary result of this has been described in [32]), and the investigation of appropriate indexing strategies which yield fast data retrieval. Moreover, we also continue to develop the visual interface for the multimedia prototype built with DOOR. This includes the visual specification of the spatial (layout), composition, and temporal (e.g., synchronization) aspects of the multimedia objects. Finally, as multimedia technology has become important in geographic applications, we plan to develop a multimedia GIS (geographic information system) according to the role model described in [29] once we have an efficient object store.

References


