Multimedia Information Management in an Object-Oriented Database System

Darrell Woelk
Won Kim

Microelectronics and Computer Technology Corporation
3500 West Balcones Center Drive
Austin, Texas 78759

Abstract

This paper describes the implementation of the Multimedia Information Manager (MIM) in the ORION object-oriented database system which is operational at MCC. We describe design objectives and implementation techniques that have satisfied the design objectives. Our design objectives include extensibility, flexibility and efficiency in supporting the capture, storage, and presentation of many types of multimedia information.

We have achieved extensibility by providing an object-oriented framework for multimedia information management. The framework consists of definitions of class hierarchies and a message passing protocol for not only the multimedia capture, storage, and presentation devices, but also the captured and stored multimedia objects. Both the class hierarchies and the protocol may be easily extended and/or modified by system developers and end users as they see fit. We have satisfied flexibility by supporting a variety of ways in which the end users may control the capture and presentation of multimedia information. Our implementation has achieved storage efficiency by using a technique for sharing storage blocks among multiple versions of a multimedia object, while achieving data transfer performance by directly interfacing the MIM to certain low level components of the ORION storage subsystem.

1. Introduction

The management of multimedia information such as images and audio is becoming an important feature of computer systems. Multimedia information can broaden the bandwidth of communication between the user and the computer system. Although the cost of the hardware required for the capture, storage, and presentation of multimedia data is decreasing every year, the software for effectively managing such information is lacking. Future database systems must provide this capability if we are to be able to share large amounts of multimedia information among many users.

In our earlier work [WOEL86], we identified two types of requirements which multimedia applications impose on a database system. One is the requirement for a data model that allows a very natural and flexible definition and evolution of the schema that can represent the composition of and the complex relationships among parts of a multimedia document. Another is the requirement for the sharing and manipulation (storage, retrieval, and transmission) of multimedia information. In [WOEL86] we concluded that an object-oriented approach would be an elegant basis for addressing all data modelling requirements (the first type) of the multimedia applications.

Subsequently, we developed an object-oriented data model by extracting a number of common concepts from existing object-oriented programming languages and systems, and then enhancing them with a number of additional concepts, including versions and predicate-based access to sets of objects. The data model, described in detail in [BANE87], has been implemented in a prototype object-oriented database system, which we have named ORION. ORION is implemented in Common Lisp [STEE84], and runs on a Symbolics 3600 Lisp Machine[SYMB85]. ORION adds persistence and sharability to the objects created and manipulated by object-oriented applications from such domains as artificial intelligence, computer-aided design, and office information systems. Important features of ORION include transaction management, versions [BANE87], composite objects [BANE87], and multimedia information management. The Proteus expert system [PETR86] developed by the MCC Artificial Intelligence Program has recently been modified to interface with ORION. The MUSE multimedia system [LUTH87] developed by the MCC Human Interface Program will be integrated with ORION in the near future.

The focus of this paper is multimedia information management in ORION. In particular, we will describe the design objectives for the Multimedia Information Manager (MIM) component of ORION, and the implementation approach we have taken to satisfy the design objectives. We have three major design objectives for supporting the capture, storage, and presentation of many types of multimedia information: extensibility, flexibility and efficiency. The most important requirement for extensibility (generalizability and modifiability) is the ability for the system developers and end users to extend the system, by adding new types of devices and protocols for the capture, storage, and presentation of multimedia information. To satisfy this requirement, we have implemented the MIM as an extensible framework explicitly using the object-oriented concepts. The framework consists of definitions of class hierarchies and a message passing protocol for not only the multimedia capture, storage, and presentation devices, but also the captured and stored multimedia objects. Both the class hierarchies and the protocol may be easily extended and/or modified by system developers and end users as they see fit.

Our implementation provides efficiency both in storage utilization and data transfer performance. We achieve storage efficiency by using a technique for sharing storage blocks among multiple versions of a multimedia object, and data transfer performance by directly interfacing the MIM to certain low level components of the ORION storage subsystem. We have not completed exhaustive testing of the performance of the system, but initial tests have supported our expectations of good data transfer performance in the system.

This paper makes two significant contributions. One is the description of our implementation that satisfies the flexibility and efficiency requirements of multimedia information management. Another is the illustration it provides of an object-oriented implementation of a framework for multimedia information management. The framework may be viewed as one further proof of the power of the object-oriented paradigm. Further, to the extent that an object-oriented implementation of the framework was motivated by the requirement to make a major component of a database system highly extensible, our approach may also provide an additional insight to the current research in extensible database systems [CARE86]. One additional contribution of this paper, although perhaps not as significant as the other two, is the identification of design requirements for a multimedia information manager.

In Section 2, we will review the object-oriented concepts which are the basis for the ORION data model. Section 3 will discuss our design objectives for the Multimedia Information Manager. Section 4 will sketch the ORION database system architecture, to provide a concrete context for discussions of the MIM implementation. In Section 5 we will describe the implementation of the MIM. The description will include the multimedia class definitions, the multimedia message passing protocol, and the aspects of the implementation which provide flexibility, efficient data storage, and efficient data transfer. Section 6 will summarize and conclude the paper.

2. Review of Object-Oriented Concepts

Existing object-oriented systems exhibit significant differences in their support of the object-oriented paradigm; [STEF86] provides an excellent account of different variations of the object concepts. In this section, to establish our terminology, we review the basic object concepts which we have selected for our data model from existing object-oriented programming languages and systems [GOLD81, BOBR83, BOBR85, LMI85, MAIE86]. This section has been extracted from our paper on the ORION data model in [BANE87].

Objects, Attributes (Instance Variables), Methods, and Messages

In object-oriented systems, all conceptual entities are modeled as objects. An ordinary integer or string is as much an object as is a complex assembly of parts, such as an aircraft or a submarine. An object consists of some private memory that holds its state. The private memory is made up of the values for a collection of attributes. The value of an attribute is itself an object, and therefore has its own private memory for its state (i.e., its attributes). A primitive object, such as an integer or a string, has no attributes. It only has a value, which is the object itself. More complex objects contain attributes, through which they reference other objects, which in turn contain attributes.

The behavior of an object is encapsulated in *methods.* Methods consist of code that manipulate or return the state of an object. Methods are a part of the definition of the object. However, methods, as well as attributes, are not visible from outside of the object. Objects can communicate with one another through messages. Messages constitute the public interface of an object. For each message understood by an object, there is a corresponding method that executes the message. An object reacts to a message by executing the corresponding method, and returning an object.

Classes

If every object is to carry its own attribute names and its own methods, the amount of information to be specified and stored can become unmanageably large. For this reason, as well as for conceptual simplicity, 'similar' objects are grouped together into a *class*. All objects belonging to the same class are described by the same attributes and the same methods. They all respond to the same messages. Objects that belong to a class are called *instances* of that class. A class describes the form (attributes) of its instances, and the operations (methods) applicable to its instances. Thus, when a message is sent to an instance, the method which implements that message is found in the definition of the class.

Class Hierarchy and Inheritance

Grouping objects into classes helps avoid the specification and storage of much redundant information. The concept of a *class hierarchy* further reduces information redundancy. A class hierarchy is a hierarchy of classes in which an edge between a pair of nodes represents the IS-A relationship; that is, the lower level node is a *specialization* of the higher level node (and conversely, the higher level node is a *generalization* of the lower level node). For a pair of classes on a class hierarchy, the higher level class is called a *superclass*, and the lower level class a *subclass*. The attributes and methods (collectively called properties) specified for a class are inherited (shared) by all its subclasses. Additional properties may be specified for each of the subclasses. A class inherits properties only from its immediate superclass. Since the latter inherits properties from its own superclass, it follows by induction that a class inherits properties from every class in its *superclass chain.*

Domains of Attributes

In object-oriented systems, the *domain* (which corresponds to data type in conventional programming languages) of an attribute is a class. The domain of an attribute of a class C may be explicitly bound to a specific class D. Then instances of the class C may take on as values for the attribute instances of the class D as well as instances of subclasses of D.

Class Lattice, Multiple Inheritance, and Name-Conflict Resolution

In many object-oriented systems (including ORION), a class can have more than one superclass, generalizing the class hierarchy to a lattice (directed acyclic graph). In a class lattice, a class inherits properties from each of its superclasses. This feature is often referred to as *multiple inheritance* [LMI85, STEF86].

The class lattice simplifies data modeling and often requires fewer classes to be specified than with a class hierarchy. However, it gives rise to conflicts in the names of attributes and methods. One type of conflict is between a class and its superclass (this type of problem also arises in a class hierarchy). Another is among the superclasses of a class; this is purely a consequence of multiple inheritance.

Name conflicts between a class and its superclasses are resolved in all systems we are aware of, and in ORION, by giving precedence to the definition within the class over that in its superclasses. The approach used in many systems, and in ORION, to resolve name conflicts among superclasses of a given class is the *superclass ordering*. If an attribute or a method with the same name appears in more than one superclass of a class C, the one chosen by default is that of the first superclass in the list of (immediate) superclasses of C, which the application will have specified.

3. Design Objectives

Multimedia applications place a set of strong requirements on a database system. In [WOEL86] we described the data modeling and functional requirements. In this section we will discuss additional requirements concerned with extensibility, flexibility, and efficiency. Our implementation of the MIM within ORION has satisfied these requirements. Our implementation of the ORION object-oriented data model has satisfied the data modeling and functional requirements enumerated in [WOEL86].

3.1 Extensibility

Extensibility is required to support new multimedia devices and new functions on multimedia information. For example, a color display device may be added to a system with relative ease, if at a high level of abstraction the color display can be viewed as a more specialized presentation device for spatial multimedia objects than a more general display device which is already supported in the system. The color display device may be further specialized by adding windowing software, and the windows can in turn be specialized to create new display and input functionality. Future database system should support the presentation of multimedia information on these presentation devices as described in [CHRI86a]. Further, database systems must also support the capture of multimedia information using such capture devices as cameras and audio digitizers.

It is also important to be able to add new multimedia storage devices, or to change the operating characteristics of storage devices. For example, read-only CD ROM [CDRO86] disks and write-once digital optical disks [CHRI86b] are both storage devices having desirable characteristics for the storage of certain types of multimedia information. The integration of these hardware devices into a system is becoming easier due to standard disk interfaces such as SCSI [KILL86]. A natural framework for logically accessing these devices must be provided by the database system.

Even multimedia information stored on magnetic disk may require special formatting for efficiency in storage and access. For example, an image may be stored using approximate geometry as described in [OREN86]. This storage format allows the expression of powerful spatial queries. The new storage format and the new query functionality can be defined as specializations of the more general capability for storing and presenting images.

3.2 Flexibility

The MIM must provide for the storage of both *spatial* and *linear multimedia objects*, both the *persistent* and *non-persistent presentation* of multimedia objects, and control of both the capture and presentation of multimedia objects. These three aspects of flexibility in multimedia information management will be discussed below. Section 5.3 describes how our implementation of the MIM provides flexibility.

First, for the purposes of capture, storage and presentation, the MIM must support both linear multimedia objects and spatial multimedia objects [WOEL87]. *Spatial multimedia objects* are multimedia objects with a logical internal format which is spatially oriented, for example, a bit-mapped image. The application may identify a specific rectangular portion of an image for presentation by specifying the upper-left corner, height, and width of a rectangle. The MIM translates these values into physical offsets in the disk storage. *Linear multimedia objects* are multimedia objects which have a logical internal format which is sequential, such as text or audio. A specific audio passage can be presented by specifying an offset and a length in logical units, such as seconds. Some multimedia objects, such as an animated bit-mapped image, can be categorized as both spatial and linear.

Second, the MIM must support two types of transfer of multimedia objects from the database: persistent and non-persistent, depending on whether or not the multimedia object remains in the system memory for manipulation by an application after its transfer. An example of *non-persistent presentation* is the playing of audio data; the audio data is transferred from the database directly to the audio hardware. The process is initiated by the application, but the MIM handles the buffering and movement of data. An example of *persistent presentation* is the display of a bit-mapped image in a window on the screen of the workstation. When the application requests that a selected image be displayed in a specific window, the image is transferred from the database to a specific area in the memory space of the application. When this area of memory is mapped to the workstation screen, the image will be displayed. Following any modifications made to this area of memory during a transaction, the application can transfer the object back to the database.

Third, once the transfer of data has begun, the application should be able to control the presentation or capture of multimedia data. For example, during the playback of audio, the application should be able to cause the playback to pause, continue, fast-forward, fast-backward, play faster, play slower, rewind, and stop.

3.3 Data Storage Efficiency

Multimedia information is in general very large, and keeping multiple copies of large objects such as images and audio can lead to a serious waste of secondary storage media. In ORION, a multimedia object is stored in a number of physical storage blocks, such that versions of the multimedia object will share those storage blocks that contain common information, and new storage blocks are allocated only for those portions of the multimedia object that hold different information. As versions of the multimedia object are updated or deleted, storage blocks that are no longer needed are automatically returned to free space for re-allocation to other multimedia objects. Section 5.4 describes this implementation in detail.

3.4 Data Transfer Efficiency

Multimedia applications require the transfer of large amounts of data between capture devices, storage devices, and presentation devices. In some cases, this information will be transferred from a storage device to a presentation device without ever being written to the system memory. In many cases, however, the digitized multimedia object will be buffered in the system memory. The MIM must optimize the performance of transfer of multimedia objects by eliminating unnecessary copying and buffering of data within the system. Section 5.5 discusses how we achieve this objective.

4. Overview of the ORION Architecture

Figure 1 shows a high level block diagram of the ORION architecture. The message handler receives all messages sent to the ORION system. The messages include user-defined messages, access messages, and system-defined functions. A *user-defined message* is a message to a method that the user defines and stores in ORION. An ac-

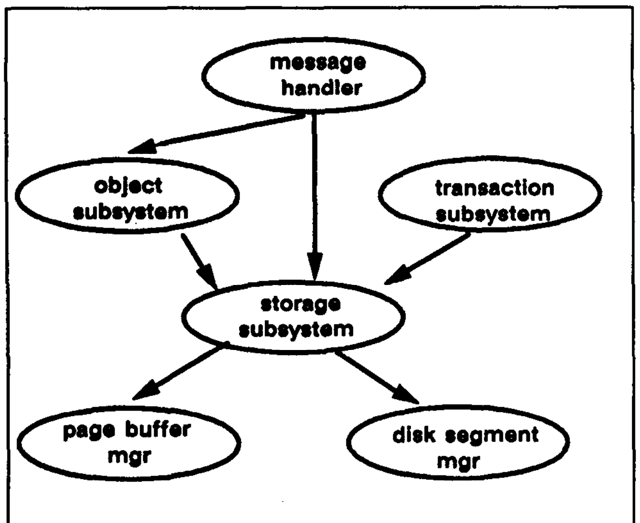

Figure 1. ORION Architecture

cess message is one that retrieves or updates the value of an attribute of a class. System-defined functions include all ORION functions for schema definition, creation and deletion of instances, transaction management, and so on.

The object subsystem of ORION handles all access to objects in the system. Functions provided by the object subsystem include identifier-based and predicate-based query processing, version management, and multimedia information management.

The storage subsystem provides access to objects on the disk. Objects are moved from the disk to page buffers. Two of the sub-modules within the storage subsystem are also shown in Figure 1. The disk segment manager manages the allocation and deallocation of segments of pages on the disk. The page buffer manager moves pages of data to and from the disk. It maintains a page table which keeps track of the disk pages present in memory. As we will discuss later, the MIM is directly interfaced with the disk segment manager and the page buffer manager to allocate and deallocate storage blocks, and to transfer data to and from the database.

The transaction subsystem provides a concurrency control mechanism to protect database integrity while allowing the interleaved execution of multiple concurrent transactions. It also accumulates a log of changes to objects within a transaction. The log is used to backout a transaction, or to recover from system crashes in the middle of a transaction.

5. Implementation of the Multimedia Information Manager

We have analyzed scenarios for the capture, storage, and presentation of many types of multimedia information and have generalized these into a framework of classes and a message protocol for interaction among instances of these classes. This framework is highly extensible, since it is based on the class lattice and message passing concepts of the object-oriented paradigm. In Section 5.1 we will describe the multimedia classes which are defined for ORION. Section 5.2 will present the message passing protocol among instances of these classes, in terms of the capture, storage, and presentation of a bit-mapped image. Then in Sections 5.3, 5.4, and 5.5, we will discuss how our implementation meets the objectives for flexibility, efficient data storage, and efficient data transfer, respectively.

5.1 Multimedia Class Definitions

Multimedia information is captured, stored, and presented in ORION using lattices of classes which represent capture devices, storage devices, captured objects, and presentation devices. However, each instance of one of the device classes represents more than just the identity of a physical device as we will describe in the following sections.

5.1.1 Presentation-Device Classes

The MIM uses ORION classes to represent presentation devices available on the system. An instance of the presentation-device, however, represents **more than just a specific physical presentation device.** Each instance also has attributes which further specify, for example, where on the device a multimedia object is to be presented and what portion of a multimedia object is to be presented. These predefined presentation-device instances can be stored in the database and used for presenting the same multimedia object using different presentation formats. Methods associated with a class are used to initialize parameters of a presentation device and initiate the presentation process. The class lattice for the presentation devices is shown in Figure 2. The shaded classes are provided with ORION. Other classes in the lattice are shown to indicate potential specializations for other media types by specific installations.

Figure 3 shows details of a portion of the class lattice for the presentation-device class. The screen-window subclass represents a window on a workstation screen that is to be used to display an image. An instance of the screen-window class has the attributes win-upper-left-x, win-upper-left-y, win-width, and win-height that represent where the window is positioned on the workstation screen. It inherits from the spatial-pres-device class the attributes upper-left-x, upper-left-y, width, and height that specify the rectangular area of an image that is to be displayed. This screen-window instance can be stored in the database and used whenever a specific rectangular area of an image is to be displayed in a specific position on the workstation screen.

5.1.2 Capture-Device Classes

ORION objects provide an abstraction for interfacing with different types of capture devices; however, as with the presentation devices, ORION methods do not take the place of low-level real-time device drivers. Figure 4 shows the class lattice for capture devices. The shaded classes are ones provided with ORION. Other classes are potential specializations for other media types. An instance of the capture-device class represents **more than just a specific physical capture device,** as described below.

Figure 5 shows details of a portion of the class lattice for the capture-device class. The spatial-capture-device class includes attributes which describe the shape and size of the rectangular area of a multimedia object to be captured. This area is described by the attributes upper-left-x, upper-left-y, width, and height. For example, if an instance has the values 0, 0, 300, 300, for these attributes, respectively, only the pixels in a 300 x 300 rectangle in the upper left-hand corner of the image will be captured and stored in the specified captured-object. The image-capture-device class has attributes, cam-width and cam-height, which describe the shape and size of the image provided by the actual camera device. As with presentation devices, pre-defined instances of the capture-device class can be stored in the database and used for the capture of a multimedia object using different capture formats.

5.1.3 Captured-Object, Storage-Device, and Disk Stream Classes

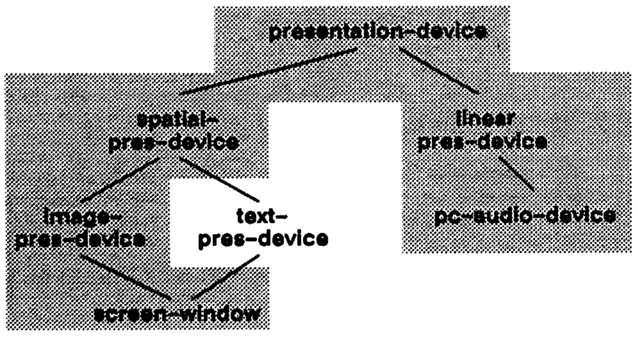

Figure 2. Presentation Device Class Lattice

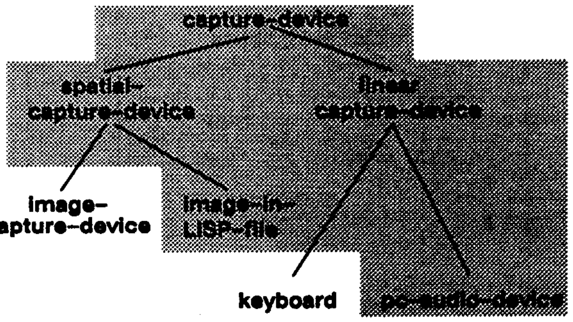

Figure 4. Capture Device Class Lattice

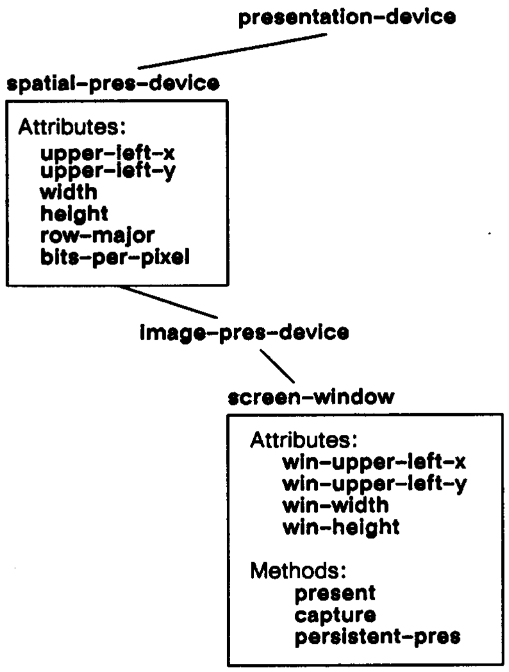

Figure 3. Details of Presentation Device Class Lattice

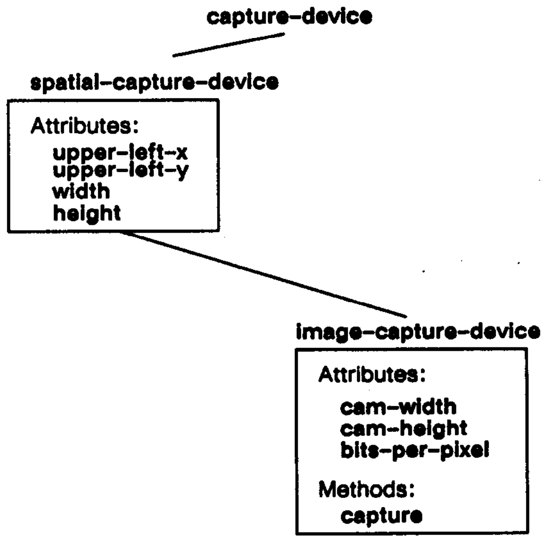

Figure 5. Details of Capture-Device Class Lattice

We have adapted the storage and access techniques for multimedia objects in ORION from previous research into the manipulation of long data objects [HASK82]. Every multimedia object stored in ORION is represented by an instance of the class captured-object or one of its subclasses. Figure 6 illustrates the class lattice for captured objects. The captured-object class defines an attribute named storage-object which has as its domain the class storage-device. The class lattice for storage devices and for disk streams are also shown in Figure 6. Transfer of data to and from storage-device instances is controlled through disk-stream instances. The shaded classes in Figure 6 are provided with ORION. Other classes in the lattice indicate potential specializations.

Figure 7 shows details of a portion of the class lattice for the captured-object class, the storage-device class, and the disk-stream class. Each instance of the captured-object class has a reference to a storage-device instance stored in its storage-object attribute. The spatial-captured-object class has attributes which further describe spatial objects. The attributes width and height describe the size and shape of the spatial object. The attribute row-major indicates the order in which the transformation from linear to spatial coordinates should take place. The attribute bits-per-pixel specifies the number of bits stored for each pixel in the spatial object.

As with presentation-device and capture-device instances, each mag-disk-storage-device instance represents **more than just the identity of a specific physical magnetic storage device.** Each instance further describes the portion of the device which is occupied by a particular multimedia object. The mag-disk-storage-device class has the block-list attribute which contains the block numbers of the physical disk blocks that make up a multimedia object. The allocated-block-list attribute specifies the blocks in the block-list which were actually allocated by this mag-disk-storage-device instance. The min-object-size-in-disk-pages attribute specifies the number of disk pages that should be allocated each time data is added to a multimedia object. The seg-id attribute specifies the segment on disk

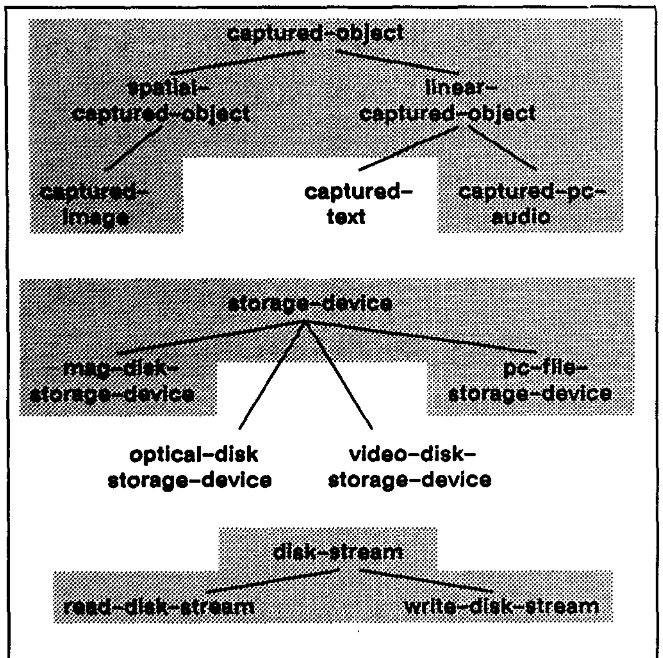

Figure 6. Captured-Object, Storage-Device, and Disk Stream Class Lattices

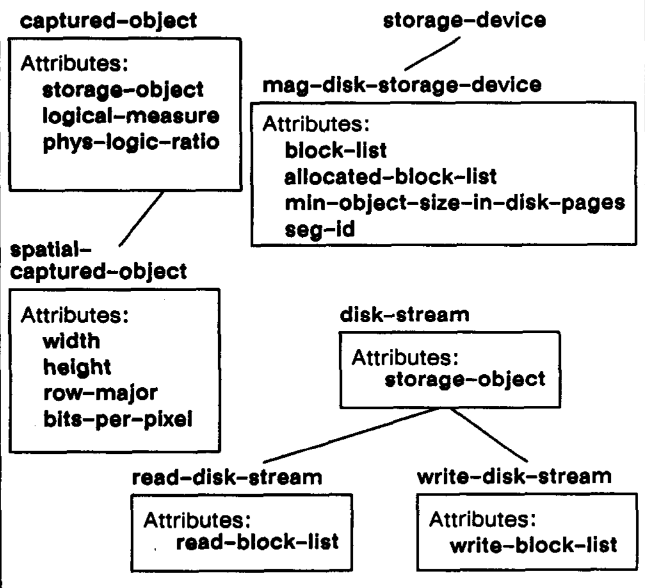

Figure 7. Details of Captured-Object, Storage Device, and Disk Stream Class Lattices

from which disk pages are to be allocated. The use of these attributes will be described in more detail in Section 5.4.

An instance of the read-disk-stream class is created whenever a multimedia object is read from disk. The read-disk-stream instance has a storage-object attribute which references the mag-disk-storage-device instance for the multimedia object. It also has a read-block-list attribute which maintains a cursor indicating the next block of the multimedia object to be read from disk. An instance of the write-disk-stream class is created whenever data is written to a multimedia object. In addition to the storage-object attribute, the write-disk-stream instance has a write-block-list attribute that is updated as disk blocks are written with data.

5.2 Multimedia Message Passing Protocol

This section will describe the message protocol for the presentation and capture of multimedia information using the ORION classes described in the previous section. The protocol will be discussed by using the example of a bit-mapped image; however, the protocol is similar for many types of multimedia information.

5.2.1 Presentation

Figure 8 shows an instance of a class called vehicle which has been defined by an application program. It also shows instances of the image-pres-device, captured-image, read-disk-stream, and mag-disk-storage-device classes described earlier. The arrows represent messages sent from one instance to another instance. The vehicle instance has an image attribute that specifies the identity of a captured-image instance that represents a picture of the vehicle. It also has a display-dev attribute that specifies the identity of an image-pres-device instance. This image-pres-device instance has attributes (described in Section 5.1.1) pre-defined by the user that specify where the image is to be displayed on the screen and what part of the image should be displayed. When the vehicle instance receives the picture message, the picture method defined for the class

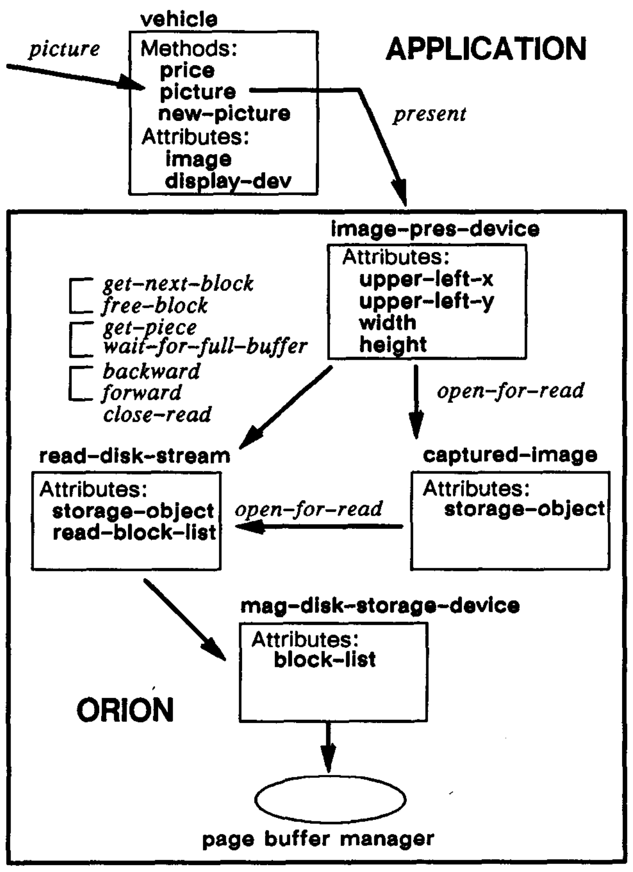

Figure 8. Message Passing Protocol for Presentation of Multimedia Information

vehicle will send a present message shown below to the specified image-pres-device instance.

(*present* **presentation-device** captured-object [physical-resource])

The physical-resource parameter above specifies a physical resource, such as the address of the video frame buffer for image presentation. (Throughout this paper, we will use italics to denote the message name, bold-face for the object receiving the message, non-bold face for the parameters of a message, and square brackets for optional parameters. Further, because of space limitations, we will explain the meaning of the message parameters only enough for the reader to follow the message protocol. Classes specified in these messages will always be the most general class acceptable. In the example above, the captured-object parameter will have a value which is an instance of the captured-image class, a subclass of the captured-object class.)

The present method of the image-pres-device class transfers image data from the captured-image instance and displays the image on a display device. The image-pres-device instance has attributes (described in Section 5.1.1) which specify the rectangular portion of the image to be displayed. It translates these rectangular coordinates into linear coordinates to be used for reading the image data from disk. It then initiates the reading of data by sending the following message to the captured-image instance:

(*open-for-read* **captured-object** [start-offset])

The start-offset is an offset in bytes from the start of the multimedia object.

The captured-image instance then creates a read-disk-stream instance and returns its identity to the image-pres-device instance. The image-pres-device will then send a get-next-block message to the read-disk-stream:

(*get-next-block* **read-disk-stream**)

The read-disk-stream instance calls the ORION page buffer manager to retrieve a block of data from disk. The address of the ORION page buffer containing the block is returned. The image-pres-device instance will transfer the data from the page buffer to a physical presentation device, and then send a free-block message to the read-disk-stream, to free the page buffer.

(*free-block* **read-disk-stream**)

A cursor will also be automatically incremented so that the next get-next-block message will read the next block of the multimedia object. When the data transfer is complete, the image-pres-device sends a *close-read* message to the read-disk-stream instance.

An alternate protocol using *get-piece* and *wait-for-full-buffer* messages is also provided by the read-disk-stream. These messages allow the sender to allocate its own buffer and have the read-disk-stream instance copy data from the disk block into that buffer. This protocol causes an extra copy operation but is valuable where the application wishes to maintain its own copy of the data.

New methods may be written to extend the system so that the media type of the presentation-device and the captured-object may be different. For example, an audio-pres-device instance presenting a captured-text instance could result in text-to-speech translation.

5.2.2 Capture

Figure 9 describes the message protocol for capturing multimedia information, which closely corresponds to the presentation message protocol. Once again, we will discuss an example using a bit-mapped image but the protocol is similar for other types of multimedia information. Classes specified for the parameters in the messages defined in this section will always be the most general class acceptable. In our example, the captured-object parameter in these messages will have a value which is an instance of the captured-image class, a subclass of the captured-object class.

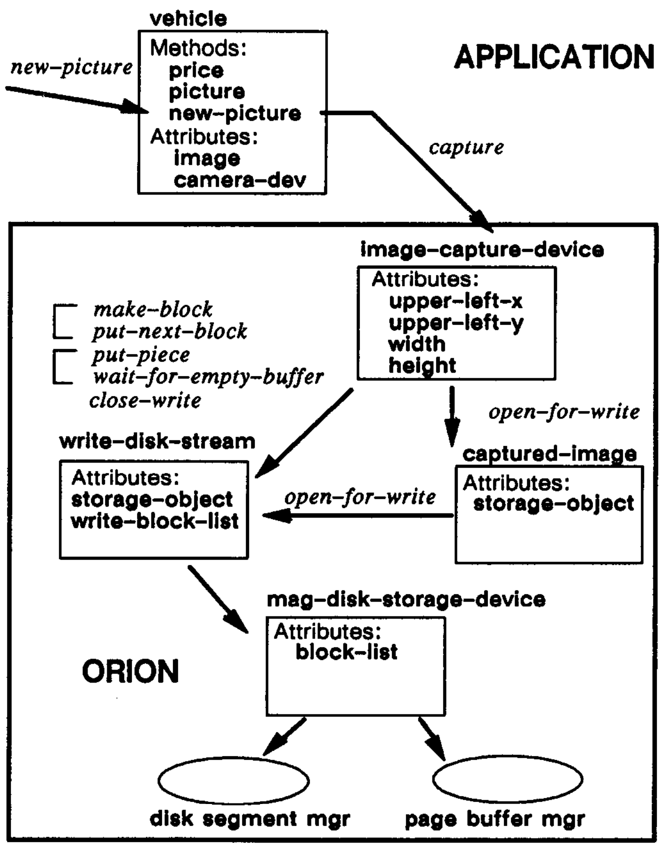

Figure 9. Message Passing Protocol for Capture of Multimedia Information

If the new-picture message is sent to the vehicle instance, the new-picture method will send a capture message to the image-capture-device instance specified in its camera-dev attribute.

(*capture* **capture-device** captured-object [physical-resource])

The captured-object parameter is an instance of the captured-object class or one of its subclasses. The physical-resource parameter specifies a physical resource, such as the address of the video frame grabber for image capture.

Information concerning the camera settings and image processing necessary prior to storing the image is stored in attributes of the image-capture-device instance. The image-capture-device initiates the writing of data by sending the following message to a captured-image instance:

(*open-for-write* **captured-object** [start-offset] [delete-count])

The start-offset is an offset in bytes from the start of the multimedia object. The delete-count indicates the number of bytes to delete beginning at the start-offset.

When a captured-image instance receives the open-for-write message, it creates an instance of the write-disk-stream class, and returns the identity of the instance to the capture-device instance. The capture-device instance will then send the following message to the write-disk-stream:

(*make-block* **write-disk-stream**)

The make-block method allocates a block of pages on disk, if necessary, by calling the disk segment manager. A page buffer in memory is also allocated for the block and the address of this page buffer is returned. The image-capture-device will then transfer data from a physical capture device to the specified page buffer.

Next, the image-capture-device will send, as many times as necessary, a put-next-block message to cause the contents of the page buffer to be written to disk. The message specifies, in the length parameter, the number of bytes which have actually been written to the block.

(*put-next-block* **write-disk-stream** length)

Once data transfer has been completed, a **close-write** message is sent. The message causes the write-disk-stream instance to update the mag-disk-storage-device instance to indicate the new disk pages which have been added to the multimedia object. Further, it deallocates any disk pages which were freed by the delete-count parameter in the open-for-write messages by calling the disk segment manager.

An alternative protocol using *put-piece* and *wait-for-empty-buffer* is also provided by the write-disk-stream. These messages allow the sender to allocate its own buffer and have the write-disk-stream instance copy data from that buffer into the disk block. This protocol causes an extra copy operation but, as in the read operation, it is valuable where the application wishes to maintain its own copy of the data.

As in the case of presentation, new methods may be written to extend the system so that the media type of the capture-device and the captured-object may be different. For example, an audio-capture-device instance and a captured-text instance may be used to implement speech recognition.

5.2.3 Manipulating Captured-Objects

We have implemented a number of other messages for captured-object instances. Because of space limitations, we will discuss only a few of them here. To create a new version of a captured-object, the following message can be sent:

(*make-captured-object-version* **captured-object**)

The method executed uses the make-version message provided by the ORION object subsystem [CHOU86] to make a new version of the captured-object. It then makes a new version of the mag-disk-storage-device instance the old version references, and modifies the new version of the captured-object instance to reference the new version of the mag-disk-storage-device instance. The two versions of the captured object share common disk blocks, as described in Section 5.4.

A copy of a captured-object instance can be created without creating a new version, by sending the following message:

(*copy-captured-object* **captured-object**)

The method executed creates a copy of the captured-object instead of a new version. As in the make-captured-object-version message, however, a new version of the referenced mag-disk-storage-device instance is created so that the two captured-object copies may share common physical disk blocks.

A captured-object instance can be deleted using the following message:

(*delete-captured-object* **captured-object**)

To delete some range of bytes in a multimedia object, the following message is sent to a captured-object instance:

(*delete-part-of-captured-object* **captured-object**
start-offset [delete-count])

The start-offset is an offset in bytes from the start of the multimedia object. The delete-count indicates the number of bytes to delete beginning at the start-offset.

5.3 Flexibility of Multimedia Data Presentation / Capture

To allow the application to identify a portion of a multimedia object in logical units (such as seconds), we have defined the captured-object class with attributes which describe the general translation from the storage representation to the presentation representation. As shown in Figure 7, they include logical-measure and physical-to-logical-ratio. The logical-measure attribute contains the definition of the logical unit of measurement (such as seconds, frames, etc.). The physical-to-logical-ratio attribute indicates the ratio of the physical length to the logical length (such as bytes of digitized audio per second).

Some types of presentation devices, such as the image-pres-device class, are capable of persistent presentation of captured-objects. The persistent presentation option on a presentation device is set by the following message:

(*persistent-pres* **presentation-device** [set])

If the set parameter is T, the persistent presentation option is invoked. The option does not actually take effect until the next *present* message is received.

After an image has been displayed and modifications made to the image, the image-pres-device instance can be sent a *capture* message to write the copy of the image to a captured-image instance:

(*capture* **presentation-device** captured-object
physical-resource)

The physical resource parameter is mandatory in this case and must be the identity of a physical resource returned by a prior *present* message.

To provide explicit control over the direction of presentation of multimedia information, our implementation has the presentation-device instance explicitly control the cursor that is maintained by the read-disk-stream. For example, in response to a user request to move ahead 10 seconds in an audio message, the presentation-device instance will translate 10 seconds into a byte count, and send the following message:

(*forward* **read-disk-stream** count)

The cursor can be decreased by sending the following message:

(*backward* **read-disk-stream** [count])

The count parameter specifies the number of bytes to subtract from the cursor.

5.4 Multimedia Data Storage Efficiency

We achieve efficient storage of multimedia data by having multiple versions of a multimedia object share common storage blocks. Our current implementation is limited to magnetic disk storage, and as such, the algorithm we describe here is in terms of a mag-disk-storage-device.

When a multimedia object is created or updated, space is allocated for the new data in blocks of N disk pages. Existing pages of data on disk are never overwritten. As Figure 7 illustrates, each mag-disk-storage-device instance maintains a list of block-entries, block-list. Each block-entry is of the form (block-id start-offset length), which represents the identity of the disk block, the start-offset within the block, and the length of the data in the block, respectively.

Figure 10 illustrates 3 captured-object instances and the mag-disk-storage-device instances they reference. The solid arrows represent references from captured-object instances to mag-disk-storage-device instances. The dashed arrows represent version relationships between objects. Originally, captured-object instance #1 was created with a reference to mag-disk-storage-device instance #1. A total of 3000 bytes of multimedia data were then written to disk blocks 1, 2, and 3. At some later time, captured-object instance #2 was created as a new version of captured-object instance #1. It was then modified by deleting 1200 bytes of data, beginning at a start-offset of 900 bytes, and inserting 1000 bytes, beginning at a start-offset of 900 bytes. The 1000 new bytes were all written to the newly allocated block 4. Then, at some later time, captured-object instance #3 was created as a copy of #1 (but not as a new version of #1). It was then modified by inserting 800 bytes at a start-offset of 400. These 800 bytes are written into the newly allocated block 5. Now, both mag-disk-storage-device instance #2 and mag-disk-storage-device instance #3 have mag-disk-storage-device instance #1 as a parent-version.

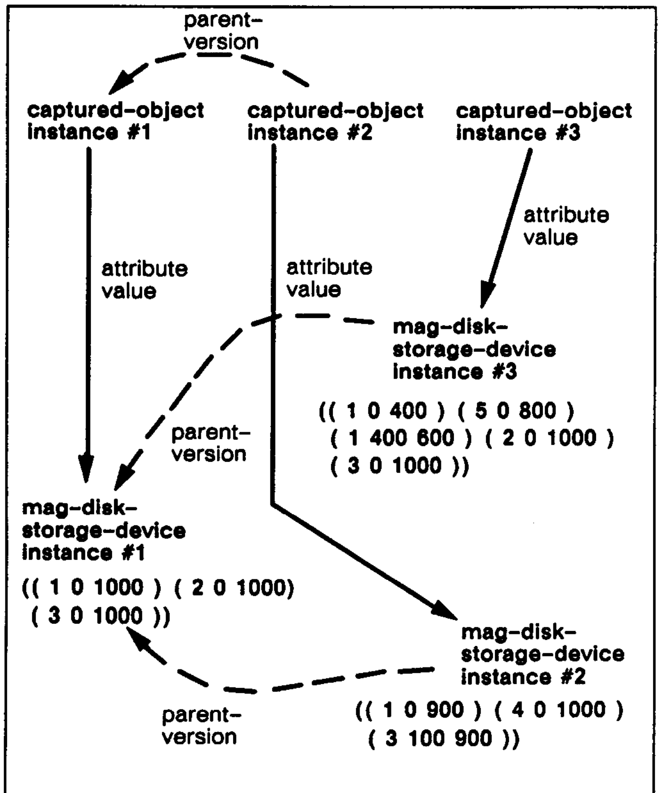

Figure 10. Use of Orion Version Management for Reducing Copies of Multimedia Data

The ORION object subsystem constructs a version hierarchy as new versions of a captured-object are created. The object subsystem also enforces the constraint that an object cannot be modified once a new version has been derived from it. Whenever disk blocks are removed from the block-list of a mag-disk-storage-device instance during an update or a delete operation, the MIM uses the version hierarchy to find disk blocks which are no longer referenced from any mag-disk-storage-device instances and which therefore can be returned to free space. The following algorithm which we have implemented is a simplified version of the algorithm proposed in [CARE86].

1. **Deallocating a Disk Block X during an Update of Version A**
   a. If X was allocated by A, free X and exit.
   b. If X was allocated by an ancestor version of A, and if the parent version of A has not been deleted, exit without freeing X.
   c. If X was allocated by an ancestor version of A, and if the parent version of A has been deleted, first search the sibling versions of A for references to X. If no references are found, search the descendant versions of each deleted ancestor version for X, until a non-deleted ancestor version is found. Then search the non-deleted ancestor version for X. If X is found during any of these searches, exit without freeing X. If X is not found, free X and exit.

2. **Deallocating a Disk Block X during the Deletion of Version A**
   a. Same as 1b.
   b. Search for X in the first non-deleted descendant version on every path rooted at A. If X is found, exit without freeing X.
   c. same as 1c.

Figure 11 illustrates two examples of version hierarchies where versions of a mag-disk-storage-device instance have been previously deleted. Versions which have already been deleted are shown in boxes. Assume that in Figure 11a, we are about to delete version V3 and that V3 references only Block X, where Block X was not allocated by V3. Rule 2a states that since V1 is the parent version of V3 and V1 has not been deleted, we know that Block X will still be used by V3. We do not need to search any further.

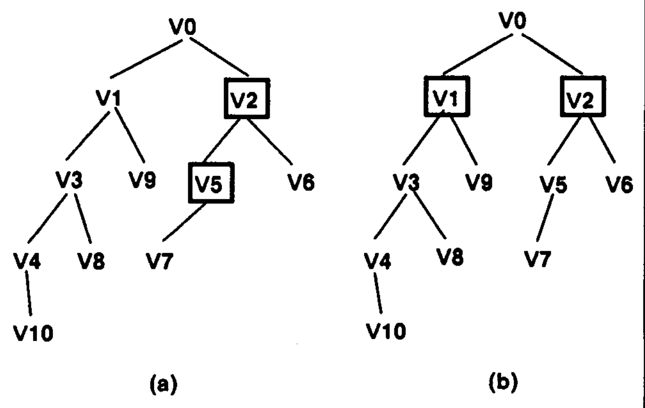

Figure 11. Deletion of Versions

In Figure 11b, again we are about to delete version V3 and we assume that V3 references only Block X, where Block X was not allocated by V3. Since V1 has been deleted, Rule 2a does not apply. Rule 2b states that we must search V4 and V8 (but not V10) for Block X. If we assume that there is no reference to Block X in V4 or V8, Rule 2c then states that we must search V0 and V9 for references to Block X. If no reference to Block X is found in V0 and V9, we can deallocate Block X on disk so that it may be reallocated for use in another multimedia object.

5.5 Multimedia Data Transfer Efficiency

We have optimized data transfer efficiency in ORION by eliminating unnecessary copying of multimedia data as it is transferred between magnetic disk storage and presentation devices in the system. We accomplish this by giving presentation-device instances and capture-device instances the capability to directly manipulate data in the ORION page buffers, thus eliminating the need to copy the data from the page buffers. Further, as in [HASK82], rather than logging the before or after image of a multimedia object (which is potentially very large), we log only the mag-disk-storage-device instance which has an attribute describing the disk blocks containing the multimedia object. All disk blocks allocated by a mag-disk-storage-device instance during a transaction are automatically deallocated if the transaction aborts. All disk blocks deallocated by a mag-disk-storage-device instance during a transaction are not actually deallocated until the transaction commits.

6. Concluding Remarks

In this paper, we described our implementation of the Multimedia Information Manager (MIM) for the ORION object-oriented database system. We first reviewed the basic object concepts which are the basis of the ORION data model. We then described our design objectives for the support of multimedia databases. These design objectives include extensibility, flexibility, and efficiency in supporting many types of multimedia information.

We then described in detail our implementation of the MIM and how it met these design objectives. A framework representing multimedia capture, storage, and presentation devices has been implemented using ORION classes. This framework may be specialized by system developers and end users to extend the functionality of the MIM. A message passing protocol was defined for the interaction among instances of these classes. This protocol may also be specialized.

We discussed our implementation of a technique for reducing unnecessary copies of multimedia data on disk storage by having multiple versions of a multimedia object share common disk blocks. We also presented our implementation of a technique for efficiently transferring multimedia data in the system by having the the methods associated with the multimedia classes directly interface with low level functions of the ORION storage subsystem.

One contribution of this paper is the description of our implementation that satisfies the flexibility and efficiency requirements of multimedia information management. Another contribution is the lucid illustration of an object-oriented implementation of a framework for multimedia information management. The framework may be viewed as one further proof of the power of the object-oriented paradigm. Since the framework is what makes the MIM highly extensible, our approach may also provide an additional insight to the current research in extensible database systems.

Using the multimedia classes and message passing protocol described in this paper, we have implemented capture, storage, and presentation of bit-mapped images and audio with ORION on the Symbolics LISP Machine. We were able to use the Symbolics Flavors window system for displaying images but we did not wish to add special-purpose camera or audio-recording hardware to the Symbolics for capturing images, capturing audio, and presenting audio. We did have access over a local area network to other systems which had this type of multimedia capability. Therefore, we created new classes to represent remote capture and presentation devices by further specializations of the capture-device and presentation-device classes. The present and capture methods for these classes were specialized in some cases to move captured data across the local area network and in other cases to actually capture multimedia data remotely, store it in the remote device, and present it remotely under the control of ORION.

Acknowledgements

We are grateful to John Cater, Bill Luther, and Jim Avery of the MCC Human Interface Program for providing expertise in the area of multimedia technology and human/machine interaction. We also thank Jay Banerjee and Hong-Tai Chou of the ORION project for their help in integrating multimedia functionality with ORION.

References

[BANE87] Banerjee, J., H. T. Chou, J. Garza, W. Kim, D. Woelk, N. Ballou, and H. J. Kim. "Data Model Issues for Object-Oriented Applications," to appear in *ACM Trans. on Office Information Systems*, April 1987.

[BOBR83] Bobrow, D.G.. and M. Stefik. *The LOOPS Manual,* Xerox PARC, Palo Alto, CA., 1983.

[BOBR85] Bobrow, D.G., K. Kahn, G. Kiczales, L. Masinter, M. Stefik, and F. Zdybel. *CommonLoops: Merging Common Lisp and Object-Oriented Programming,* Intelligent Systems Laboratory Series ISL-85-8, Xerox PARC, Palo Alto, CA., 1985.

[CARE86] M. Carey, D. DeWitt, J.E. Richardson, and E.J. Shekita. "Object and File Management in the EXODUS Extensible Database System," *Proc. 12th Intl Conf. on Very Large Data Bases,* August 1986, pp. 91-100.

[CDRO86] *CD ROM, The New Papyrus,* edited by S. Lambert and S. Ropiequet, Microsoft Press, Redmond, WA., 1986.

[CHOU86] Chou, H.T., and W. Kim. "A Unifying Framework for Versions in a CAD Environment," in *Proc. Intl Conf. on Very Large Data Bases*, August 1986, Kyoto, Japan.

[CHRI86a] Christodoulakis, S., F. Ho, and M Theodoridou. "The Multimedia Object Presentation Manager of MINOS: A Symmetric Approach," *Proc. ACM SIGMOD Intl Conf. on the Management of Data*, May 1986, pp. 295-310.

[CHRI86b] Christodoulakis, S., and C. Faloutsos. "Design and Performance Considerations for an Optical Disk-Bases, Multimedia Object Server," *IEEE Computer*, December 1986, pp. 45-56.

[GOLD81] Goldberg, A. "Introducing the Smalltalk-80 System," *Byte*, vol. 6, no. 8, August 1981, pp. 14-26.

[HASK82] R. Haskin and R. Lorie. "On Extending the Functions of a Relational Database System," in *Proc. ACM SIGMOD Intl Conf. on Management of Data*, June 1982, pp. 207-212.

[KILL86] Killmon P. "For Computer Systems and Peripherals, Smarter is Better," *Computer Design*, January 15, 1986, pp. 57-70.

[LMI85] *ObjectLISP User Manual,* LMI, Cambridge, MA, 1985.

[LUTH87] Luther W., D. Woelk, and M. Carter. "MUSE: Multimedia User Sensory Environment," to appear in *IEEE Knowledge Engineering Newsletter*, February 1987.

[MAIE86] Maier, D., Stein, J., Otis, A., and Purdy, A. "Development of an Object-Oriented DBMS," Oregon Graduate Center: Technical Report CS/E-86-005, April 1986.

[OREN86] Orenstein J. "Spatial Query Processing in an Object-Oriented System," *Proc. ACM SIGMOD Intl Conf. on the Management of Data*, May 1986, pp. 326-336.

[PETR86] Petrie, C., D. Russinoff, and D. Steiner. "Proteus: A Default Reasoning Perspective," *Fifth Generation Systems Conf.,* National Institute for Software, Washington, D.C., October, 1986.

[STEE84] Guy L. Steele Jr., Scott E. Fahlman, Richard P. Gabriel, David A. Moon, and Daniel L. Weinreb, "Common Lisp," *Digital Press*, 1984.

[STEF86] Stefik, M., and D.G. Bobrow. "Object-Oriented Programming: Themes and Variations," *The AI Magazine*, January 1986, pp. 40-62.

[SYMB85] Symbolics Inc., "User's Guide to Symbolics Computers," *Symbolics Manual #* 996015, March 1985.

[WOEL86] D. Woelk, Won Kim, and W. Luther. "An Object-Oriented Approach to Multimedia Databases," *Proc. ACM SIGMOD Intl Conf. on the Management of Data*, May 1986, pp. 311-325.

[WOEL87] D. Woelk, W. Luther, and W. Kim. "Multimedia Applications and Database Requirements," to appear in *Proc. IEEE Computer Society Symposium on Office Automation*, April 1987.