**Multimedia Information Management**

**in an Object-Oriented Database System**

Darrell Woe/k

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 da-tabase system which is operational at MCC. We describe design objectives and implementation techniques that have satisfied the design objectives. Our design objectives in-clude extensibility, flexibility and efficiency in supporting the capture, storage, and presentation of many types of multi-media information.

We have achieved extensibility by providing an object-ori-ented 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 hier-archies 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 multi-media 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 im-ages and audio is becoming an important feature of com-puter systems. Multimedia information can broaden the bandwidth of communication between the user and the com-puter 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.

Permission to copy without fee all or part ‘of this material is granted provided that the copies are not made or distributed for direct commercial advantage,the VLDB copyright notice and the title of the publication and its date appear,and notice is given that copying is by permission of the Very Large Data Base Endow-ment. To copy otherwise.or to republish, requiresa fee and/or spe cid permissionfrom the Endowment.

In our earlier work (WOELBB] , we identified two types of requirements which multimedia applications impose on a da-tabase 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 docu-ment. Another is the requirement for the sharing and ma-nipulation (storage, retrieval, and transmission) of multime-dia information. In [WOELBB] we concluded that an object-oriented approach would be an elegant basis for addressing all data modelling requirements (the first type) of the multl-media applications.

Subsequently, we developed an object-oriented data model by extracting a number of common concepts from existing object-oriented programming languages and sys-tems, 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-ori-ented database system, which we have named ORION. ORION Is implemented in Common Lisp [STEE84], and runs on a Symbolics 3800 Lisp MachinejSYMB851. ORION adds persistence and sharability to the objects created and ma-nipulated 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 [BANE871 , composite objects [BANE87], and multimedia information manage-ment. The Proteus expert system [PETRBB] developed by the MCC Artificial Intelligence Proaram has recently been modified to interface with ORION. The MUSE multimedia sys-tem ILUTH871 develooed bv the MCC Human Interface Pro-gram-will be integrated with ORION in the near future.

The focus of this paper is multimedia information man-agement 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 re-quirement for extensibility (generallrability and modifiability) is the ability for the system developers and end users to ex-tend the system, by adding new types of devices and proto-cols for the capture, storage, and presentation of multimedia information. To satisfy this requirement, we have lmple-mented the MIM as an extensible framework explicitly using the object-oriented concepts. The framework consists of definitions of class hierarchies and a message passing proto-col for not only the multimedia capture, storage, and presen-tation devices, but also the captured and stored multimedia objects. Both the class hierarohies and the protocol may be

Proceedings of the 13th VLDB Conference, Brighton 1987 319

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 stor-age efficiency by using a technique for sharing storage blocks among multiple versions of a multimedia object, and data transfer performance by directly interfacing the MINI to certain low level components of the ORION storage subsys-tem. We have not completed exhaustive testing of the per-formance of the system, bdt initial tests have supported our expectations of good data transfer performance in the sys-tem.

This paper makes two significant contributions. One is the description of our implementation that satisfies the flexi-bility and efficiency requirements of multimedia information management. Another is the illustration it provides of an

object-oriented implementation of a framework for multime-dia information management. The framework may be viewed as one further proof of the power of the object-ori-ented paradigm. Further, to the extent that an object-ori-ented implementation of the framework was motivated by the requirement to make a major component of a database svstem highly extensible, our approach may also provide an bbditional%sight to the current research in extensible data-base svstems fCARE881. 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 con-cepts which are the basis for the ORION data model. Section 3 will discuss our design objectives for the Multimedia Infor-mation Manager. Section 4 will sketch the ORION database system architecture, to provide a concrete context for dis-cussions 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 mes-sage passing protocol, and the aspects of the implementa-tion which provide flexibility, efficient data storage, and effi-cient data transfer. Section 8 will summarize and conclude the paper.

2. Review of Object-Oriented Concepts

Existing object-oriented systems exhibit significant dif-ferences in their support of the object-oriented paradigm: (STEF881 provides an excellent account of different vari-ations 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, LMl85, MAIE86J. This section has been extracted from our paper on the ORION data model in [BANE87].

Objects, Attributes (Instance Variables), Meth-ods, 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 con-tain 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 ob-ject. 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 c/ass. 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 in-stances, and the operations (methods) applicable to its in-stances. Thus, when a message is sent to an instance, the method which implements that message is found in the defi-nition of the class.

Class Hierarchy and Inheritance

Grouping objects into classes helps avoid the specifica-tion and storage of much redundant information. The con-cept of a class hierarchy further reduces information redun-dancy. A class hierarchy is a hierarchy of classes in which an edge between a pair of nodes represents the IS-A rela-tionship; 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 super-class, 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 corre-sponds to data type in conventional programming lan-guages) 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 sub-classes 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 su-perclasses. This feature is often referred to as multiple in-heritance (LMl85, STEF86].

The class lattice simplifies data modeling and often re-quires fewer classes to be specified than with a class hierar-chy. However, it gives rise to conflicts in the names of at-tributes 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,

Proceedings of the 13th VLDB Conference, Brighton 1987 320 by giving precedence to the definition within the class over that in its superclasses. The approach used in many sys-tems, and in ORION. to resolve name conflicts among super-classes of a given class is the superclass ordering. If an at-tribute 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) super-classes of C, which the application will have specified.

3. Design Objectives

Multimedia applications place a set of strong require-ments on a database system. In [WOEL86] we described the data modeling and functional requirements. In this section we will discuss additional requirements concerned with ex-tensibility, 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 enu-merated in [WOEL66].

Figure 1. ORION Architecture

cess messege 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 dbjects in the system. Functions provided by the object sub-system include identifier-based and predicate-based query processing, version management, and multimedia informa-tion 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. lt 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 stor-age blocks, and to transfer data to and from the database.

The transaction subsystem provides a concurrency con-trol mechanism to protect database integrity while allowing the interleaved execution of multiple concurrent transac-tions. 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. Sec-tion 5.2 will present the messaQe 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 pre-sented 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 .l Presentation-Device Classes

The MIM uses ORION classes to represent presentation devices available on the system. An instance of the presen-tation-device, however, represents more than just a rpe-cific 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 por-tion of a multimedia object is to be presented. These pre-defined presentation-device instances can be stored in the database and used for presenting the same multimedia ob-ject using different presentation formats. Methods associ-ated with a class are used to initialize parameters of a pres-entation 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 speciali-zations 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 imaqe. 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 in-stance 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. FiQure 4 shows the class lattice for capture devices. The shaded classes are ones provided with ORION. Other classes are potential spe-cializations for other media types. An instance of the cap-ture-device class represents man 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 speci-fied 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 differ-ent capture, formats.

5.1.3 Captured-Object, Storage-Device, and Disk

Stream Classes

322 proceedings of the 13th VLDB Conference, Brighton 1987

Figure 2. Presentation Device Class Lattice

presentation-device

spatial-pres-device

image-pres-device

\

screen-window

Attributes: win-upper-left-x win-upper-left-y win-width win-height

Methods: present capture

persistent-pres

Figure 3. D;e;Fe of Presentation Device Class

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 mul-timedia object stored in ORION is represented by an instance of the class captured-object or one of its subclasses. Figure 8 illustrates the class lattice for captured objects. The cap-tured-object class defines an attribute named storage-ob-ject 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 in-stances. The shaded classes in Figure 6 are provided with ORION. Other classes in the lattice indicate potential spe-cializations.

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-ob-image- 0 capture-device 1,keyboard

Figure 4. Capture Devioe Cl8ss Lattlco

capture-device spatial-capture-device

Attributes:

upper-le t-x upper-le It y

width height image-capture-device Figure 5. D<e; $eof Capture-Device Class ject 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 ob-jects. 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 spa-tial object.

As with presentation-device and capture-device in-stances, each mag-disk-storage-device instance repre-sents 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