An Object-Oriented Approach to Multimedia Databases

Darrell Woelk*
Won Kim
Willis Luther

Microelectronics and Computer Technology Corporation
9430 Research Blvd
Austin, Texas 78759

ABSTRACT

This paper identifies data modelling and data access and sharing requirements which multimedia applications impose on a database system It shows the capabilities of an object-based data model and indicates extensions which are needed to meet the data modelling aspects of these requirements A logical implementation of the operations on the model is described The model generalizes the notions of instantiation and generalization in the standard object-oriented paradigm, and augments it with the notions of aggregation and relationships which are specialized for a multimedia application environment Objects may exist in aggregation hierarchies which provide the capability to integrate diverse types of multimedia information such as text, sound, bit-mapped images, and complex graphics drawings Objects may also be linked through other user-defined relationships to capture such application functions as voice annotation and referencing of one document by another Using this model, the semantics of aggregation and relationships in a multimedia application environment can be understood and efficiently supported by a database system

1 0 Introduction

This paper will identify the requirements for a multimedia database system and propose an object-oriented database approach which meets these requirements Multimedia data is defined as data in the form of text, sound, and images (digitized images and complex vector graphics drawings) In [WOEL85] we studied the logical and physical characteristics of multimedia data, as well as their usage in various application areas

A number of systems have been developed which integrate graphical representation with a DBMS to broaden the bandwidth of information transfer between the computer and the user [FOGG84, WILS80, WONG82] Other systems provide the function of storing and retrieving digitized images from a database [SLOA82] Some systems have the capability to store and retrieve documents composed of text, images, and audio [FORS84, CHRI84, POGG85, YANK85, SAKA85]

In the implementation of a number of graphics-based systems, such as the Xerox Smalltalk system [GOLD81] and the Apple MacIntosh system [APPL83], the object-oriented programming paradigm has proven to be highly valuable In the object-oriented paradigm, a real-world entity is modeled as an instance (object) of a class which has a number of attributes (properties) and operations (methods) applicable to the objects A class, and therefore an object, may inherit properties and methods from other classes Thus the fundamental data modelling concepts instantiation (an object is an instance of a class) and generalization (a class inherits properties and methods from other classes) are built into the object-oriented paradigm Further, the inheritance mechanism makes it possible for applications to define new classes and have them inherit properties from existing classes, this makes the applications easily extendible

The programming language Simula [BIRT73] introduced the concept of a class, which later formed the basis of Smalltalk [GOLD81] Flavors [WEIN81], LOOPS [BOBR83], and Object-LISP [LMI85] provide object-oriented features for the LISP programming environment Frame-based rule systems [MINS75, FOX85] in artificial intelligence also embody the concept of objects There are some recent proposals for systems which merge the programming and data languages to form an object-oriented programming environment that is supported by a database system Examples of this trend include GemStone [COPE84], IRIS [BEEC83], OPAL [AHLS84], and TAXIS [MYLO80]

Our research at MCC into the compound document management application of multimedia databases has led us to an object-oriented database approach for multimedia applications Objects in a multimedia database have various properties and participate in a number of relationships with other objects We have found that by generalizing the basic notions of objects (instantiation and generalization), and augmenting them with the notions of aggregation (an object contains other objects) and relation-

* On assignment from NCR Corporation

ships (an object is related to another object), we can capture the data modelling requirements of multimedia applications Using this model, the semantics of aggregation and relationships in a multimedia application environment can be understood and efficiently supported by a database system

In this paper, we first establish data modelling and data access and sharing requirements of multimedia applications We then propose an object-oriented data model which meets the data modelling aspects of these requirements, and we describe a logical implementation of the operations on the data model The remainder of this paper is organized as follows In Section 2 we describe an example of a multimedia application drawn from a multimedia document management system under development We present the functional requirements of a multimedia database system in Section 3 Section 4 describes our object-oriented data model for the support of multimedia applications In Section 5 we present a logical implementation of the model, in terms of operations on a logical database design supporting the data model Section 6 provides a summary and conclusion

2 0 Description of a Representative Multimedia Application

Figure 1 presents a typical document which might be generated with a multimedia document manager It contains three text paragraphs, a drawing in the middle of the page which has been created by combining primitive graphics shapes along with text, and an image at the bottom of the page which is a photograph stored as a bit-mapped image The highlighting of the "From D Woelk" in Figure 1 will be a blinking highlight to draw attention The small "speaker" icon next to the line indicates that there is a voice message associated with this line The highlighted box in the drawing is another blinking highlight The large arrow next to the box indicates that there is more detail concerning this object in another document

The document in Figure 1 could have been created with a simple typewriter and some creative pasting However, the power of the multimedia document system does not come from the creation of static documents which look nice Rather, it will come from creating a body of information which can be shared by many users Paper documents are a model of information transfer with which people are familiar today and, therefore, they make a reasonable metaphor for information transfer This will change as the bandwidth of information transfer between humans and computers increases because of improved computer capability for processing (storage, reproduction, and recognition) of speech, images, and knowledge The combination of these will create new techniques for human-to-human and human-to-computer information transfer The data model proposed in this paper will support these new techniques

Figure 2 presents a logical view of the information in the document in Figure 1 Notice that the information forms an *aggregation* (or *part-of,* or *containment) hierarchy* [SMIT77] This should be intuitively obvious because even in this simple example, a person will naturally separate the body of a memo from the header information The example of a book with a table of contents, chapters, sub-chapters, and an index is also an obvious example Note that there are seven types of information represented in Figure 2

1 The connected Circles represent the logical parts of the document They are connected to form the aggregation hierarchy Unshaded Circles represent what we might consider to be part of a template for a Memo For example, all Memos contain a Memo-Header, a Body, and a Memo-Trailer This can be thought of as representing the schema for a memo The schema can be used as a template for creating new Memos by enforcing constraints on the structure of the Memo (a Memo can only contain a Memo-Heading, Body, and Memo-Trailer), on the types of attributes which are allowable (a Memo-Logo can only contain Text), and on the relationships among parts of the Memo (a From-Line can be annotated by a Voice Message) Furthermore, the schema can be used for phrasing high level queries to access all or portions of one or more Memos ("Retrieve MEMO where MEMO FROM-NAME = D Woelk")

2 Shaded Circles represent the portion of the aggregation hierarchy which is unique to this particular Memo For example, the Paragraphs contained in the Body of the Memo are unique to this particular Memo

3 The Squares represent unformatted intrinsic data such as text, digitized images, complex vector graphics drawings, or sound These Unshaded Squares represent data which is present for every Memo For example, the word "FROM " will occur in every Memo

4 The Shaded Squares represent text, image, or audio data which we might consider to be unique to this particular Memo For example, the words "D Woelk" would only occur on this Memo

5 The Ovals represent information about the data in the Shaded and Unshaded Squares as well as information about the Shaded and Unshaded Circles of the aggregation hierarchy For example, all of the text in the Memo-Header is to be displayed in Font-Size = 14 The Memo node, however, also has a Font Size = 12 which is to be used for all text in the Memo except where over-ridden lower in the aggregation hierarchy In this example, the Memo-Header Font-Size = 14 will over-ride the Memo Font-Size = 12

6 The Rectangles represent operations on the data in the Squares, Circles, and Ovals of the aggregation hierarchy For example, the Rectangle labelled SEND in Figure 2 contains procedural operations for sending the document to the people listed in the To-Line and in the CC-List

7 The dotted lines represent relationships among nodes which are not represented in the aggregation hierarchy For example, the dotted line from the first Group node under the Drawing node to the Workstation Manual node indicates that a workstation manual document contains more details concerning the abstract object represented by this group of graphic shapes

3 0 Functional Requirements of Multimedia Applications

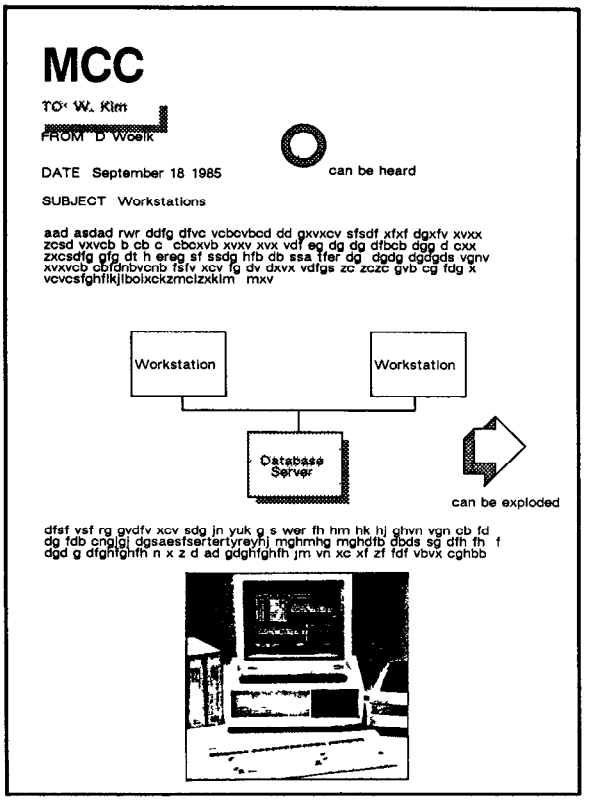

FIGURE 1
Example Document

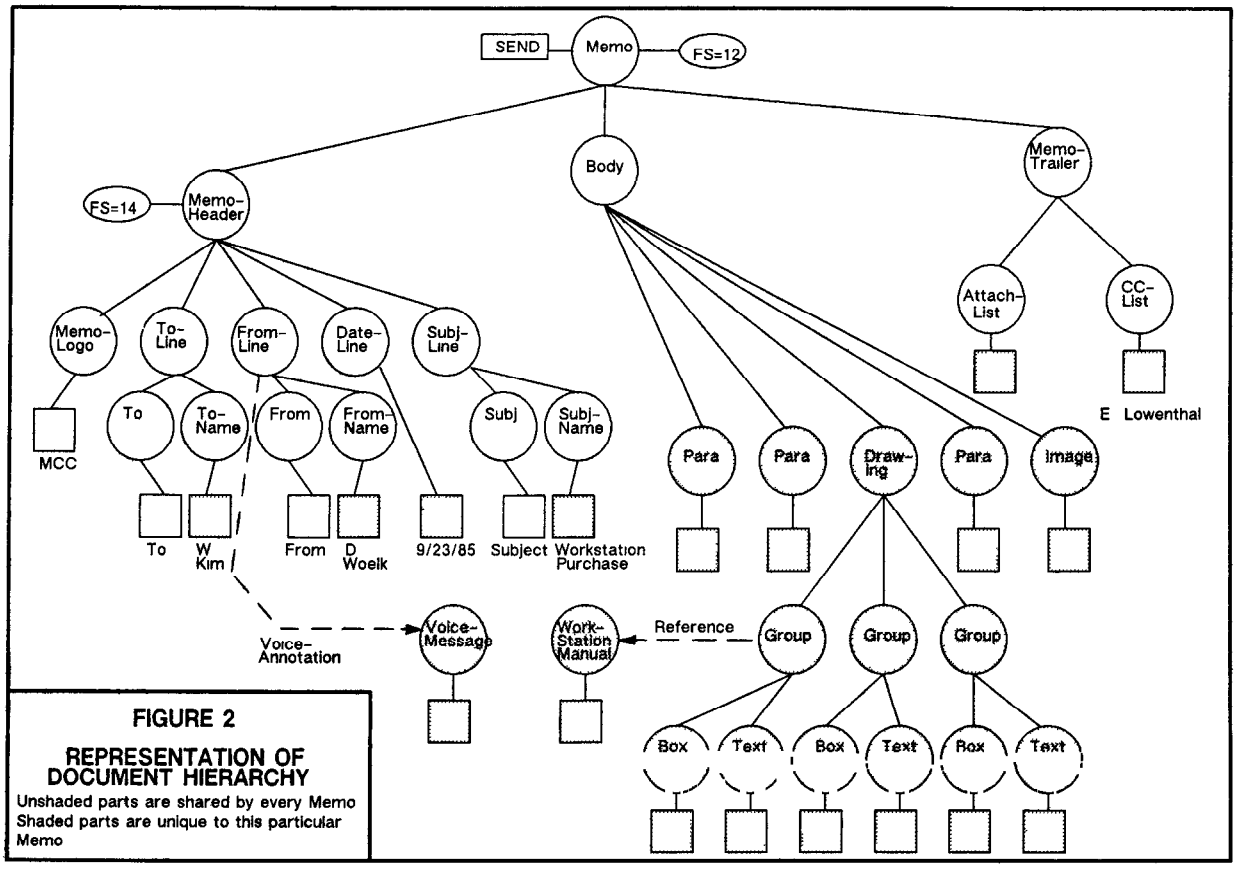

FIGURE 2

REPRESENTATION OF DOCUMENT HIERARCHY

Unshaded parts are shared by every Memo
Shaded parts are unique to this particular Memo

The example application in Section 2, although rather simple, exposes a spectrum of requirements, many of which present problems for traditional database systems These requirements are listed below

1 The aggregation hierarchy presented in Figure 2 must be supported by the database system Nodes at each level of the hierarchy represent an abstraction of all of the nodes below This abstraction mechanism provides for the integration of diverse types of multimedia information The flexibility of the aggregation hierarchy is also essential for the storage and manipulation of the vector graphics shapes shown under the Drawing node in Figure 2 Complex drawings may be represented by hierarchies of considerable depth where subtrees of the hierarchies may be shared among drawings [PHIG84] The relational model of data does not support this type of structure directly The application must take responsibility for tracing the hierarchy This situation has been improved by enhancements to the relational model which allow definition of complex objects [KIM85a, LORI83]

Ordering is also necessary at some levels of the aggregation hierarchy For example, the user may wish to express that a Memo will always contain exactly one Memo-Header, one Body, and one Memo-Trailer and they must always exist in that order Another example is the Memo instance in Figure 2 where the particular ordering of the nodes under the Body node is an integral part of the representation of the Memo

2 In addition to its place in the aggregation hierarchy, each node may also be a part of a *generalization hierarchy* [SMIT77] A generalization hierarchy is used to define a node N as a subtype of some other node M such that N can inherit properties from M For example, the Memo-Header node may be a *specialization* of a Header, where Headers always have a Font Size = 14 Then, the Font Size = 14 need not exist as an unshaded oval connected to the Memo-Header node in Figure 2 The value would be inherited from the Header class of objects and used to over-ride the Font Size = 12 in the unshaded oval of the Memo node The notion of inheritance will become increasingly important in the compound document management environment as documents take on more properties and behavior

3 Note that in addition to providing structural and domain constraints, the unshaded squares representing part of the schema in Figure 2 also include data which should be included in every Memo, such as the word "From " There is no easy way to express this in the schema for the relational model The application program must take responsibility for maintaining the data in the unshaded squares as data in separate relations and for presenting it as part of every Memo

4 The properties associated with a node may be represented as either data (oval) or as procedural operations (rectangle) For example, instead of the Font Size = 14 oval, a procedure rectangle may be connected to the Memo-Header which calculates the optimum Font Size based on the resolution of the terminal being used and the other objects presently being displayed If the user wishes to enter a Font Size value for a Memo-Header on a specific Memo, a procedure may constrain the selections available to the user Both of these examples can also be viewed as constraints which are associated with the Memo-Header Another example is the Rectangle labelled SEND in Figure 2 Access of this rectangle will cause the Memo to be sent to the persons listed in the shaded squares below the COPYEE node The procedure may constrain the mode of delivery of the Memo to be decided on the basis of the addresses of the recipients, the size of the document (how large are the images?), the contents of the document (is audio included?), etc

5 It is difficult to define a schema for the information within the Body of the Memo in Figure 2 For example, the Body can consist of any number of Paragraphs, Drawings, or Images in any order Further, there may be constraints which are not easily expressed in a declarative manner For example, the ordering of the Paragraphs, Drawings, and Images in the document may be dependent on the user's point of view or on the level of understanding which the user has of the information being presented The system will use feedback from the user to determine constraints on the order of presentation The best way to represent these constraints is through rules associated with the Body These rules can be represented as procedural data as described in the previous paragraph Rules of this complexity cannot be expressed in traditional database systems

6 There is very little a priori knowledge of the structure of relationships (represented by dotted lines in Figure 2) among nodes In general, any node can have a relationship with any other node regardless of type and position in the aggregation hierarchy The database system must support this type of flexibility There is also little a priori knowledge of the manner or order in which these relationships will be used for accessing information in the database The relationships may actually be an integral part of the user's view of the document and they may be used extensively for navigation This same behavior has also been observed in the mechanical CAD/CAM environment where there are many relationships such as "connected-to" and "on-top-of" which are not part of the normal aggregation hierarchy [CAMM84]

7 There is a need to dynamically modify the schema for Memo This same requirement has been observed in engineering databases [KIM85a, REHF84] For example, a new version of the Memo schema may be created which represents Database Department Memos These are similar to Memos except that they contain a different version of the Memo-Logo and another node following the Subject node, called Priority Indicator The Priority Indicator may contain a number indicating the importance of the document Traditional concepts in schema definition would require entering a new static schema for Database Department Memo, with no simple way of indicating that a Database Department Memo is really just another kind of Memo This semantic information would have to be maintained by the application

8 In addition to the logical structure presented in Figure 2, there is also information concerning the physical presentation of the document This has traditionally been viewed as the static mapping of the intrinsic data such as text or graphic drawings onto some physical display device (paper or display terminal) This mapping can be viewed

as a separate aggregation hierarchy of physical pages and rectangles for presentation, which ultimately has intrinsic data at the leaf nodes

This mapping aggregation hierarchy can become much more complex when sound is added, allowing simultaneous presentation of both sound and images Further complexity is added when the user is allowed to interactively modify aspects of the physical presentation, such as the physical location, orientation, or color of objects on a screen The database system must manage this interactive modification of physical presentation information

9 The creation and control of versions of documents is also an important part of this application The decisions controlling the naming of versions, the selection of default versions, and the notification of documents which reference modified versions will vary with the environment The database system should provide support for version creation, control, and change notification [KATZ84, KIM85a]

10 Concurrent access to the same data by multiple users must be controlled For example, we may specify that changes made to a document will not be visible to other documents, until the changes are committed If changes are being made to the Body of the Memo in Figure 2 and if another Memo is being read which references the Body, the viewer of the second Memo will see the most recent released version rather than the Body being updated This is a straightforward method of concurrency control which suffices for many document management applications

11 The unformatted intrinsic data represented by the Squares may consist of a large number of bytes A bit-mapped still image may require as much as 4,000,000 bytes and one minute of digitized speech may require up to 480,000 bytes of storage The movement of these large amounts of data in the system must be minimized and extra buffering of data must also be minimized Special functions are necessary in the database system to support such requests as "Move backwards 15 seconds" in a voice message and "Replay" the message

12 A document may contain text or complex graphics drawings which are generated based on the value of data items in a database For example, a document may contain a bar chart which describes the average salaries of employees in each department This document can be viewed as either a snapshot of the underlying data at some point in time or as a window into the underlying data The selected view will determine whether the bar chart is updated when the salary of an employee changes This is the snapshot and view mechanism found in conventional database systems

13 The ability to share data among multiple documents is critical to this application There are two primary motivations for this sharing, reduction in use of secondary storage and the ability to reference a specific version of another document Reductions in the use of secondary storage come from sharing the data represented by the squares in Figure 2 For example, if the image in the document in Figure 2 is to be included in another document, it is only necessary to reference the Image node The image need not be copied An example of referencing a specific version of a document is shown in the Reference Relationship from the first Group node under the Drawing node in Figure 2 This Reference is to a portion of another document and is assumed to point to the most recent version of that document

14 Associative access to stored documents must be provided by the database system This access may be based on document structure (all documents with images), document type (all memos), or document contents (all documents containing the word *database*)

15 As is the case with any type of application, database recovery [GRAY81] is an obvious requirement This includes both secondary-storage recovery and transaction recovery For recovery from secondary storage (e g , disk-head crash), the database needs to be periodically dumped to tapes Transaction recovery is important in maintaining mutual consistency of data across multiple documents that a user updates within a single transaction In case of system crashes, all changes to all documents touched within a transaction must be committed or aborted together As in CAD environments [BANC85a], a transaction in a multimedia-application environment will be a long-duration transaction, consisting of a set of conventional short-duration transactions, and involve more than one cooperating users

While the requirements listed above are based on the example of a multimedia document system, they are also found in the much broader area of design applications Research has revealed similar types of requirements in the CAD environment [HASK82, KIM85a, REHF84, CAMM84] Applications in a CAD environment involve an attempt to iteratively capture a concept and translate it into engineering artifacts The data model for these applications must be flexible enough to model both the concepts and the engineering artifacts and to document the iterative design steps

The database approach described in this document will integrate many of the features of the systems described above into a single database system This system will meet all of the functional requirements presented above The object-oriented approach of the system will provide the flexibility of information representation and manipulation necessary for satisfying requirements 1-8 Requirements 12-15 will be met by adapting traditional database technology to the object-oriented approach Requirements 9-11 will be met by adding new functions to the database system which are valuable regardless of whether or not the system is object-oriented

4 0 Multimedia Application Model

The application will be described in terms of diagrams which are similar to semantic nets [NILS80] and entity-relationship diagrams [CHEN76] The diagram is a directed acyclic graph containing nodes and directed arcs The types of nodes and the legal directed arcs from each type of node are shown in Figure 3 There are six types of nodes represented in Figure 3 token objects representing classes, token objects representing instances, relationship objects, attributes, method objects, and intrinsic data objects Every node (with the exception of Attributes) is labelled with a unique numeric identifier This identifier

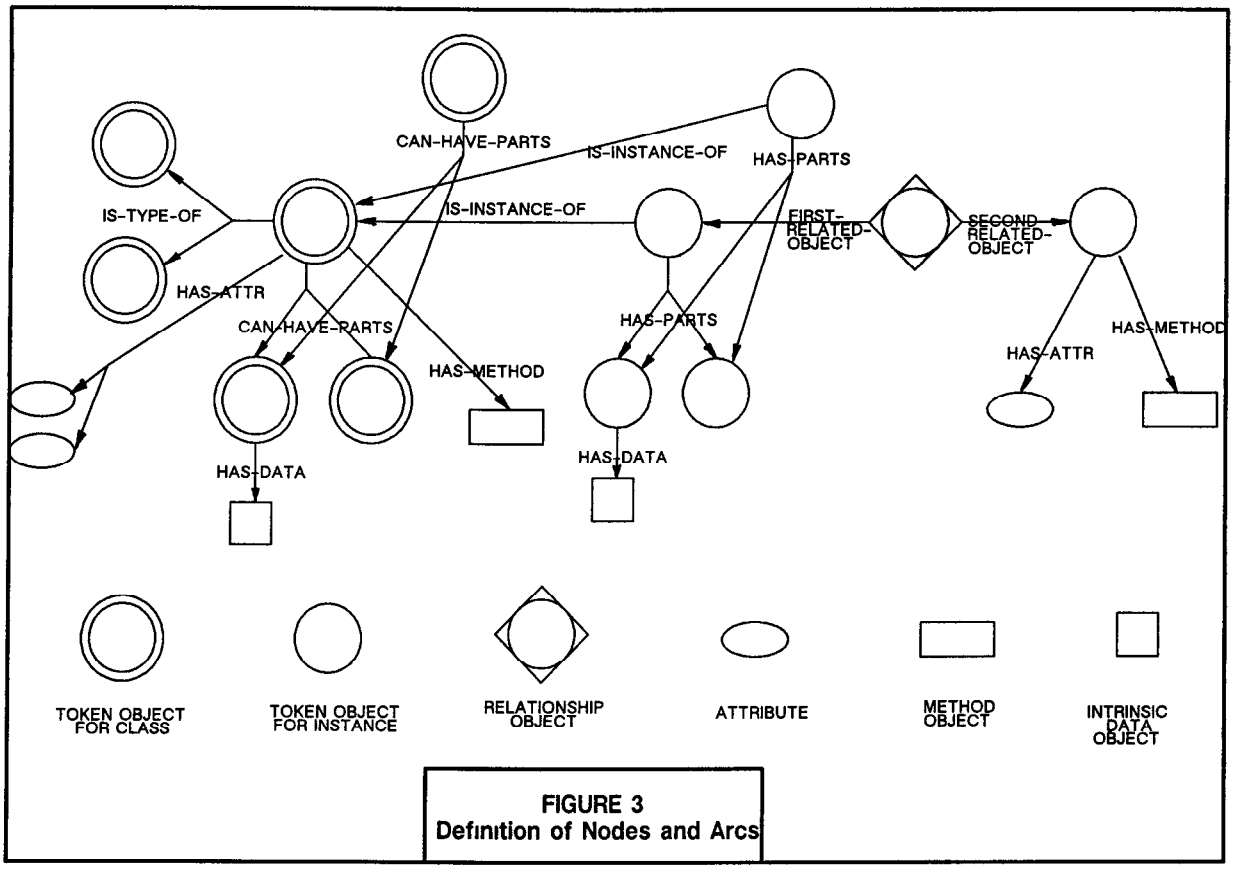

FIGURE 3
Definition of Nodes and Arcs

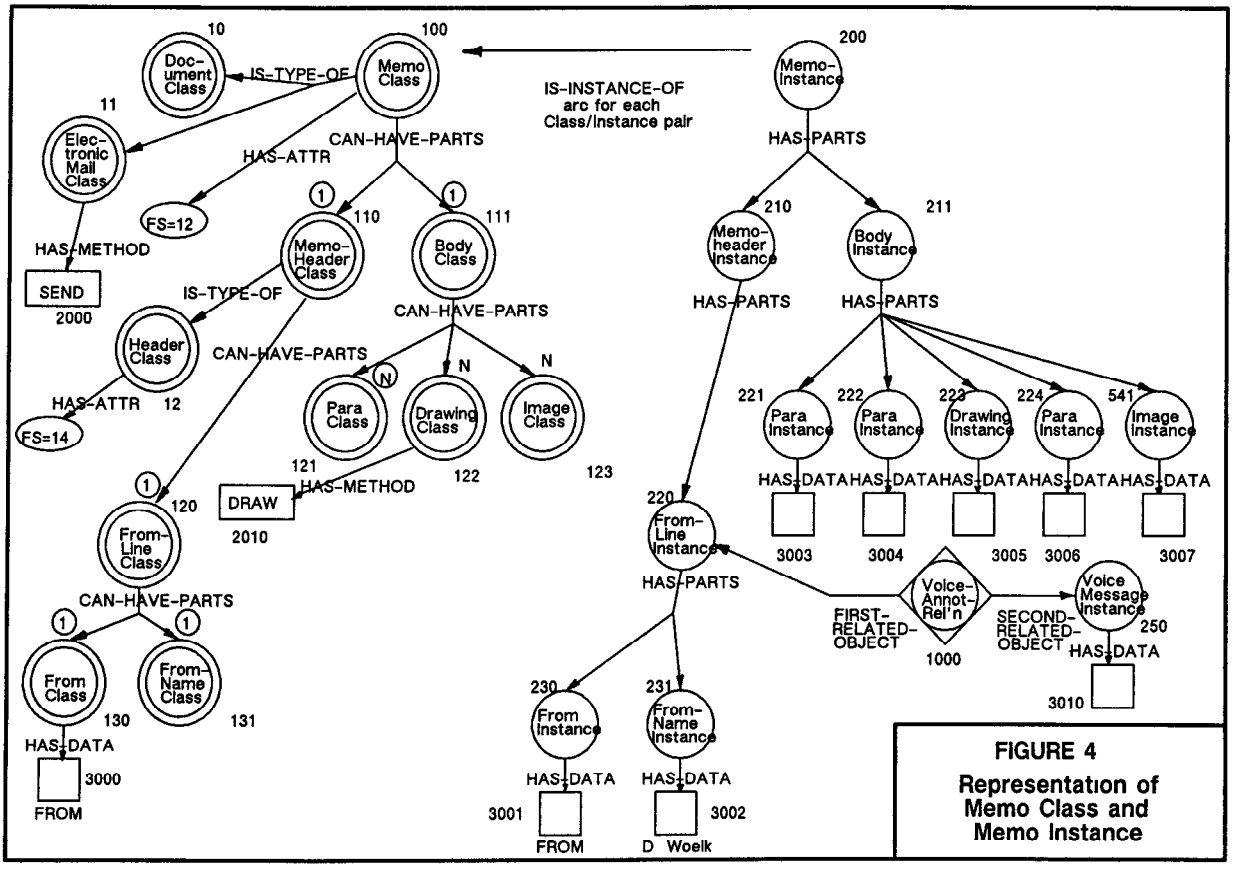

FIGURE 4
Representation of Memo Class and Memo Instance

provides a unique system wide identification for every object in the system The identifier is important for the logical data structures described in Section 5

4 1 Token Objects for Classes and Instances

Multimedia applications require that the database store and manipulate a diverse group of complex types of data We have developed the concept of a *token object* to provide a single mechanism for representing diverse types of data and the relationships among these diverse types of data The key to accomplishing this goal is the flexibility in the expression of *abstraction* Smalltalk objects and LISP objects provide abstraction in the form of *generalization* and *instantiation* A token object provides generalization and instantiation, but adds to these the power of abstraction through *aggregation* A token object "stands-for" all of the token objects and intrinsic data beneath it in the aggregation hierarchy This is similar to the approach taken in the Mosaic design support system [ATWO85]

A token object also differs from a Smalltalk object in that each token object can be stored independently and can be included in many aggregation hierarchies, thus creating an aggregation lattice Methods are treated as separate objects which can be linked to multiple token objects, rather than belonging to classes The following describes the legal directed arcs which are associated with a token object as shown in Figure 3

Instantiation
Token objects are used to implement the concept of instantiation Each instance is represented by a token object which has an **IS-INSTANCE-OF** directed arc to an object class represented by another token object For example, the **IS-INSTANCE-OF** directed arc between the Memo instance node and the Memo class node in Figure 4 represents instantiation The instance will inherit properties from the class unless over-ridden by the instance

Generalization
Token objects are also used to implement generalization An object class, represented by a token object, can have an **IS-TYPE-OF** directed arc to another token object representing another object class In Figure 4, there is an **IS-TYPE-OF** directed arc between Memo-Header class and Header class This arc indicates that Memo-Header is a sub-class of Header It will inherit properties of Header, such as the Font Size = 14, unless over-ridden in the Memo-Header properties An object may be a sub-class of more than one class Therefore, the **IS-TYPE-OF** arc may refer to a set of objects This creates a *generalization lattice* rather than a *generalization hierarchy* This set is ordered so as to specify precedence if there is a conflict in inheritance of properties

Aggregation
The implementation of aggregation is one of the most important uses of the token object Aggregation can be expressed for both object classes and object instances In Figure 4, there is a **HAS-PARTS** directed arc between the Body instance and the three Paragraph instances, the Drawing instance, and the Image instance This arc indicates that the Body instance contains the other instances, creating an *aggregation hierarchy* The ordering of the instances determines the order of presentation

A token object can be viewed as either a *simple token object* or an *aggregate token object* An aggregate token object consists of a simple token object and all of its descendants in the aggregation hierarchy The aggregate token object is important in modelling the multimedia document application and other engineering design applications which are structured hierarchically An aggregate token object can share a subtree of its aggregation hierarchy with another aggregate token object, thus creating an *aggregation lattice* The aggregation lattice is the basis for data sharing, version control, and some database access techniques

Properties can be propagated down the aggregation lattice [BOBR83] For example, if two documents share a common paragraph, the Font Size value for the paragraph will be propagated from the document which is presently being viewed Note that the value of an attribute is propagated from the containing instance to the parts which it contains in the aggregation lattice This differs from inheritance where a class inherits a property from a superclass in the generalization lattice

When aggregation is used for classes of objects, the meaning is slightly different In Figure 4, there is a **CAN-HAVE-PARTS** directed arc between the Body class and the three classes, Para, Drawing, and Image This indicates that a Body can contain paragraphs, drawings, and images The notation "N" next to the Para, Drawing, and Image classes indicates that any number of paragraphs, drawings, and images can occur The N is circled for the Para class to specify that a Body *must* include at least one paragraph The circled notation "1" next to the Body class specifies that a Memo must contain a Body and can contain *only one* Body

Attributes
Each token object has a **HAS-ATTRIBUTES** arc which points to multiple *attributes* as shown in Figure 3 Both object classes and object instances may have these arcs An example of an attribute is the Font Size = 14 in Figure 4 The Font-Size attribute can take its value from a restricted set of values specified by the user Other examples of attribute data are physical location of a graphics object in some coordinate system, color of a graphic object, etc An attribute is not considered to be an object It has no unique identifier and can not be shared by more than one token object

Methods
The operations on a token object are referred to as *methods* as in Smalltalk [GOLD81] Each token object has a **HAS-METHODS** arc which points to one or more programs which are stored independently from the token object Both object classes and object instances may have these arcs The program will contain procedural statements in a selected programming language (such as Lisp) An example is the SEND method in Figure 4 which sends the Memo instance via electronic mail Another example is the DRAW method which handles the physical presentation of instances of the Drawing class

4 2 Intrinsic Data Objects

Intrinsic data is represented by squares at the leaf nodes of the aggregation hierarchy in the diagram in Figure 3 Above the level of intrinsic data, the system can manipulate token objects, which stand for the intrinsic data, without concern for the internal format of the intrinsic data This provides great power and flexibility for manipulating different types of multimedia data A special-purpose *media manager* for each type of media will manipulate the intrinsic data Portions of this media manager will reside in the database system For example, if a request is received to play 5 minutes of sound, the database system will translate this request and return the correct number of bytes of digitized sound to the speech output device

4 3 Relationship Objects

While the token object formalizes the generalization, instantiation, and aggregation relationships which an object may have, there are many other relationships among objects which are necessary to model a multimedia application These are modelled through the *relationship object* nodes in the diagram in Figure 3 These relationships can be named by the user and provide for functions such as annotation of text by sound and references to other documents Figure 4 shows a From-Line instance related to a Voice-Message instance via a relationship object which represents a Voice-Annotation relationship Since a relationship object is an object with a unique identity, a relationship object may also have relationships with other objects For example, the Voice-Annotation relationship object described above could itself be annotated by another Voice-Message through a separate relationship object representing a separate Voice-Annotation relationship

5 0 Logical Implementation

This section will describe the logical implementation of the database functions necessary to support the multimedia document application These functions will first be described in terms of the diagrams which were introduced in the previous section Then, in Section 5 5, the logical data structures for implementing the functions will be described

5 1 Operations on Classes and Instances

5 1 1 Operations on Classes

The creation of either an object class or an object instance requires the creation of a token object The left side of Figure 4 is a partial representation of the token objects which describe the Memo class Classes can be created, modified and deleted dynamically at any time by the user When a class is deleted, all of its instances are deleted as well However, sub-classes of the class and their instances are not necessarily deleted This is a research issue which is still under investigation

Classes can be created as specializations of one or more other classes through **IS-TYPE-OF** links to other token objects Classes can also be explicitly linked to other classes to form an aggregation lattice using the **CAN-HAVE-PARTS** link Attributes for a class can be defined by specifying the attribute name, legal values which it may have, and an initial value The generalization lattice will be used extensively for inheritance of attributes, thus reducing work required to create new classes Intrinsic data, such as the "FROM " square in Figure 4, can be included with the class information to provide intrinsic data which is common to all instances of the class

The class information may be modified The user can create a new version of the class, which will be used for the creation of new instances Versions will be discussed in more detail in Section 5 2

5 1 2 Operations on Instances

The right side of Figure 4 depicts an instance of the Memo class The Memo instance will inherit all of the attributes from the Memo class as well as the attributes from the Document class and the Electronic-Mail class When the user requests the creation of a new Memo instance, token objects are created for the instances at all levels of the aggregation hierarchy The class information on the left side of Figure 4 is used to constrain the selection of classes whose instances can be linked using a **HAS-PARTS** arc on the right side of Figure 4 For example, the Memo instance can only contain a Memo-Header instance and a Body instance

A new version of a class can also be created during the generation of a document instance For example, suppose the user wishes to add a Priority-Indicator to the Memo-Header The Priority-Indicator will contain a number indicating the priority of the Memo The user can create a new class called Priority-Indicator which can be linked to the Memo-Header class with a **CAN-HAVE-PARTS** arc An instance of this class can be created and linked to the Memo-Header instance with a **HAS-PARTS** arc The Memo-Header class has now been modified, thus creating a new version of Memo-Header class as described in Section 5 2

5 2 Version Creation and Control

The control of the evolution of documents is an integral part of the conceptual view of this application environment Version control in the document creation environment is similar to that in any design environment such as VLSI chip design or software system design Version creation and control is implemented using versions of token objects Since token objects are used to implement both classes and instances, version control applies to both classes and instances Two types of versions will be discussed here Historical versions represent a history of a document as it has evolved over a period of time Alternative versions are different implementations or representations of the same abstract object [KATZ84]

5 2 1 Historical Versions

The user must explicitly request the creation of a new historical version The length of time between the designation of historical versions by the user will vary with the specific application The unshaded token objects in Figure 5 represent an instance of a Book as it exists at some point in time Suppose that the Book is being created by multiple authors and that an author wishes to make changes to one of the Chapters The author will retrieve the Chapter from

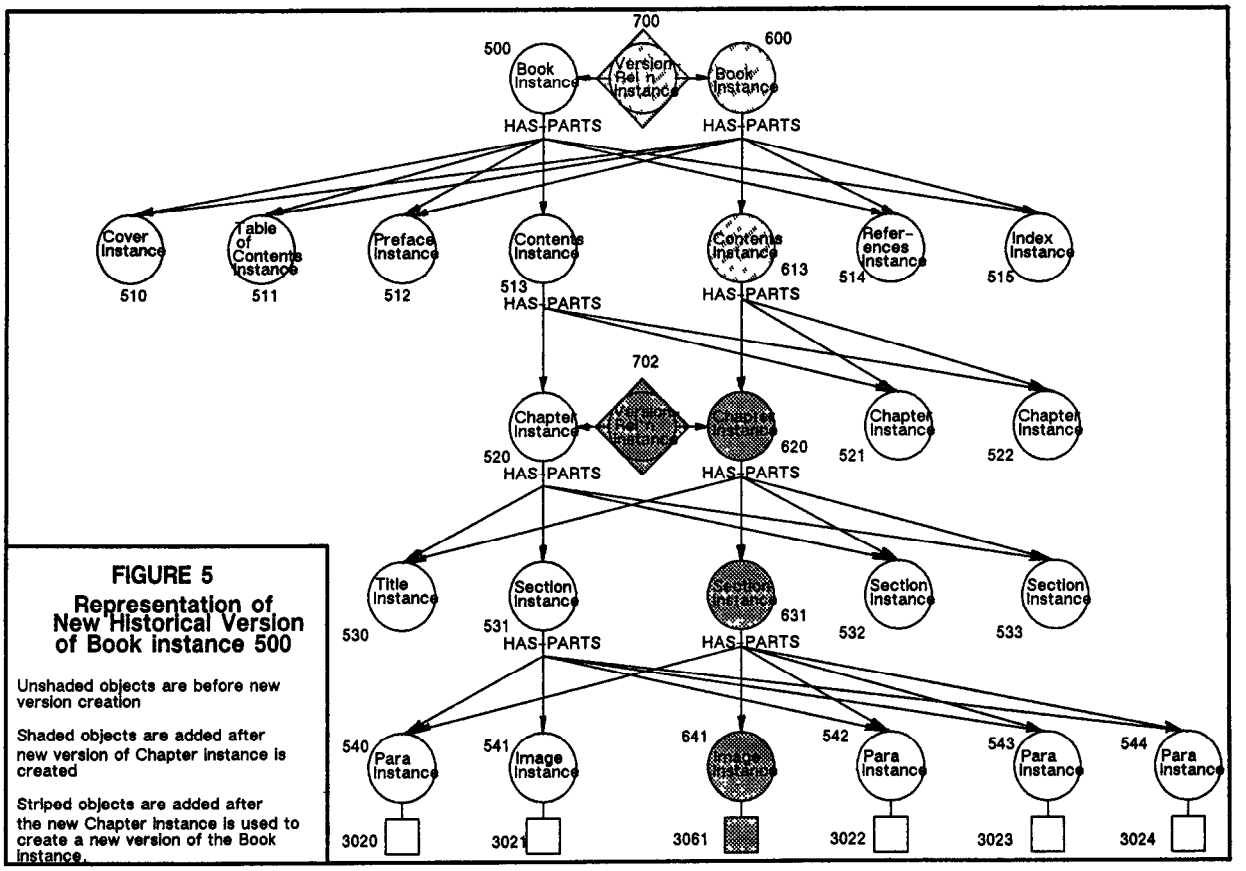

FIGURE 5

Representation of New Historical Version of Book instance 500

Unshaded objects are before new version creation

Shaded objects are added after new version of Chapter instance is created

Striped objects are added after the new Chapter instance is used to create a new version of the Book instance.

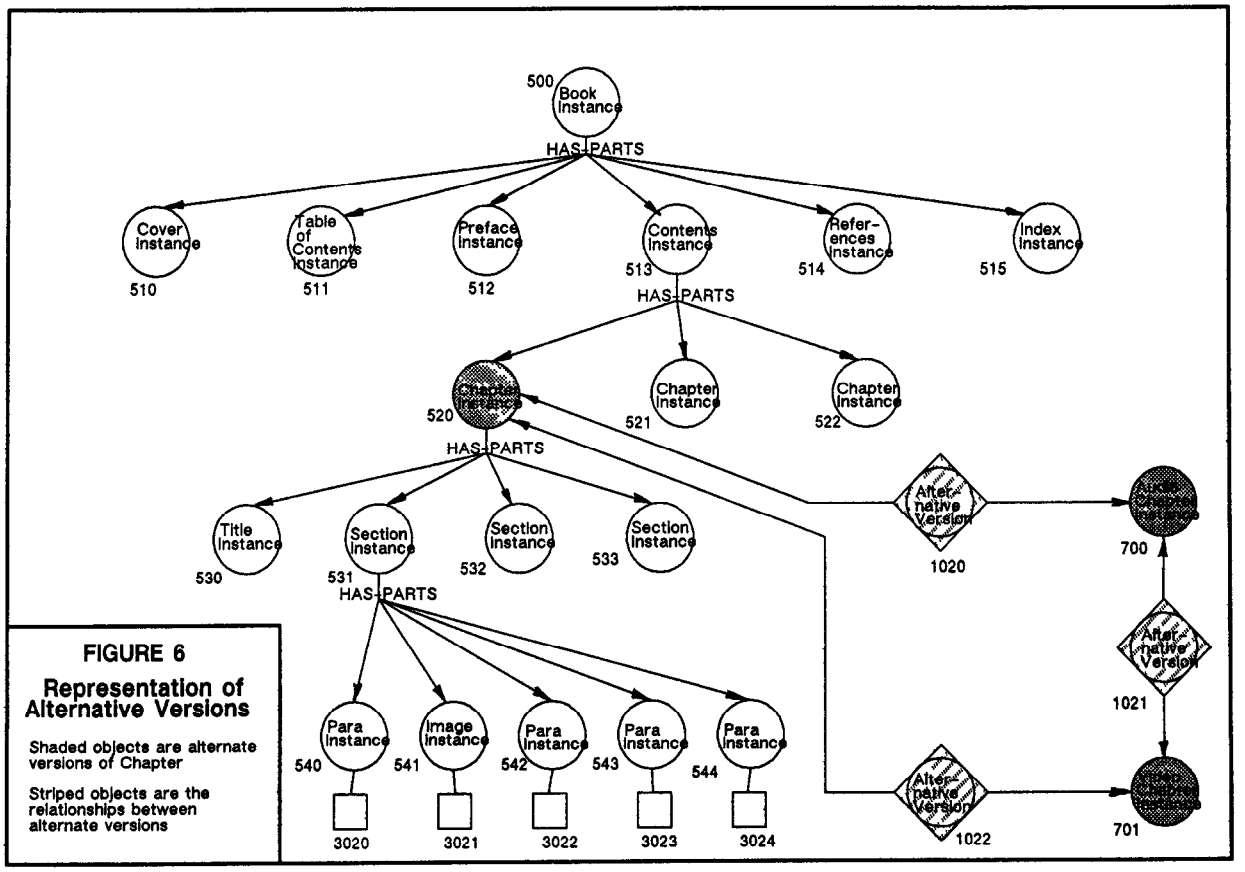

FIGURE 6

Representation of Alternative Versions

Shaded objects are alternate versions of Chapter

Striped objects are the relationships between alternate versions

the database and indicate that a new historical version of the Chapter is going to be created

Suppose that the author wishes to make a modification to the Image instance 541 at the bottom of Figure 5 A copy of the token object representing Image instance 541 will be created and given a unique identifier This new Image instance 641 is shown as shaded in Figure 5 The new copy of the image can now be modified by the author The creation of Image instance 641 will cause the creation of new token objects upwards in the aggregation hierarchy This is termed *percolation* in [ATWO85] The shaded Section instance 631 will be created by this percolation Percolation will stop at the Chapter instance 620 The shaded token objects in Figure 5 now represent the new version of the Chapter

The author may now explicitly request that a new historical version of the Book instance 500 be created as Book instance 600 This will cause the creation of new token objects to percolate up to the Book instance level as shown in the striped token objects in Figure 5

5 2 2 Alternative Versions

Historical versions imply sequential changes to an object Alternative versions represent different implementations or representations of the same abstract object For example, the shaded token objects in Figure 6 represent three completely different representations for a Chapter There is a text, an audio, and a movie representation of the Chapter There may also be multiple historical versions of each of these alternatives The Book instance 500 in Figure 6 is linked through its aggregation hierarchy to one of the alternatives The alternatives are linked to each other through relationship objects representing Alternative-Version-Relationships These are shown as striped token objects in Figure 6 Authors can now create historical versions of the Book which combine different historical and alternative versions of the Chapter At the implementation level, there is not a significant distinction between historical versions and alternative versions

5 3 Sharing of Token Objects

The discussion of versions has shown how a token object or intrinsic data can become a member of more than one aggregation hierarchy implicitly as versions are formed It is also possible to explicitly place a token object or intrinsic data into more than one aggregation hierarchy There are two ways to do this depending on whether or not modifications made later to the object are to be seen

5 3 1 Sharing by Deferred-Copy

A *deferred copy* is a logical copy which is used so that large intrinsic data objects need not be replicated Suppose that a user wishes to copy an image from one document to another document The user does not want to see any future changes in the image, the user only wishes to make a copy of it This type of sharing of an object can be accomplished through the **HAS-PARTS** links In Figure 7, the user issued a request to copy the shaded Image instance 541 from the Book into the Memo Instead of making a physical copy, the Image instance 541 was linked to the Body instance 211 via the **HAS-PARTS** arc As far as the user is concerned, a new copy of the image has been created If, as in Figure 5, Image instance 541 in the Book instance 500 is modified to create a new Image instance 641, the Memo will continue to include the old Image instance 541

5 3 2 Sharing by Reference

Now suppose that the user wishes to include the image in a document, but always wants the document to contain the most recent version of the image This time the user will request to include a *reference* to the image in the document This will cause the creation of a Reference token object which will be used to link the image into the aggregation hierarchy of the Memo Figure 8 shows the result of this reference The shaded Image instance 541 has been linked to the Memo instance 200 via the striped token object representing the Reference instance 900 Now, when the Image instance 541 is modified, Memo instance 800 will contain the new version An attribute of the Reference token object also contains a flag indicating whether the viewer should be notified that the image has been modified since the last time it was viewed [KIM85b]

5 4 Accessing Objects

The object-oriented model is associated with an interactive user-friendly environment The model itself encourages and simplifies the concept of pointing at an object to find out the allowable operations on the object and then selecting a desired operation Object-oriented models do not emphasize other modes of access The approach proposed here will take advantage of the associative access which is inherent in instantiation, aggregation, generalization, and user-defined relationships A number of different modes of access are outlined below

Access to an Object Instance Through its Class
Every object is an instance of a class of objects The token object implements this relationship with an **IS-INSTANCE-OF** directed arc from the instance to the class The inverse of this arc will provide the identity of the instances of each class For example, the Memo class 100 in Figure 4 can identify all of the Memo instances, such as Memo instance 200 The extent to which this access mode is used will have to be considered in designing the physical implementation of the system

Access Through the Aggregation Hierarchy
A token object can be viewed as either a simple token object or an aggregate token object as described in Section 4 1 Therefore, classes and instances can be viewed as either simple or aggregate A request to retrieve the simple Memo instance 200 in Figure 4 will return only one token object Requests for descendants in the aggregation hierarchy, such as Memo-Header instance, can then be found through the **HAS-PARTS** arcs

On the other hand, a request to retrieve the aggregate Memo instance 200 will return the token object for Memo instance 200, plus all of its descendants linked through the **HAS-PARTS** arcs Higher level queries such as "Retrieve MEMO where MEMO FROM-NAME = D Woelk" can also be expressed using the aggregation hierarchy

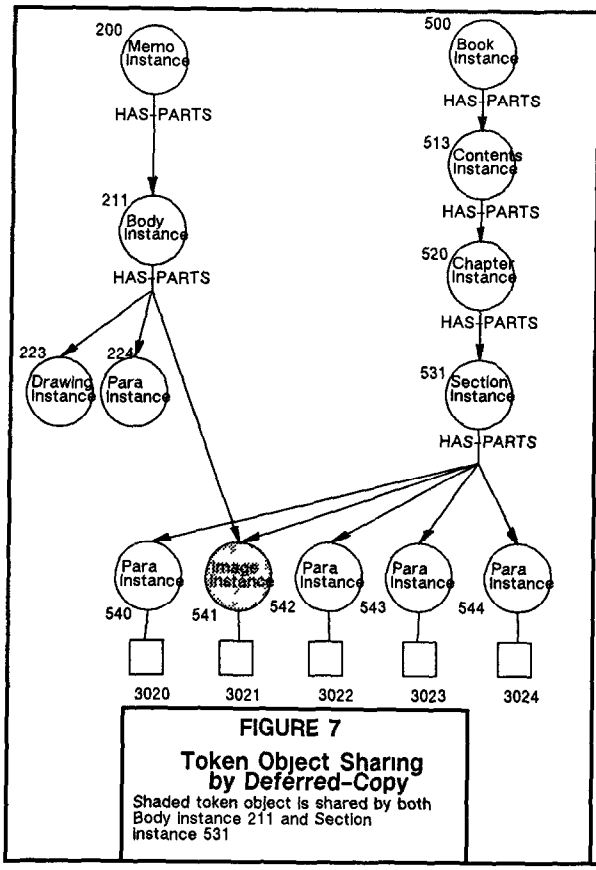

FIGURE 7

Token Object Sharing by Deferred-Copy

Shaded token object is shared by both Body instance 211 and Section instance 531

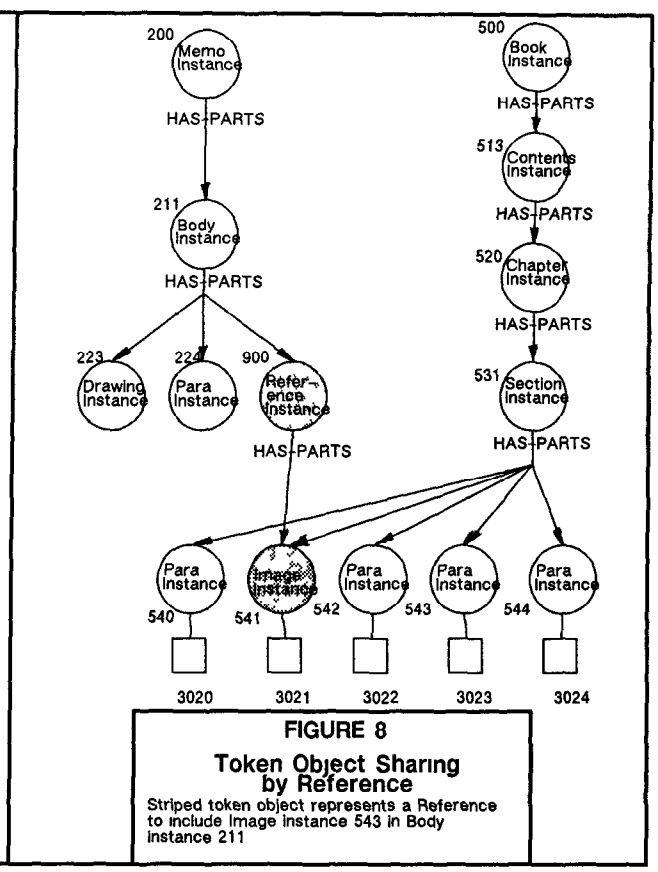

FIGURE 8

Token Object Sharing by Reference

Striped token object represents a Reference to include Image instance 543 in Body instance 211

TOKEN OBJECTS for CLASSES

obj-id	class-name	is-type-of obj-id	can-have-parts			has-data obj-id	has-attributes	has-methods
			min	max	obj-id			
100	Memo	10 (Document) 11 (Elect Mail)	1 1	1 1	110 (Memo-Header) 111 (Body)		Font-Size=12	
10	Document							
11	Electronic-Mail							2000 (Send)
110	Memo-Header	12 (Header)		1	120 (From-Line)			
12	Header						Font-Size=14	
120	From-Line		1 1	1 1	130 (From) 131 (From-Name)			
130	From					3000		
131	From-Name							
111	Body		1 0 0	N N N	121 (Para) 122 (Drawing) 123 (Image)			
121	Para							
122	Drawing							2010 (Draw)
123	Image							

METHOD OBJECTS

obj-id	method
2000	(Send Program)
2010	(Draw Program)

FIGURE 9

Logical Data Structures

Logical data structures which represent the Memo class in Figure 4

Comments enclosed in parentheses

Access Through the Generalization Hierarchy
The generalization hierarchy contains valuable information which can be used for accessing objects A request such as "Retrieve all Documents" can be answered by first finding all of the sub-classes of the Document class In Figure 4, the Document class has only one sub-class which is indicated by the **IS-TYPE-OF** directed arc from the Memo class to the Document class The inverse of this arc will provide a link between the Document class and the Memo sub-class All Memo instances can then be returned using the inverse of the **IS-INSTANCE-OF** arc

Access Through Relationships
Access to objects through instantiation, aggregation, and generalization paths all use information about structural relationships among objects which the user has provided The user may also create other relationships among objects through the creation of relationship objects as described in Section 4 3 These relationships can be used to move from one object to another as in Figure 4, where a Voice-Annotation relationship object is used to link a From-Line instance and a Voice Message instance Relationship objects may also be searched directly to provide associative access for a query such as "Retrieve all Voice-Annotations"

Access Through Feature Extraction from Intrinsic Data
Intrinsic data may exist at the leaf nodes of the aggregation lattice Since the internal format and presentation characteristics of each type of intrinsic data is different, a special-purpose media manager will be needed for manipulating this data For some types of media, it is possible to access a subset of the data based on certain *features* of the data These features are not explicitly described as part of the data and must first be *extracted* from the data

This extraction process may be very slow or even impossible in some cases In speech recognition, for example, the feature extraction process is presently efficient only for limited vocabulary and limited number of independent speakers speaking slowly Image recognition is equally restricted Searching for patterns in text, on the other hand, is a common practice in text editors and information retrieval systems This searching can also be accelerated through the use of specialized hardware [HASK83] It would be possible, therefore, to pass a pattern to the Text media manager and have the media manager search for the pattern in all text passages or in some subset of text passages

Combinations of Access Modes
Any of the access modes described above can be combined to create a query An example is the query "Retrieve Memos which contain pictures of database machines and which reference the 1985 Database Program Plan" This query combines a number of access modes, and as such it consists of a number of sub-queries The performance of the execution of the query may be dependent on the order in which the individual sub-queries are executed One sequence of execution is to find the 1985 Database Program Plan, search for Reference relationship objects linked to the 1985 Database Program Plan, find the Memos which make reference to the 1985 Database Program Plan, search for those Memos which have pictures, and display these pictures for user recognition of a "database machine" Other sequences of sub-query execution are possible and may result in different performance

Browsing as an Access Mode
Browsing is an important mode of accessing documents in a multimedia document management environment Many times the user cannot precisely formulate a query using the access methods listed above which will result in a satisfactory answer There may be too many documents which fit the selection criteria or there may be none at all In either case the query must be reformulated to either expand or narrow the selection criteria If many documents are returned as a result of a query, the user may wish to look at a few before deciding to narrow the selection criteria As described in [CHRI84], the query may then be reformulated with a request that documents already viewed should not be viewed again In addition, the user can organize and classify the documents which satisfy the selection criteria by interactively creating a temporary "document" which contains these documents

5 5 Logical Data Structures

Figures 9 and 10 illustrate the logical data structures which are used to implement the functions described in the previous section The data structures contain data which represent the Memo class and Memo instance that were described in Figure 4 Figure 11 represents the two Book instance versions described in Figure 5 (Text enclosed in parentheses in these structures are comments that we have added to aid the reader in understanding the example)

The token object format used for representing classes is presented in Figure 9, where each row represents a class The token object format for instances is presented in Figure 10 and Figure 11, where each row represents an instance Note that the structures are similar since classes and instances are represented in a similar manner in the diagram of Figure 4 However, there are a few differences A class does not have an **IS-INSTANCE-OF** field, whereas an instance does not have an **IS-TYPE-OF** field Further, the semantics of the **CAN-HAVE-PARTS** field of the class and the **HAS-PARTS** field of the instance are different The **CAN-HAVE-PARTS** field of a class specifies the classes of the objects which an instance of the class *may* contain in its next lower level in an aggregation hierarchy The **MIN** value specifies the minimum number of instances of the class which can be contained The **MAX** value is the maximum number of instances of the class that can be contained The **HAS-PARTS** field of an instance, on the other hand, specifies the *actual* instances which are contained in the next lower level

A couple of other observations can be made concerning the logical data structures of the token objects used to represent classes and instances It is obvious that these tables are not normalized relations, due to the multiple values for some fields For example, the **HAS-PARTS** field of the sixth row of the token object structure in Figure 10 contains an ordered list of five object identifiers The structures might be represented as complex objects as defined in [KIM85a, LORI83] or in the Set and Tuple Data Model of [BANC85b] However, these models cannot represent the inheritance (or propagation) of properties through generalization (or aggregation)

TOKEN OBJECTS for INSTANCES

obj-id	is-instance-of obj-id	has-parts obj-id	has-data obj-id	has-attributes	has-methods
200	100 (Memo)	210 (Memo-Header) 211 (Body)			
210	110 (Memo-Header)	220 (From-Line)			
220	120 (From-Line)	230 (From) 231 (From-Name)			
230	130 (From)		3001		
231	131 (From-Name)		3002		
211	111 (Body)	221 (Para) 222 (Para) 223 (Drawing) 224 (Para) 541 (Image)			
221	121 (Para)		3003		
222	121 (Para)		3004		
223	122 (Drawing)		3005		
224	121 (Para)		3006		
541	123 (Image)		3007		
250	(Voice-Message)		3010		

RELATIONSHIP OBJECTS

obj-id	first-related-obj-id	second-related obj-id	relationship-type
1000	220 (From-Lin	250 (Voice-Mess	Voice-Annotation
700	500 (Book)	600 (Book)	Historical-Version
702	520 (Chapter)	620 (Chapter)	Historical-Version

FIGURE 10

Logical Data Structures (cont'd)

Logical data structures which represent memo instance in Figure 4

Comments enclosed in parentheses

TOKEN OBJECTS for INSTANCES

obj-id	is-instance-of obj-id	has-parts obj-id	has-data obj-id	has-attributes	has-methods
500	(Book)	510 (Cover) 511 (T of C) 512 (Preface) 513 (Contents) 514 (References) 515 (Index)			
513	(Contents)	520 (Chapter) 521 (Chapter) 522 (Chapter)			
520	(Chapter)	530 (Title) 531 (Section) 532 (Section) 533 (Section)			
531	(Section)	540 (Para) 541 (Image) 542 (Para) 543 (Para) 544 (Para)			
541	(Image)		3021		
600	(Book)	510 511 512 613 514 515			
613	(Contents)	620 521 522			
620	(Chapter)	530 531 532 533			
631	(Section)	540 641 542 543 544			
641	(Image)		3061		

FIGURE 11

Logical Data Structures (cont'd)

Logical data structures which represent two versions of Book instance in Figure 5

Comments enclosed in parentheses

The creation of a new class will result in the insertion of a new row in the token object structure in Figure 9 The **IS-TYPE-OF** field will contain a list of the object identifiers for the classes of which this class is a sub-class The creation of an instance of a class will result in the creation of a new row in the token object structure in Figure 10 The **IS-INSTANCE-OF** field will contain the object identifier of the class If a class is deleted, all of the instances of this class must be deleted Note that this will require searching of all the **IS-INSTANCE-OF** fields for the object identifier of the instances of the deleted class, unless an index of inverse pointers to instances is maintained for each class

The creation of a relationship object will result in the creation of a new row in the relationship objects structure in Figure 10 The first row of that structure represents the Voice-Annotation relationship which was shown in Figure 4 The **FIRST-RELATED-OBJECT** and **SECOND-RELATED-OBJECT** fields contain the object identifiers for the token objects which are related, the From-Line instance 220 and the Voice-Message instance 250 An index of inverse pointers from token objects for instances to relationship objects are necessary to find the relationships in which a token object takes part if a full search of the relationship objects structure is to be avoided

The unshaded portion of Figure 11 represents the Book instance 500 from Figure 5 The lightly shaded portion represents the new version of the book which is identified as Book instance 600 These two instances both contain many of the same object identifiers in their respective **HAS-PARTS** fields The relationship object created to represent the version relationship between the two instances is also shown in Figure 10

Figure 7 described the creation of a deferred-copy of Image instance 541 taken from Book instance 500 which was included in Memo instance 200 The entries in the **HAS-PARTS** fields which represent this deferred-copy are shown as darkly shaded in Figure 10 and Figure 11 Notice that in row 5 of Figure 11, the Image instance 541 entry does not include any indication of the identities of the aggregation hierarchies in which it exists Here again an index of inverse pointers is necessary to eliminate the need for searching the **HAS-PARTS** fields of all of the token objects

Inheritance of properties through the generalization lattice can cause extensive pointer chasing For example, to find the Font-Size value for the Memo-Header instance 210 in Figure 10 (Row 2), the system must look at the Memo-Header class 110 in Figure 9 (Row 4) for a Font-Size attribute Since no value exists for Font-size for Memo-Header class 110, the system must look at the Header class 12 (Row 5) where a Font-Size = 14 is found

6 0 Concluding Remarks

In this paper, first we identified two types of requirements which multimedia (compound document management) 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 compound documents and capture the complex relationships among parts of compound documents Another is the requirement for sharing and manipulating (storage, retrieval, and transmission) compound documents containing images and voice as intrinsic data

We then proposed an object-oriented data model which generalizes the notions of instantiation and generalization that form the basis of the current object-oriented systems, and which augments the conventional object-oriented paradigm with the notions of aggregation and relationships We showed that this object-oriented data model is an elegant approach to addressing all data modelling requirements of the multimedia applications Further, we showed, in terms of 'logical' data structures, how operations on the data model (e g , creation and deletion of classes and instances) can be implemented in an object-oriented database system

There are a number of important issues that require further research First, property inheritance and constraints management is more complex in our system than in conventional object-oriented systems since our data model supports the notions of instantiation, generalization, and aggregation We need more work on this Second, we need to extend our 'logical' implementation of the data model to a 'physical' implementation In particular, we need to address such issues as buffering techniques for transmitting the long intrinsic data, a cost model and a query processing strategy for complex associative search of the contents of compound documents, and storage structures for the 'logical' data structures we developed to support the operations on our data model

REFERENCES

[AHLS84] Ahlsen M , A Bjornerstedt, S Britts, C Hulten, and L Soderlund "An Architecture for Object Management in OIS," *ACM Trans on Office Information Systems*, vol 2, no 3, July 1984, pp 173-196

[APPL83] *MACINTOSH*, by Apple Computer, Inc Cupertino, Ca, 1983

[ATWO85] Atwood, T M "An Object-Oriented DBMS for Design Support Applications," *Proc IEEE COMPINT 85*, Montreal, Canada,pp 299-307

[BANC85a] Bancilhon, F , W Kim, and H F Korth "A Model of CAD Transactions," *Proc VLDB*, 1985

[BANC85b] Bancilhon, F "A Set and Tuple Data Model," *MCC Technical Report DB-020-85*, June 1985

[BEEC83] Beech, D and S Feldman "The Integrated Data Model A Database Perspective," *Proc VLDB*, 1983, pp 302-304

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

[BIRT73] Birtwistle, G , O J Dahl, B Myhrhaug, and K Nygaard *Simula Begin*, Auerbach Publishers, Philadelphia, Pa , 1973

[CAMM84] Cammarata, S and M Melkanoff "An Interactive Data Dictionary Facility for CAD/CAM Data Bases," *Proc First Int'nl Workshop on Expert Database Systems*, Oct 1984, pp 360-377

[CHEN76] Chen, P P S "The Entity-Relationship Model - Towards a Unified View of Data," *ACM Trans on Database Systems*, vol 1, no 1, 1976, pp 9-36

[CHRI84] Christodoulakis, S , J Vanderbroek, J Li, S Wan, Y Wang, M Papa, and E Bertino "Development of a Multimedia Information System for an Office Environment," *Proc VLDB*, 1984, pp 261-271

[COPE84] Copeland, G and D Maier "Making Smalltalk a Database System," *ACM SIGMOD*, June 1984, pp 316-325

[FOGG84] Fogg D "Lessons from a 'Living in a Database' Graphical Query Interface," *ACM SIGMOD*, 1984, pp 100-106

[FORS84] Forsdick, H C , R H Thomas, G G Robertson, and V H Travers "Initial Experience with Multimedia Documents in Diamond," *IEEE Database Engineering Quarterly Bulletin*, vol 7 no 3, Sept 1984

[FOX85] Fox, M , J Wright, and D Adam "Experiences with SRL An Analysis of a Frame-based Knowledge Representation," *Proc First Int'nl Workshop on Expert Database Systems*, Oct 1984, pp 224-237

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

[GRAY81] Gray, J N , P McJones, M Blasgen, B Lindsay, R Lorie, T Price, F Putzola and I Traiger "Recovery Manager of a Data Management System," ACM Computing Surveys, vol 13, no 2, June 1981, pp 223-242

[HASK82] Haskin, R and R Lorie "On Extending the Functions of a Relational Database System," in *ACM/SIGMOD*, June 1982, pp 207-212

[HASK83] Haskin, R and L A Hollaar "Operational Characteristics of a Hardware-Based Pattern Matcher," *ACM Trans on Database Systems*, vol 8, no 1, March 1983, pp 15-40

[KATZ84] Katz, R and T Lehman "Database Support for Versions and Alternatives of Large Design Files," *IEEE Trans on Software Engineering*, vol SE-10, no 3, March 1984, pp 191-200

[KIM85a] Kim, W "CAD Database Requirements - Rev 1," *MCC Technical Report DB-058-85*, July 1985

[KIM85b] Kim, W , H Chou, and D Woelk "On Versions in a Distributed CAD System," *MCC Technical Report* DB-60-85

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

[LORI83] Lorie, R and W Plouffe "Complex Objects and Their Use in Design Transactions," *Proc ACM Database Week Engineering Design Applications*, May 1983, pp 115-121

[MINS75] Minsky M "A Framework for Representing Knowledge," In *The Psychology of Computer Vision*, P Winston (Ed), New York McGraw-Hill

[MYLO80] Mylopoulos, J , P Bernstein, and H Wong "A Language Facility for Designing Database-Intensive Applications," *ACM Trans Database Systems*, vol 5, no 2, 1980, pp 185-207

[NILS80] Nilsson, N *Principles of Artificial Intelligence*, Tioga Publishing Co , Palo Alto, Ca , 1980, pp 370-378

[PHIG85] *Programmers Hierarchical Interactive Graphics System (PHIGS)*, document X3H3/85-21, X3 Secretariat, CBEMA, 311 First St NW, Suite 500, Washington, DC 20001, 1985

[POGG85] Poggio, A , J J Garcia Luna Aceves, E J Craighill, D Moran, L Aguilar, D Worthington, and J Hight "CCWS A Computer-Based Multimedia Information System," *IEEE Computer*, vol 18, no 10, Oct 1985, pp 92-103

[REHF84] Rehfuss, S , M Freiling, and J Alexander "Particularity in Engineering," Proc *First Int'nl Workshop on Expert Database Systems*, Oct 1984, pp 677-684

[SAKA85] Sakata, S and T Ueda "A Distributed Interoffice Mail System," *IEEE Computer*, vol 18, no 10, Oct 1985, pp 106-116

[SLOA82] Sloan, K R and A Lippman "Data Bases of / about / with Images," *IEEE Conf on Pattern Recognition and Image Processing*, June 1982, pp 441-446

[SMIT77] Smith, J and D C P Smith "Database Abstractions Aggregation and Generalization," *ACM Trans Database Systems*, vol 2, no 2, 1977, pp 105-133

[WEIN81] Weinberg, D and D Moon *Lisp Machine Manual*, Symbolics, Inc , July 1981

[WILS80] Wilson, G and C Herot "Semantics vs Graphics -- To Show or Not To Show," *Proc VLDB*, 1980, pp 183-197

[WOEL85] Woelk, D and W Luther "Multimedia Database Requirements - Rev 0," *MCC Technical Report DB-042-85* , July, 1985

[WONG82] Wong, H and I Kuo "GUIDE Graphical UserInterface for Database Exploration," *Proc VLDB*, 1982, pp 22-32

[YANK85] Yankelovich, N , N Meyrowitz, and A van Dam "Reading and Writing the Electronic Book," *IEEE Computer*, vol 18, no 10, Oct 1985, pp 15-30