NIST Special Publication 500-179

Computer Systems Technology

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology

NIST

Object Database Management Systems: Concepts and Features

Christopher E. Dabrowski
Elizabeth N. Fong
Deyuan Yang

NIST Special Publication 500-179

Object Database Management Systems: Concepts and Features

Christopher E. Dabrowski
Elizabeth N. Fong
Deyuan Yang

National Computer Systems Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

April 1990

Reports on Computer Systems Technology

National Institute of Standards and Technology Special Publication 500-179
Natl. Inst. Stand. Technol. Spec. Publ. 500-179, 63 pages (Apr. 1990)
CODEN: NSPUE2

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1990

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, DC 20402

PREFACE

The National Computer Systems Laboratory (NCSL) (formerly Institute for Computer Sciences and Technology (ICST)) within the National Institute of Standards and Technology (NIST) (formerly National Bureau of Standards (NBS)) has a mission under Public Law 89-306 (Brooks Act) to promote the "economic and efficient purchase, lease, maintenance, operation, and utilization of automatic data processing equipment by federal departments and agencies." When a potentially valuable technique first appears, NCSL may be involved in research and evaluation. Later on, standardization of the results of such research, in cooperation with voluntary industry standards bodies, may best serve federal interests. Finally, NCSL helps federal agencies make practical use of existing standards and technology through consulting services and the development of supporting guidelines and software.

A new information management technology, called Object Database Management Systems (ODBMS) is rapidly emerging. A common definition of ODBMS is needed to avoid confusion among the users, vendors, and standards developers in the database community. The purpose of this report is to describe object concepts and identify a set of features associated with ODBMS.

Certain commercial software products and companies are identified in this report for purposes of specific illustration. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products identified are necessarily the best available for the purpose.

ABSTRACT

The last decade has seen the emergence of object concepts and their infusion into information systems technology. This phenomenon began with the advent of programming languages that included object concepts. More recently, object concepts have been merged with database management system technology, resulting in the production of some object database management systems. As a result, the term object database management system (ODBMS) is now becoming a recognized and important topic in the database community. The purpose of this report is provide managers and software analysts a state-of-the-art review of object concepts and to describe features associated with object database management systems.

Keywords: class; database; database management system; object; object-oriented; object database management system; ODBMS.

ACKNOWLEDGMENTS

The technical work for this report was done through extensive review of published literature and hands-on experimentation with three donated object database management systems. We would like to thank these system vendors for their cooperation and their donation of these systems to the Knowledge-Based Systems Laboratory of NCSL.

We also acknowledge the contributions of David Jefferson, Joseph Collica, Leonard Gallagher, Wayne McCoy, Bruce Rosen, and Thomasin Kirkendall of NIST who read early versions of this paper and provided valuable advice.

TABLE OF CONTENTS

1.0 INTRODUCTION

The purpose of this report is to provide managers and software analysts a state-of-the-art review of object concepts and to describe features associated with object database management systems (ODBMS).

At the present time, there are many areas in computer science in which object concepts are being applied. The scope of this report is limited to the object data model and object database management systems.

1.1 Motivation

In the past 30 years, information systems technology has advanced rapidly and produced remarkable achievements. Databases have evolved from simple file systems to complex and highly interrelated collections of data that now serve large communities of users and support numerous and diverse applications.

The last decade has seen the emergence of object concepts and their infusion into information systems technology. This phenomenon began with the advent of programming languages based on object concepts.[1] Object-oriented programming has since come to be associated with state-of-the-art software development practices. It has also been connected to new disciplines such as artificial intelligence and interactive computer graphics.

More recently, object concepts have been merged with database management system technology, resulting in the production of commercial object database management systems [ZDON90]. As a result, the terms object database and object database management system have now become commonly used buzzwords in the database community.

However, among many potential database application developers and users, little is actually known about this new technology. Part of the reason is a general lack of knowledge about what the term "object" means, what object-oriented programming is, or what an object data model might be. Similarly, there is a lack of understanding about what an

[1]SMALLTALK [GOLD83] was the first serious effort in developing an object-oriented programming language for industrial use.

object database management system is and how it differs from traditional database management systems.

1.2 New Requirements for Database Management Systems

Database management systems (DBMS) have proven to be cost-effective tools for organizing and maintaining large volumes of data. In government and business enterprises, databases are increasingly becoming the focus of many new and varied applications. Many of these applications have data processing requirements which exceed the capabilities provided by database management systems oriented toward business data processing.

Applications associated with such emerging technological disciplines as computer aided design (CAD), geographic information systems (GIS), and knowledge-based systems (KBS) now require databases which can store large quantities of information having complex structures. Further, these applications require support for data types not supported by conventional systems. For example, a database supporting a large engineering design system may have to store tremendous amounts of data in the form of technical design diagrams and descriptions. The database schema must be able to express the complexity of the overall design as well as the relationships which exist between individual design components. Additionally, the database must also support the operations which must be performed during the course of design activity.

Often, conventional database management systems based on the relational, network, or hierarchic data models cannot effectively meet requirements for representing and manipulating complex information. Moreover, engineering or CAD applications may require display of designs consisting of groups of complex components which must be combined, separated, overlaid, and otherwise modified to perform various design tasks. A new concept is needed to provide a basis for the development of a database management system that can effectively represent, retrieve, and update such information, and also perform other data management functions.

1.3 New Possibilities in Database Management Systems

Object database management systems combine the novel concepts associated with object-oriented programming languages with the capabilities of database management systems. Object database management systems are now at the early stages of commercialization.

ODBMS have some features not traditionally present in conventional database management systems. The object data model permits representation of complex data structures and type hierarchies and provides the ability to define data types that combine both data structure and procedure definition. This results in much greater flexibility in the representation of structure and behavior in real-world systems.

ODBMS combine enhanced data typing capability, the ability to define and manipulate complex data structures, the functionality of a programming language, and database management technology. By combining these diverse capabilities, it creates new application development environments. These environments can then be used to create large, sophisticated software systems. The data management requirements of many of these applications also differ significantly from those of traditional business systems in such areas as transaction management. ODBMS are evolving to meet these new requirements.

1.4 The Outline of this Report

This report will first provide a description of object concepts in section 2. Section 3 contains a discussion of object-oriented programming languages. This will provide the reader with the necessary understanding of the object paradigm and how object-oriented programming languages developed.

Section 4 discusses databases, data models and database management systems. Section 5 presents an informal definition of the object data model. Section 6 presents a general description of the architecture of ODBMS. This architecture will provide a basis for the identification of the key features of ODBMS. Section 7 discusses recent developments in ODBMS. Section 8 contains a discussion of ODBMS areas which are still in the research stage, and then provides a short conclusion section.

A glossary of terms, as they appear underlined in this report, are presented in the Appendix.

2.0 OVERVIEW OF OBJECT CONCEPTS

There is no single agreed upon definition of what the terms "object" and "object-oriented" mean. The term "object-oriented" was originally associated with developments in programming languages in the 1970s. More recently, new approaches to data modeling and database managements have begun to incorporate what are now called simply "object" concepts.[2]

The foundation for object concepts is derived from advanced developments in programming languages. Accordingly, for computer professionals with backgrounds in database management systems, object concepts are often new and confusing.

The purpose of this section is to introduce some of the general concepts and terms associated with object techniques and to compare and contrast these concepts, where possible, to those found in database management systems.

2.1 What is an Object?

Conceptually, an <u>object</u> is something which is perceived as an identifiable, self-contained unit, which is distinguishable from its surroundings. An object may be described by a set of attributes that constitute an internal <u>state</u>, and a set of operations which defines its behavior.

In the real world, we can easily identify many objects. An example might be an airplane. An airplane is readily identifiable and is distinguishable from its surroundings. An object has a set of attributes that constitutes an internal state which can change over time. The attributes for an airplane might be direction, altitude, groundspeed (if it is in motion), weight, and color. An object also has a behavior consisting of a set of operations. The operations of an airplane might be the ability to move, change velocity, and respond to stimuli such as wind or external temperature.

In a computer program, the concept of an object is implemented as a software component. This component consists of data structures and procedures. The object's data structures represent its attributes while the object's procedures can be invoked to perform the operations which define the behavior of a real world entity it represents. For example, the object AIRCRAFT can have variables for AIRCRAFT-ALTITUDE, AIRCRAFT-

[2]The word "oriented" is largely a noise word. In this report, "oriented" will be omitted unless it is used to refer to "Object-Oriented Programming Language."

DIRECTION, AIRCRAFT-GROUNDSPEED, AIRCRAFT-POSITION, and AIRCRAFT-FUEL-CONSUMPTION. AIRCRAFT can also have defined procedures to carry out its operations: SET-ALTITUDE, SHOW-DIRECTION, COMPUTE-GROUNDSPEED, SHOW-GROUNDSPEED, COMPUTE-FUEL-CONSUMPTION, SHOW-FUEL-CONSUMPTION, etc.

Objects are unique. Each object has an internal identifier assigned to it known as an <u>object ID</u>, or <u>handle</u>. Hence, there may be many AIRCRAFT objects, each with a unique object ID.

It is possible to think of the data structures associated with objects in terms of the definition of an individual database schema. In the AIRCRAFT example, altitude, direction, position, groundspeed, and fuel consumption might be attributes of the schema. In a conventional database, the attributes of AIRCRAFT might be stored as a record in the database. However, conventional database management systems generally do not support the specific association of procedures with data.

2.2 Communicating With Objects: Messages and Methods

Communication with an object is accomplished by sending a <u>message</u> to it. An object has a set of messages it responds to which constitutes its public interface, or <u>protocol</u>. Messages consist of the name of the message, the name of the target object to which it is being sent, and the necessary arguments, if any. When an object receives a message, one of its procedures is invoked. The procedure then performs an operation which may return a result. In most object systems, procedures are referred to as <u>methods</u>, a term that will be used throughout this report. A method may itself send messages to the object or to other objects.

To see how this works in the AIRCRAFT example introduced above, a computer program simulating the flight of airplanes may model the effect of winds on an aircraft by sending a COMPUTE-GROUNDSPEED message to an AIRCRAFT object. This message would include arguments stating the direction and speed of the wind. The COMPUTE-GROUNDSPEED method for the AIRCRAFT object is then invoked and a new groundspeed and direction are calculated. The corresponding variables are updated, thus changing the object's internal state.

In an object system, message passing between objects is the predominant mode of computation. The conventional top-down program organization characterized by program control through subroutine invocation is replaced by a collection of autonomous objects, which send and receive messages.

2.3 Encapsulation of Objects

The details of the operation performed are not specified in the message and are not visible to the message sender. An object's internal state is accessible only by the object itself. An object's data structures may not be accessed and updated directly by an external agent, but can only be effected indirectly through message passing and method invocation. This characteristic hiding of the object's internal state is known as object encapsulation. Intuitively, encapsulation of an object makes it into a "black box".

Objects can also be thought of in terms of data abstraction. Data abstraction is a technique used in many programming languages in which the operations and internal representation of a computational entity are partially removed from external view. Abstraction allows the results of the entity's computation to be seen while hiding the means by which the computation is performed. An object is thus effectively divided into two parts: an interface part and an implementation part. The interface part is the object's protocol, or the messages to which it will respond. The implementation part is the object's internal state and methods.

To return to the example, when the AIRCRAFT object receives the COMPUTE-GROUNDSPEED message, the details of the method performing the computation and the updates to the object's internal state are not seen. However, the new groundspeed and direction of the AIRCRAFT object may be obtained using the object's protocol; by sending SHOW-GROUNDSPEED and SHOW-DIRECTION messages to it which cause the values of AIRCRAFT-GROUNDSPEED and AIRCRAFT-DIRECTION to be returned.

2.4 Classes of Objects

Objects having the same data structures and behavior can be considered to be instances of the same type, or class. A class has a description consisting of a set of common data structures, known as instance variables, a common protocol consisting of the set of messages that class instances will respond to, and the set of methods for implementing common operations.[3] Classes are also sometimes referred to as abstract data types [JOSE89].

Class descriptions serve as templates from which new objects are created. Many classes may be defined in a

[3] Not all object-oriented programming languages support the class concept. For an example of an object-oriented language without classes, see [UNGA87].

particular application. Each object is an instance of one of the classes and has the same data structures provided in its class description. However, the object ID or handle of each instance is unique, and the internal states of different instances, e.g., the values of their instance variables, may differ. Each object responds to the same messages and implements the same operations defined in the methods of the object's class.

From the database viewpoint, a class can be thought of as a record type consisting of the metadata which provides all the information necessary to construct and use a particular object. Similarly, it is possible to think of the instances of a class as being records stored in a file. New records having different values can be added to the file. A data dictionary for the database management system may contain descriptions of many different record types each with a different set of attributes.

2.5 Inheritance

Complex applications which make use of objects, such as CAD systems, often require the definition of many different classes. It is possible to have a separate class definition for each kind of object needed. However, if some objects are more specific kinds of other objects, it would be natural to take advantage of this relationship when defining classes. This could be done by having class definitions in which a specific class can share or borrow part of the definition of a more general class.

The mechanism for creating a class definition that derives instance variables and methods from another class definition is called <u>inheritance</u>. When a class inherits instance variables and methods from another class, it is referred to as a <u>subclass</u>. The class from which the instance variables and methods are inherited is a <u>superclass</u>. The concepts of superclass and subclass are analogous to the concepts of <u>generalization</u> and <u>specialization</u> familiar in data modeling methodologies [SMIT77].

In the example, PASSENGER_AIRCRAFT and CARGO_AIRCRAFT may be specialized types of AIRCRAFT. A PASSENGER_AIRCRAFT would have all the attributes and derive much of its behavior from AIRCRAFT. But it may have additional attributes, such as number of passengers. PASSENGER_AIRCRAFT may also have a more specific behavior not shared with AIRCRAFT, determined by constraints such as maximum rate of descent.

PASSENGER_AIRCRAFT can thus be defined as a subclass of AIRCRAFT, inheriting all the instance variables in the definition of the class AIRCRAFT, but also one of its own: NUMBER_OF_PASSENGERS. All instances of PASSENGER_AIRCRAFT will have the instance variables of AIRCRAFT, but will also have NUMBER_OF_PASSENGERS.

Similarly, PASSENGER_AIRCRAFT may inherit SHOW-GROUNDSPEED and SHOW-DIRECTION methods from AIRCRAFT, but have a separate SET-ALTITUDE method defined for it with special constraints on maximum rates of descent. The SET-ALTITUDE method defined in PASSENGER_AIRCRAFT replaces or shadows the method of the same name defined in AIRCRAFT. Additional methods may also be defined for PASSENGER-AIRCRAFT giving it the capability to respond to messages that its superclass cannot.

2.6 Class Hierarchy

With inheritance, class hierarchies can be created which reflect natural relationships found in real world domains. For instance, we may further specialize PASSENGER_AIRCRAFT into subclasses, for example, COMMUTER_PLANES and AIRLINERS. Figure 2.1 shows an example of class hierarchy.

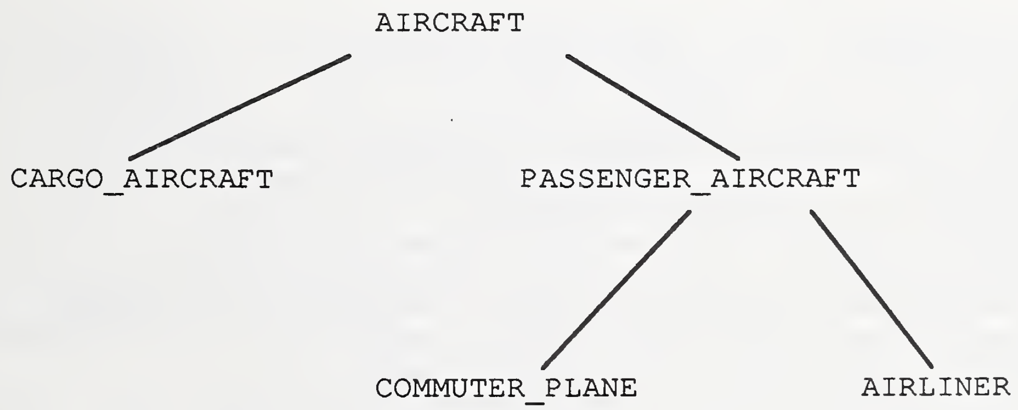

Figure 2.1 - Example Of Class Hierarchy

By extensive inheritance of instance variables and methods, class hierarchies can be created which significantly reduce the amount of code needed to implement an application. For this reason, object-oriented programs are said to promote code reuse.

It is also possible for a class to have more than one superclass. This is known as multiple inheritance. Object-oriented languages which support multiple inheritance can be used to produce class structures which are nonhierarchical. For this reason, multiple inheritance is often advantageous in programming applications with more complex modeling requirements. Issues relating to multiple inheritance are discussed in [CANN82] and [CARD90].

2.7 Polymorphic Behavior and Run Time Binding

Polymorphism adds an important dimension to an object's behavior. Polymorphism means that an object's response to a message is determined by the class to which the object belongs.[4] Instances of different classes can be addressed in a uniform manner, e.g., receive the same messages, yet exhibit different behaviors.

In the example, polymorphic behavior can be achieved by first defining two classes of airplanes: PASSENGER_AIRCRAFT and CARGO_AIRCRAFT. Then, a COMPUTE-FUEL-CONSUMPTION method may be defined for each class, but with a different procedure specified in each method definition. Instances of either class can respond to COMPUTE-FUEL-CONSUMPTION messages, however the calculation of the fuel consumption rate will differ depending on the class of the airplane object.

Another kind of polymorphic behavior is associated with inheritance. An object's response to a message can be determined by methods inherited from a superclass. For instance in the example described above, PASSENGER_AIRCRAFT inherited SHOW-GROUNDSPEED and SHOW-DIRECTION methods from AIRCRAFT. When a SHOW-GROUNDSPEED message is sent to an instance of PASSENGER_AIRCRAFT, the response is made by the method obtained from the definition of AIRCRAFT.

Multiple Inheritance permits definition of complex forms of polymorphic behavior which can sometimes involve combining methods from two or more superclasses.

Polymorphic behavior is closely associated with run time binding, or dynamic binding. Run time binding means that the selection of the method to respond to the message is made at run time, as opposed to compile time. This means that the precise method which will be executed is not known until run time. The selection is actually performed by an internal mechanism maintained by the system for this purpose.

[4] More generally, polymorphism refers to being able to apply a generic operation to data of different types. For each type, a different piece of code is defined to execute the operation.

2.8 Composite Objects

Composite objects, or complex objects, are among the most recent developments in object technology. A composite object is made up of other objects. In other words, it is constructed from a collection of parts, each of which is itself an object.

For instance, an AIRCRAFT might be defined as a composite object consisting of separate parts for JET_ENGINE, FUSELAGE, COCKPIT, etc. Each component part is an object and an instance of a class. The process can be extended to produce a part hierarchy, illustrated below in Figure 2.2.

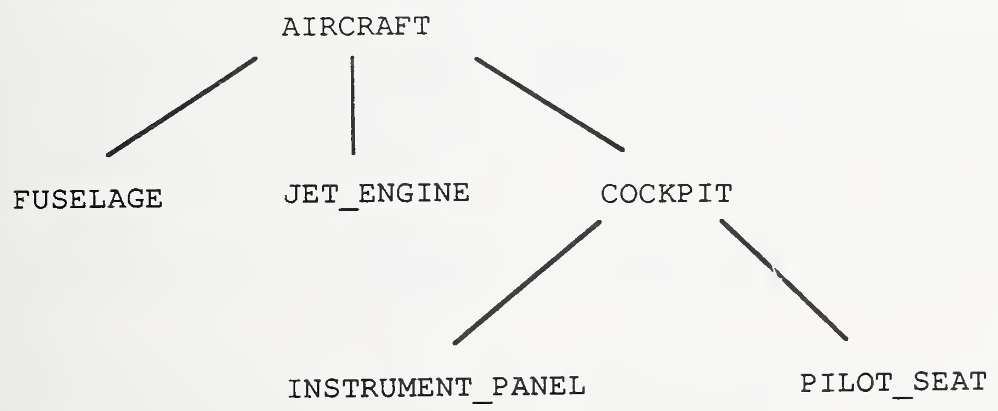

Figure 2.2 - Example of a Part Hierarchy

Individual components are said to be in an "Is-Part-Of" relationship to the composite object. For instance, INSTRUMENT_PANEL and PILOT_SEAT are in an "Is-Part-Of" relationship to COCKPIT.

Composite objects have proven useful in many application domains. Some examples are the modeling of part hierarchies in engineering design systems, and representing spatial data in geographic information systems.

Composite objects have special requirements for generic operations on their components [DAYA90]. These include operations for manipulation of the composite object as a whole, for manipulation of component subsets; and the ability to define operations which can be propagated to components via relationships. Experimental implementations which support composite objects are described in [DAYA90], [BLAK87], and [KIMW87a].

3.0 OBJECT-ORIENTED PROGRAMMING LANGUAGES

The purpose of this section is to describe how object-oriented programming languages (OOPL) developed and to provide an understanding of what an object-oriented language is.

3.1 The Development of Object-Oriented Programming

The forerunner of object-oriented programming languages is Simula [DAHL66]. Invented in the 1960s, Simula was based on the Algol language. Simula was intended for simulation of real world events and is usually credited with introducing the concepts of encapsulation and class [HORO83]. One of the first programming languages that supported objects as computational entities was CLU [LISK77].

The term "object-oriented" originated during the course of the development of Smalltalk [GOLD83], [HORO83]. Smalltalk resulted from a research effort undertaken in the 1970s at the Xerox Palo Alto Research Center to develop new and more productive ways of programming. The product of this work was Smalltalk-80, a comprehensive software development system which included the Smalltalk programming language, an interactive development environment and operating system, and a personal computer workstation.

Smalltalk was the first programming language in which all information was represented in objects and in which message passing was the dominant mode of computation. The concepts of class, inheritance, and class hierarchy became fundamental parts of Smalltalk in its early stages.

The advent of Lisp machines in the early 1980s gave further impetus to the development object-oriented languages. Among these were Flavors, an object-oriented programming system implemented on the Symbolics Lisp Machine, and LOOPS, an object-oriented programming system developed at the Xerox Palo Alto Research Center [STEF83]. Influenced heavily by Smalltalk-80, Lisp-based object-oriented programming systems were crucial to the design and implementation of Lisp machine operating systems and graphics software [CANN82]. At the writing of this report, an ANSI Common Lisp standard is being developed for the Lisp Language which will include a Common Lisp Object System (CLOS) [BOBR88].

In the middle 1980s, Bjarne Stroustroup introduced an object-oriented system for the C language called C++ [STRO86]. This language implemented most of the object concepts presented in section 2 of this report and quickly became popular among C developers.

More recently, a number of commercial programming languages have facilities which include at least some of the object concepts presented in section 2. These include such languages as ADA and OBJECT PASCAL.

3.2 Characteristics of Object-Oriented Programming Languages

The commercial object-oriented languages of today vary significantly as to which features they support. Yet, for the purpose of this report, it is useful to have some sense of which of the concepts described in section 2 constitute a core set of features available in an object-oriented language.

The following list constitutes a generalized intersection of features supported in the most widely used object-oriented languages, e.g., Smalltalk, C++, and the CLOS:

- objects as definable programmatic entities,
- object encapsulation in some degree or form,
- definition of object classes,

- message passing,
- definition of class methods,
- run time binding,

- class inheritance,
- polymorphism.

Multiple inheritance and complex objects, while available in some languages, are excluded from the list presented here. There are also popular languages which do not support classes or class inheritance but which are sometimes regarded as object-oriented [UNGA87].

3.3 The Need for Persistence and Data Sharing

Data created by computer programs is transient, existing only in the virtual memory system of the computer. Once the program terminates, or if the machine crashes, the data may disappear or be lost. Data persistence means that the data a program uses can exist after the program terminates. Persistent data is stored permanently on a secondary storage device, such as a disk.

Similarly, in most programming languages, access to data resident in the workspace of a computer program cannot be shared among several users in a controlled manner which ensures that the consistency and integrity of the data is maintained. That is, mechanisms to control concurrent access to data are not generally available in programming languages.

Yet, data persistence and concurrency control are necessary for applications which require large amounts of data that must be shared among many users. During the 1980s, experimental programs using OOPL became more common, finally resulting in the first commercial applications. As the amount of data manipulated by object systems increased, and as the object paradigm has begun to be used in design applications and in other sophisticated domains of endeavor, the need for persistence and data sharing became obvious. The solution to this need has been to combine OOPL with DBMS capabilities, resulting in the emergence of object database management systems.

Similar requirements motivated the development of the first database management systems 2 decades ago.

4.0 DATABASES, DATA MODELS, AND DATABASE MANAGEMENT SYSTEMS

This section describes those database concepts necessary to support the requirements for an ODBMS. The combination of object concepts with features of DBMS forms the basis for an ODBMS.

4.1 Databases

The most important element of data management applications is the data which resides in secondary storage. Every database is a model of some portion of the real world. Applications access the database and use this model. At all times, the contents of a database represent a snapshot of the state of an application environment. Therefore, it is appropriate that the structure of a database mirror the structure of the system that it models.

4.2 Data Models

The pattern used to organize the database and the relationships within the data is called the data model [CODD81]. A data model can be considered as consisting of three components:

- o Data structure - the basic building blocks describing the way data is organized.
- o Operators - the set of functions which can be used to act on the data structures.
- o Integrity rules - the valid states in which the data stored in the database may exist.

The primary purpose of any data model is to provide a formal means of representing information and a formal means of manipulating the representation. A good data model can help describe and model the application effectively.

Data models differ in their capabilities for expressing properties of applications. The three popular data models for most commercially available DBMS are the relational model, the hierarchical model, and the network or CODASYL model.

4.3 Database Management Systems

A database management system understands the structure of the data, and provides a language for defining and manipulating stored data. The primary functions of the database management system are to store data and provide operations on the database. The operations usually include create, delete, update, and search of data.

Not all DBMS provide the same set of features, and varying degrees of functionality exist for the same features. Some essential features generally supported by a modern DBMS are summarized as follows:

- **Persistence**

  Persistence is the property wherein the state of the database survives the execution of a process in order to be reused later in another process.

- **Data Sharing**

  Data sharing is the property that permits simultaneous use of the database by multiple users. A DBMS that permits sharing must provide some <u>concurrency control</u> mechanism that prevents users from executing inconsistent actions on the database. Locking and serializability of operations are concurrency control mechanisms offered by a DBMS.

- **Recovery**

  Recovery refers to the capability of the DBMS to return its data to a consistent and coherent state after a hardware or software failure.

- **Database Language**

  The database language permits external access to the DBMS. The database language may include the Data Definition Language (DDL), the Data Manipulation Language (DML), the Data Control Language (DCL), and an ad hoc query language. The DDL is used to define the database schema. Sometimes, a separate schema manipulation language may also exist to modify schema definitions. The DML is used to examine and manipulate contents of the database. The DCL is used to specify parameters needed to define the internal organization of the database, such as indexes, buffer size, etc. An ad hoc query language is provided for interactive specification of queries.

The DBMS, and the ODBMS, may offer additional desirable features, such as versioning, view definition, integrity checking, security and authorization control, database administration utilities, etc.

4.4 Databases, Database Management Systems, and Database Systems

It is important to distinguish between the terms "database," "database management system," and "database system." These terms can be defined as follows:

- **Database (DB)**

  The term database refers to the stored data. The stored data includes the schema information (or metadata) that defines the structure of the data and the actual data itself. An object database (ODB) will therefore refer to the database schema together with the stored application data. In an ODB, the database schema is described in class definitions which also include the definition of methods. In an ODB, the stored data consists of objects.

- **Database Management System (DBMS)**

  The DBMS is a software system that provides capabilities for the definition and manipulation of databases. The DBMS also provides additional functionalities such as recovery, concurrency control, security, etc. An object database management system (ODBMS), thus, will be considered to be a software system that provides DBMS functions together with the mechanisms for definition and manipulation of ODB. The ODBMS provides a connection between the application program and the database.

- **Database System (DBS)**

  The DBS consists of the complete set of application programs, the database management system, and the stored data. In this report, an object database system (ODBS) will consist of ODB, ODBMS, and the set of user applications.

5.0 THE OBJECT DATA MODEL

In the past 2 decades, many alternative data models have been formulated, including the entity-relationship data model [CHEN76], the semantic data model [HAMM81], and the functional data model [SHIP81], etc. In all of these efforts, the goal has been to provide a more natural and richer way of representing the semantics of complex application domains.

Several papers have been written describing the object data model including [MARI88] and [BANE87]. So far, there is no complete consensus on the object data model. This section summarizes concepts that have been generally accepted as belonging to the object data model.

5.1 The Structural Aspects of the Object Data Model

Objects, classes, and inheritance form the basis for the structural aspects of the object data model. More specifically, this means the following:

- o Objects, as presented in section 2, are basic entities which have data structures and operations.
- o Every object has an object ID that is a unique, system provided, identifier.
- o Classes describe generic object types. Class descriptions are used to create individual objects, or class instances. All objects are members of a class.
- o Classes are related through inheritance. Classes can be related to each other by superclass or subclass relationships to form class hierarchies.
- o Class definition is the mechanism for specifying the database schema. The <u>database schema</u> consists of all the classes that have been defined for a particular application. Class definitions include both inheritance relationships (superclass to subclass) and structural relationships between classes (analogous to relationships in the entity-relationship-attribute model).
- o A complete database schema may consist of one or more class hierarchies together with structural relationships. Individual schema descriptions refer to the instance variables of individual classes.
- o The definition of a class can include instance variables having any allowable system defined or user defined type, including types consisting of other classes. For

example, the class PERSON has an attribute SPOUSE whose data type is also PERSON.

The database schema can be dynamically and incrementally extended by the definition of new classes. <u>Extensibility</u> refers to the generic ability to define and add new data types, including those types unavailable in conventional database management systems such as voice, graphics, and images. In the object data model, the database schema may be extended by defining and adding new classes that contain data structures and operations for representing and manipulating unconventional data types.

5.2 The Operations of the Object Data Model

As described in section 2, message passing is the basis for the operations of the object data model. The operations may be described as follows:

- Objects communicate through messages. Methods are procedures that determine how an object responds to a message. Methods are defined for classes.

- Polymorphic behavior is a consequence of being able to send the same message to instances of different classes. An object's behavior in response to a message is determined by selecting the appropriate method defined for the object's class or for a superclass.

- The object data model supports operations to create and delete class instances.

- The object data model supports operations to create, delete, and modify class definitions. Class modification is analogous to schema modification or redefinition in conventional DBMS.

- A class instance may be updated by methods that change the values of its instance variables, thus changing the object's internal state. In a conventional DBMS, this is analogous to updating fields in a single record.

It is important to note that in several implementations, including [GOLD83] and [BOBR88], class definitions are themselves objects, called <u>class objects</u>. Class objects are instances of a generic class, or metaclass. Hence, operations to create, modify, and delete class definitions can also be implemented as messages.

5.3 The Integrity Rules of the Object Data Model

In the object data model, integrity rules are a consequence of the model's structure and operations. The following rules are important to note:

o All objects must obey the protocol specified by their class definitions. That is, an object can respond only to those messages allowed by the class it belongs to.

o Objects are encapsulated. Encapsulation is achieved by limiting access to objects through use of the message protocol defined for the object's class.

o In principle, object ID supports referential integrity. Unique object IDs are assigned to each object when it is created. An object does not exist unless an object ID has been assigned to it. If an object is deleted or removed, its object ID is also deleted, thus preventing any reference to the deleted object.

However, commercial ODBMS do not necessarily support all of the implications of referential integrity. For instance, the concept of cascading deletes is not currently supported by most commercial ODBMS. That is, when an object is deleted, it is not possible to specify the automatic deletion of other objects that reference the deleted object.

5.4 The Object Data Model versus the Relational Data Model

While the pure object theorist may not agree with equating object data model concepts with those of the relational data model, the following tables provide such equivalences. It is presented not to demonstrate any direct equivalence, but is intended to serve merely as an intuitive aid for persons already familiar with mature database technology.

Table 5.1. Comparison of Structure

Object Model	Relational Model
Class hierarchy	Database schema
Class definition	Table definition
Class instance (object)	Tuple or record
Instance variable	Column or field
Object ID	Primary key or identifier
Class inheritance	<no correspondence>[a]

Table 5.2. Comparison of Operations

Object Model	Relational Model
Define, delete and modify a class or instance variable	Define, delete and modify a relation or field
Define, delete and modify method	<no correspondence>
Create class instance, e.g., create an individual object	Add a row (record) to a table (file)
Send/receive message	<no correspondence>
<no correspondence>[b]	Select
<no correspondence>	Join
<no correspondence>	Project

[a]The emerging SQL3 standard may include the ability to define schema hierarchies.

[b]In practice, many ODBMS provide a query language with Select, Join, and Project operations.

Table 5.3. Comparison of Integrity Rules

Object Model	Relational Model
Object protocol	<no correspondence>
Encapsulation	<no correspondence>
Object ID	Referential integrity

The structure, operators, and integrity rules of the data model are realized in the object database management system. Section 6 describes the overall architecture of the ODBMS.

6.0 THE ODBMS ARCHITECTURE AND FEATURES

The overall framework of a typical ODBMS and the major components of the architecture will be discussed in this section. This will be followed by more detailed descriptions of the individual features of the ODBMS including the database language, message processing, facilities for schema definition and modification, transaction management, storage management, and security and semantic integrity considerations. Finally, the distributed features of ODBMS will be shown.

6.1 Overview of the ODBMS Architecture

The term architecture refers to an abstract description of the organization of a system for purposes of showing functional components and the interfaces between them. There is no single agreed upon architecture for ODBMS. Moreover, architecture can be a matter of perspective, being dependent upon whether the ODBMS is being viewed by a developer, end-user, or system administrator.

To aid the illustration of features, the ODBMS architecture will be characterized as consisting of three major components:

1) The Object Manager, which provides the interface between external processes and the ODBMS;

2) The Object Server, which is responsible for providing basic DBMS services such as transaction management and object storage management; and

3) The Persistent Object Store, or the ODB, itself.

External processes may be generated by various users accessing the ODBMS. Figure 6.1 shows the basic components of the ODBMS architecture.

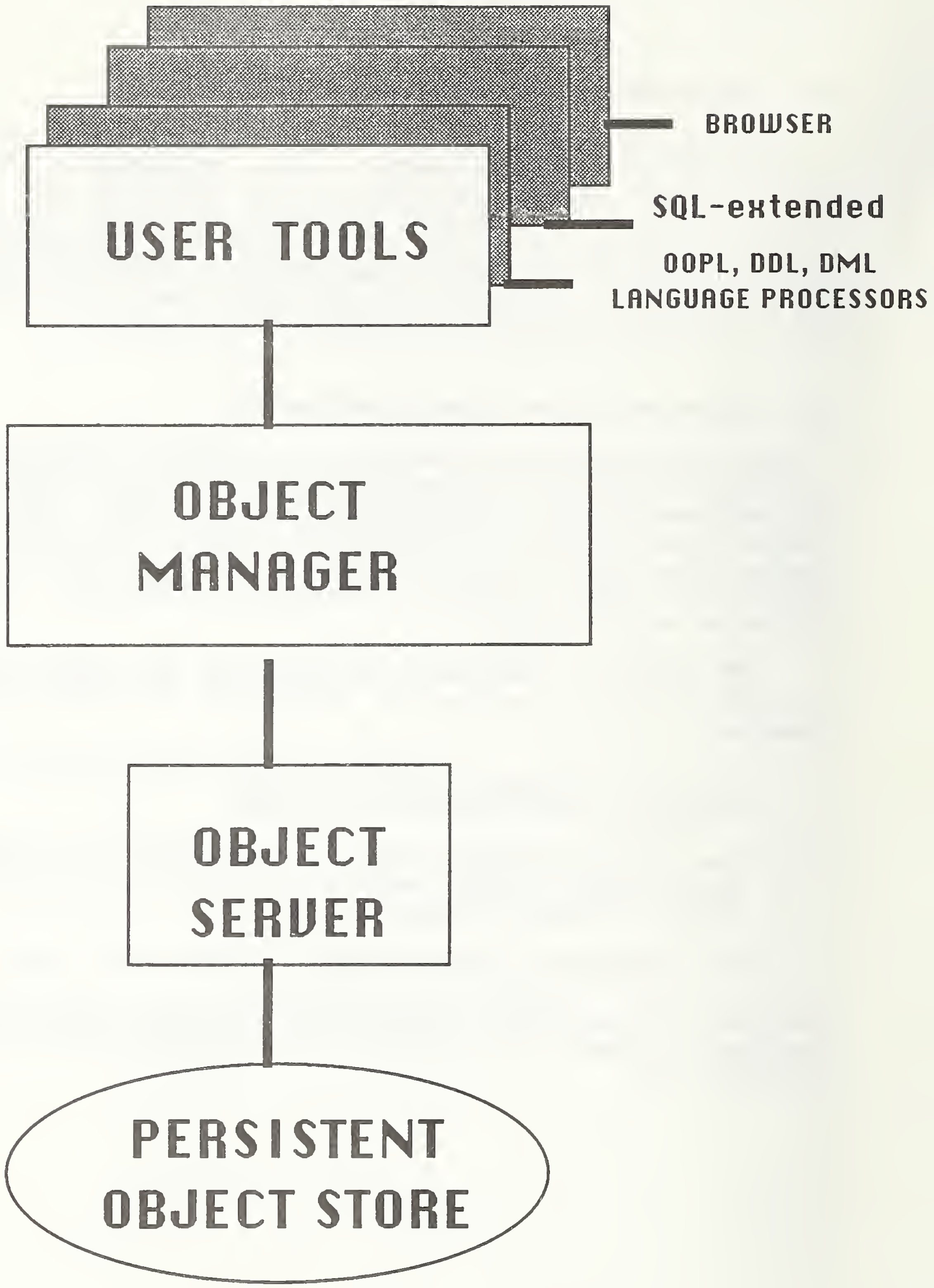

Figure 6.1. Overall Architecture of the ODBMS.

External end users and application developers may use software tools such as text editors, graphics editors, class and object browsers, automated database design aids, and CAD/CAM design system interfaces. These systems may serve as front-end tool kits that interface with the Object Manager.

The Object Manager provides a complete implementation of the object data model to the external developer or user. This would include the ability to define the structures and execute the operations specified by the model.

The Object Manager receives requests to create class definitions, modify existing class definitions, handle messages generated by an executing application program, and process ad hoc queries using a query translator. The Object Manager performs run time binding, necessary syntax and type checking operations. Requests are then submitted to the Object Server as transactions.

The Object Manager's functions, as depicted in figure 6.1, consist of:

1) performing message processing functions including run time binding and type checking, as well as query translation; and

2) providing facilities for definition and modification of the database schema including integration of new or revised class definitions into existing class hierarchies or lattices.

The Object Server manages the actual retrieval, insertion, deletion, and update of stored objects in the persistent object store. A single server may handle transactions submitted from more than one Object Manager. The Object Server's functions consist of:

1) transaction management including concurrency control, buffer management, and recovery services; and

2) physical storage management including the placement of objects and implementation of access methods.

Backup and archiving services may also be provided by the Object Server.

It must be noted that the features identified for Object Manager and Object Server are, to some extent, arbitrary. For example, Object Server can be a simple fetch system based upon a physical address on disk or it can be a complete back-end

DBMS. More discussion on back-end DBMS will appear later in this section.

6.2 The ODBMS Database Language

At the present time, commercial ODBMS are accessed primarily through object-oriented programming languages such as Smalltalk, Common Lisp, and C++. The interface between the OOPL and the ODBMS is the database language. As discussed in section 4.3, a DBMS must provide a database language to permit definition and manipulation of the database schema and the stored data. The ODBMS must have a database language to permit access and manipulation of the object data model and to retrieve and update objects.

In contrast to many conventional DBMS, the ODBMS database language is firmly embedded in the OOPL. That is, database language statements are not part of a separate language having its own interpreter. Since OOPL existed before ODBMS, many DDL and DML statements are adaptations of previously existing OOPL statements. The ODBMS database language consists of the following:

o **Data Definition Language (DDL)**

The ODBMS must provide a DDL for schema definition. The DDL must permit definition of classes including inheritance links and definition of methods that specify object behavior. The DDL also must be able to specify additional constraints and semantic integrity rules if appropriate.

o **Data Manipulation Language (DML)**

The ODBMS must provide a DML for retrieving, creating, deleting, and updating individual objects. In the ODBMS, this is achieved through the message passing mechanism.

o **Ad Hoc Query Language**

Nearly every commercial DBMS supports retrieval of database subsets by specifying logical conditions based on values using an ad hoc query language. The object data model, as presented in section 5, permits retrieval of individual objects referencing the object's ID. To provide for subset retrievals over groups of objects, some ODBMS (and some OOPL implementations) provide an ad hoc query language [FISH89], [KIMW89]. In many implementations, the query language relies on message passing for selection and retrieval of objects (this will be discussed in subsequent sections).

The ODBMS can be expected to interface to one or more commercial object-oriented programming languages. The issue of having one database language for all OOPL interfaces is discussed in section 7.

As stated in section 5.2, class definitions (including methods) are implemented as class objects. DDL statements to create class definitions are messages to a generic class, or metaclass, to create an instance of itself, e.g., a class object. Therefore, in keeping with the object paradigm, all of the operations specified by the ODBMS database language can be implemented through message passing.

6.3 Message Processing

The Object Manager provides the interface between external processes and the ODBMS. It receives messages for individual objects, performs run time binding and type checking operations, and dispatches the external request for objects to the Object Server. To perform these services, the Object Manager must have access to a copy of the class definitions being used by the processes that are currently active.

Message passing and query processing can be summarized as follows:

- o **Session Control**

  This includes necessary functions such as maintaining the external user's local workspace for database operations.

- o **Run Time Binding**

  As discussed in section 2, run time binding refers to the selection of a method for a message sent to an object at run time. The basis for the selection is the class hierarchy to which the object belongs. If not directly available, the Object Server may be requested to provide the correct method. Run time binding is the mechanism by which polymorphic behavior is implemented.

- o **Creation of New Objects**

  Creation of new class instances must be initiated in the Object Manager and object IDs must be assigned. If type checking on instance variable values is enforced, these values must be checked to ensure they are of the type defined for the variable and are within a valid range, if this range is defined.

o **Dispatch of Object Requests and Object Updates**

External requests for objects, newly created objects, and updated objects, must be forwarded by the Object Manager as transactions to the appropriate Object Server.

o **Query Translation**

To support an ad hoc query language, an ODBMS should have a query translator and perhaps a query optimizer. Queries can be translated into execution plans in which selection and retrieval of objects is accomplished through message passing. This presupposes that the message protocol for the object's class is defined to permit access to the instance variables necessary to select the object. Support for ODBMS query languages raises issues that will be discussed in section 7.

Objects, class definitions, and methods requested by the Object Manager are retrieved by the Object Server from the Persistent Object Store.

6.4 Schema Definition and Modification

Support for schema definition and modification consists of the following:

o **Providing Access to Existing Class Definitions**

Class definitions may themselves be implemented as objects and stored internally as such. Definitions of all classes provided by the ODBMS, as well as classes created by developers, can be stored permanently in the Persistent Object Store, possibly in a <u>class library</u>, or a data dictionary. Class libraries may consist of a set of specific classes used in a specific domain. The Object Manager must make available to the developer those class definitions necessary for particular applications.

o **Extensibility of the Database Schema**

This includes processing database language statements specifying creation, removal, or modification of class definitions. It must also include support for class inheritance. Additions and modifications of class definitions must be checked for syntactic correctness and to ensure that they constitute valid changes to the database schema in accordance with the inheritance rules in effect. Newly created or modified class definitions must be forwarded as transactions to the Object Server.

o **Dynamic Class Redefinition (Schema Evolution)**

Prototyping of complex applications requires frequent changes to the database schema, even after the database has been populated with objects. Therefore, the ODBMS must provide not only the ability to dynamically modify class definitions, but it must also ensure structural consistency between modified definitions and previously existing objects of that class. Further, it must ensure consistency between references in existing programs and objects of changed classes. The actual modification of the structure of existing class instances would be performed by the Object Server.

The Object Manager submits requests for retrieval and update of class definitions to the appropriate Object Server. The Server processes these operations as transactions.

6.5 Transaction Management

Transaction management is an important service provided by the Object Server. The ODBMS transaction management mechanism may receive requests for retrieval or update of stored objects and class definitions from one or more Object Managers. Transaction management consists of the following features:

o **Support for Sharing and Concurrency Control**

The ODBMS must support concurrent sharing of data by multiple users. In order to maintain database integrity when concurrently executing transactions are attempting to access the same objects, the ODBMS must provide a mechanism for guaranteeing that such transactions are serialized. Alternative implementations of concurrency control have been proposed and implemented in commercial systems, including both optimistic concurrency control and pessimistic concurrency control based on locking.

o **Transfer of Objects between Secondary Storage and User Workspace and Buffer Management**

Retrieving data from secondary storage is a basic function provided by any DBMS. Data retrieval is accomplished through access paths discussed in section 6.7. Management of buffers may be the responsibility of either the ODBMS or the host operating system.

o **Recovery**

In the event of transaction failure or hardware failure, the database must remain intact and consistent. A transaction log must be maintained for this purpose. This is a service provided by most commercial DBMS.

In addition, the transaction processing requirements associated with design activities has motivated the proposed addition of advanced transaction management capabilities to object database systems. These include:

- **Support for Cooperative Transactions**

  During development of design applications, a portion of the design is often worked on by a group of individuals in a cooperative effort. This effort could last several hours or even days. Substantial numbers of objects (possibly composite objects) constituting the design may be updated in extended sequences of transactions taking place in long design sessions involving several cooperating users. Under these circumstances, individual transactions may be required to communicate, or even interact, with each other. The established criteria for strict serialization, based on locking protocols, must be relaxed and replaced by more flexible, design specific criteria. This more flexible criteria might ensure that the updated information is correct, but also permit a more random mix of transactions. For further information, see [SKAR89].

- **Support for Versioning**

  Version management is a facility for tracking and recording changes made to data over time. Version management systems are essential for recording the history of design changes. A versioned object may have a number of alternative states, each of which corresponds to a distinct version of the object over time. The version management system tracks version successors and predecessors. When objects constituting a portion of the design are retrieved, the system must ensure that versions of these objects are consistent and compatible. Prototype implementations of version management systems in ODBMS are described in [FISH89] and [KIMW88b].[5]

Cooperative transaction processing and versioning appear to be probable requirements for future CAD systems, office information systems, and sophisticated artificial intelligence systems. At the writing of this report, these features are

[5]Versioning is considered part of the larger topic of change management. For further information about versioning, change management, and object database management systems, refer to [BJOR89], [JOSE90], and [FORD88].

being studied extensively in universities and research institutions, but have not yet been commercialized.

Transaction management relies on storage management to perform the actual retrieval and update of stored objects.

6.6 Storage Management

Storage Management refers to maintenance of the physical level organization of the ODB and support for access paths necessary to ensure efficient access to stored objects. Storage management is performed by the Object Server, which may partially be implemented with a conventional DBMS back-end. Basic data storage functions may be characterized as consisting of:

- **Support for Persistence**

  Objects that are created and added to the ODBMS must be retained after the session ends. That is, objects must be persistent. They must be maintained in file structures in secondary storage for access by subsequent users. Just as in a conventional DBMS, file level structures for storage and access must be supported.

- **Support for Large Objects**

  The ODBMS must be able to support storage and manipulation of variable length objects of any size, including those which span more than one physical storage block. This is necessary to ensure that the database schema can be extended to support a wide variety of applications. For instance, voice or image applications require unusual data types involving long streams of data, sometimes called Binary Large Objects (BLOBs).

- **Backup/Archiving Facilities**

  Just as in any DBMS, it must be possible to offload the Persistent Object Store to an offline medium such as magnetic tape for archiving purposes.

The complexity of logical organization of information in object databases, the large quantity of stored objects, and the volume of transactions which access objects (either individually or in groups) make performance a critical issue for the ODBMS. Ensuring fast response times in an ODBMS requires special indexing mechanisms together with use of advanced capabilities to optimize physical organization of data. Features supporting retrieval and update of stored objects may consist of some or all of the following:

- **Support for Access Paths**

  Support for access paths is necessary to ensure efficient retrieval and update of data stored in large databases. This includes indexing of objects on the basis of Object ID, to permit efficient retrieval of individual objects, as well as indexing of objects by instance variable values, for retrieval of subsets of objects in range queries.

- **Specialized Types of Indexes for Objects**

  To improve performance, special forms of indexes have been developed for objects. Compound indexes may be specified combining several instance variables belonging to objects of possibly different classes. These objects may be related in such a way that an instance variable of one object has as its value the Object ID of one, or possibly several, different objects. The relationship may be extended to form complex paths of objects, particularly in the case of composite objects. Creating indexes on these paths permits rapid access of groups of objects. It also speeds navigation through networks of interconnected objects.

- **Object Clustering**

  Clustering of objects in the same secondary storage sector is also important. This permits efficient retrieval of groups of objects which are accessed together, particularly in the case of composite objects.

- **Segmentation of Stored Objects**

  Similarly, large objects may be divided and stored in separate physical structures to enhance performance. For instance, there may be objects that have instance variables that are frequently accessed and instance variables with long text or character strings that are seldom retrieved. Frequently accessed instance variables can be stored in one record structure while rarely accessed instance variables can be stored in another.

It should be noted that, in general, use of access methods, clustering, and segmentation are techniques that are not peculiar to ODBMS, but may also be used for optimization of performance in any DBMS. However, these techniques can be expected to have special relevance in an ODBMS. In the case of clustering, it can be a particularly useful mechanism for storing objects referenced together such as composite objects. Access paths, clustering and segmentation may be used by a database administrator to fine tune the physical database design of the ODBMS for improved performance.

6.7 Security and Semantic Integrity

Just as in conventional database management systems, security and semantic integrity features are important considerations.

- **Security**

  Security refers to the protection facilities offered by a DBMS to prevent unauthorized access to the database. There are many ways of authenticating users and controlling access to data.

- **Semantic Integrity**

  Semantic integrity refers to the declaration of semantic and structural integrity rules and the enforcement of these rules. There are many different kinds of rules, such as typing constraints, values of domain constraints, uniqueness constraints, etc. Depending on the implementation of the ODBMS, integrity rules may be automatically enforced at run time, at compile time, or may be performed only when a message is sent.

6.8 The Distributed Features

The requirements of design applications may necessitate that the ODBMS exist on several hardware platforms that can communicate over a computer network. In a CAD design system, several designers using software development tools, working on different platforms, may access data stored in the ODBMS. Using a CAD tool, each designer may conduct an interactive session for which an individual copy of the Object Manager is created as a process. More than one Object Manager may submit transactions to the same Object Server concurrently.

The Object Server with which the Object Manager communicates may also exist on the same platform as the Object Manager, or the Manager and Server may exist on different platform. Similarly, an organization with a set of interrelated design activities may require more than one Object Server, each of which manages a separate Persistent Object Store. To facilitate communication across separate hardware platforms, network communications software may be required. Figure 6.2 illustrates a networked system with several Object Managers and Servers.

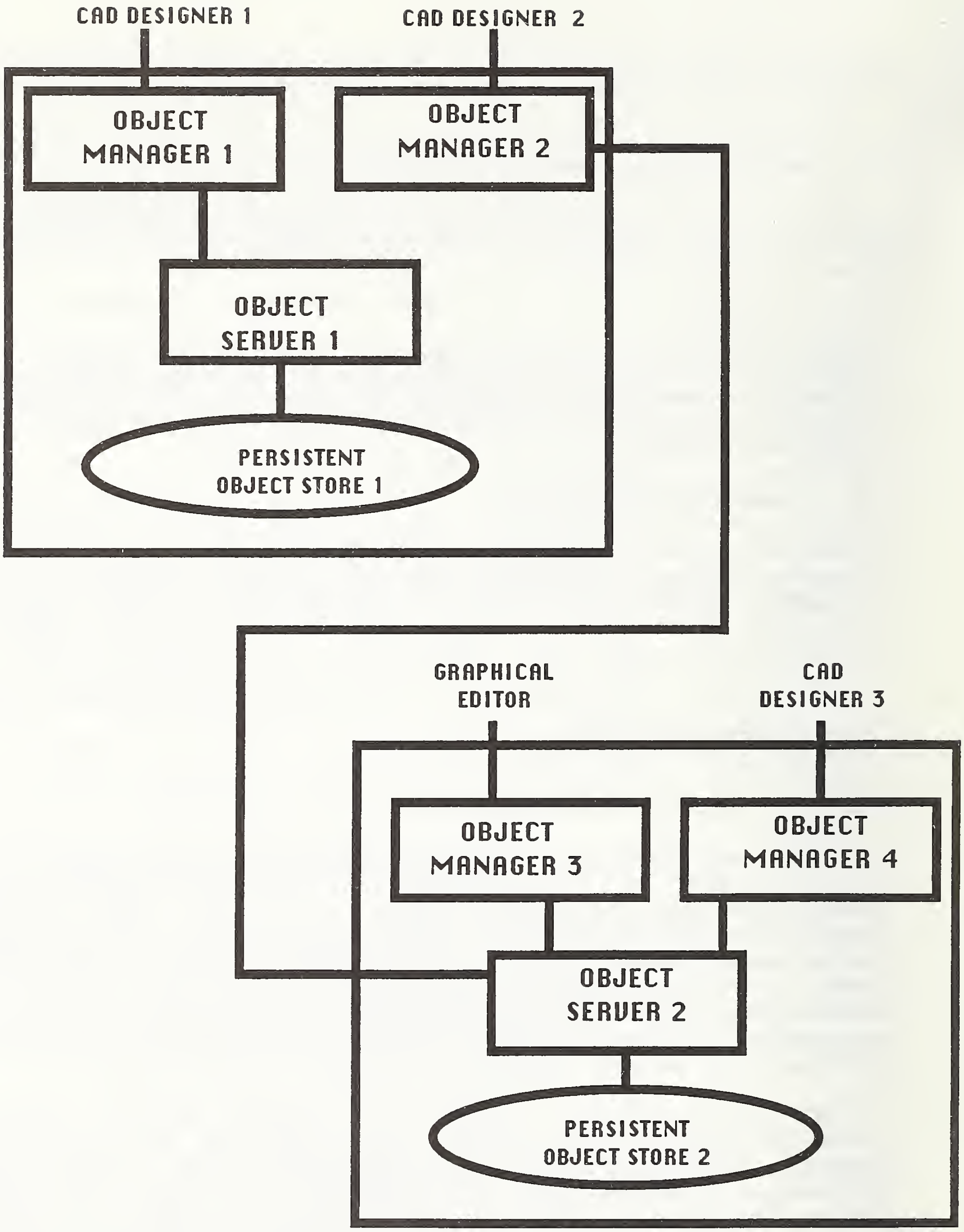

Figure 6.2. The ODBMS Networked Architecture.

In some implementations, the functions of the Object Server can be partially implemented by a relational DBMS. In this case, the Server consists of two distinct components: a front-end module which maintains an internal representation of a limited portion of the object data model and handles requests for objects in the form of Object IDs, and a back-end relational DBMS which is responsible for transaction management and storage of object data in relational tables. The back-end component may itself exist on a different platform. In such cases, a specialized remote data access (RDA) protocol based on the SQL standard can be used. Currently, an RDA standard is under development [ISO89].

In order to support distributed processing, the ODBMS must automatically manage the access of objects stored on separate hardware platforms. The linkage between Object Manager and Object Server must be explicitly initiated by the system user. As the complexity of design applications and organizational usage increases, the emergence of distributed ODBMS can be expected.

7.0 RECENT DEVELOPMENTS IN ODBMS

Efforts at developing ODBMS have generally proceeded by combining the special nature of object-oriented programming with DBMS. Thus, ODBMS products in both the commercial market and the research world seem to be developed from two directions:

1. Using an object-oriented programming language as a basis for adding database management system capabilities, and

2. Extending conventional database management systems based on established data models (such as the relational) to support the functions of object systems.

7.1 Extending OOPL to Include DBMS Functions

One way of developing the ODBMS is to add DBMS services to an object-oriented programming language. This has been the approach taken in the development of many of the commercial ODBMS available today. These database services include:

- o Persistent storage of objects and archiving facilities.

- o Transaction management including sharing, concurrency control and recovery.

- o Support for query languages and access paths.

Other features can be supported in an ODBMS such as versioning and cooperative transactions.

7.2 Extending Conventional or Relational DBMS

Efforts at extending database management systems to support nontraditional database applications are also under way. Motivated by the requirements of design applications, these efforts have focused on augmenting conventional database management systems with the ability to define and manipulate data types not supported by traditional systems.

Some of these new capabilities are considered to reflect the influence of object concepts. Yet, as a whole, these systems may be distinguished from ODBMS.

<u>Extensible database management systems</u> are database management systems intended to provide database support for computer applications that are difficult to implement using conventional database management systems [ZDON90]. The goal of extensible database management systems is to provide the

application developer with both the ability to design database systems based on a richer and more expressive data model and the data management services needed for application domains which cannot easily make use of conventional DBMS.

Much of the research in extensible database management systems has focused on developing semantic constructs, that, although not directly supported in relational systems, might be added relatively easily. Extensible database management systems include features such as:

- **Extended Data Typing which Enhances Modeling Capability in Domains such as Design and Artificial Intelligence.**

  This includes support for data typing facilities that allow an extended set of system defined data types (including types such as text, graphics, voice, and image), abstract data types, and incorporation of procedure definitions in declarations of data types. It may also be possible to define generic data types from which more specific, application level, data types can be created dynamically.

- **Definition and Manipulation of Type Hierarchies and Object Hierarchies.**

  These capabilities are similar to those provided in the object model. This includes the ability to specify type hierarchies based on inheritance, to define composite or aggregate data types, as well as to specify shared object hierarchies.

- **Support Extensions to Query Languages**

  The ability to define extended data types must be accompanied by a query language that supports additional operations for selection and retrieval of information.

- **Provide Facilities for Active Databases and Inferencing**

  The term <u>active database</u> refers to database systems where retrieval and update operations can automatically cause invocation of procedures. Procedures known as alerters and triggers can be associated with fields. When the field is accessed, the procedure is invoked. More extensive inferencing capabilities can be provided, including production rules, to permit implementation of knowledge-based systems applications.

o **Support for Versioning and Long Duration Transactions**

Extensible database systems in design environments may also require support for version controls and long duration, cooperative transactions.

7.3 Review of ODBMS

The approach of adding DBMS capability to an object-oriented programming language has been taken by some private companies. This approach has resulted in many of the commercial object database management systems available today. Examples are the Statice system from Symbolics [WEIN88] and G-base from Graphael (distributed by Object Database Incorporated) [GRAP87]. Both systems are Lisp-based ODBMS. Further, GemStone from Servio Logic Development Corporation [COPE84], [MAIE86a], [MAIE86b], [MAIE87] offers a Smalltalk-based language, OPAL, for data definition, data manipulation, and general computation. Recently, ONTOS from Ontologic Incorporated [ONTO89] uses a C++ language binding to interface with an Object Server.

The approach of extending the conventional DBMS has been evident in research efforts to develop experimental systems, particularly in the case of extensible database systems. Some of these systems, while they do not incorporate all aspects of the object data model, as presented in section 5, do support many of the functions associated with object systems. Examples are the POSTGRES project at the University of California at Berkeley [STON86a], [STON86b], [STON87a], [STON87b], [STON88], [ROWE86], [ROWE87] which uses INGRES as the underlying DBMS. Iris at Hewlett-Packard [LYNG86], [FISH87a], [FISH89] uses the ALLBASE relational DBMS as its storage manager. Other extensible database systems are Starburst by IBM Almaden Research Center [SCHW86], [LIND87], the EXODUS project at University of Wisconsin [CARE86], [GRAE87], [RICH87], and the GENESIS project at the University of Texas-Austin [BATO88a], and [BATO88b].

There are also some research prototype ODBMS which do not seem to be developed along either of these two approaches. These are the CACTIS project at the University of Colorado [HUDS87], the ROSE project at the Rensselaer Polytechnic Institute [HARD87], [HARD88a], [HARD88b], [HARD88c], and the Zeitgeist project at Texas Instruments [FORD88].

The above list of systems is not meant to be exhaustive. There are at least a dozen industrial research projects around the world developing prototype ODBMS, and recently several startup companies are now in various stages of readiness to offer their versions of ODBMS.

7.4 Standardization

Despite the emergence of ODBMS prototypes and products, there is currently no identified standard for the object data model and ODBMS. Various groups are now attempting to specify standards for different components of the object technology. These groups include the Object Management Group, Open Software Foundation, and the ANSI Object-Oriented Database Task Group (OODBTG), operating under the X3/SPARC Database System Study Group. The OODBTG is currently working on a reference model for ODBMS to identify potential areas for standardization [OODB90].

Standards are under development for the Common Lisp Object System (CLOS) and the C++ language.

In the area of database standards, relational SQL has now firmly taken root [ANSI89]. Thus any standards for the ODBMS must address new application areas or must explore the relationship of SQL to the object database language.

8.0 ISSUES AND CONCLUSIONS

The state-of-the-art in object database management systems is presented in this report. The review of ODBMS technology is based upon published literature and actual experimentation with selected ODBMS products.

Although some commercial ODBMS have appeared, this technology is still in the process of rapid development and is likely to remain so for at least the remainder of this decade. Many of the ideas have been discussed from the point of view of programming languages. Designers of DBMS are trying to incorporate these object-oriented notions into a framework that has been established for databases. Hence, many issues and questions are continuing to be topics of intensive research in the database community.

8.1 Issues

Most commercial ODBMS provide an ad hoc query language to permit users to specify conditional queries, such as retrieval of all objects belonging to a certain class that have a certain value or range of values for an instance variable. However, the semantics of the object data model are not necessarily expressible in query language provided by relational DBMS.

- **The Need for an Object Query Language**

  The query language for an ODBMS may have to express query conditions involving inheritance relationships. For instance, queries may specify retrieval of objects belonging to a class, or to a class and some or all of its subclasses. Similarly, it may be necessary to state query conditions which specify retrieval of a composite object together with all or some of its parts. A relational query language could not easily be used to state these queries, and in fact, may not be able to state them at all. This suggests the need for development of an object algebra, to support an object query language.

- **Object Structured Query Language (OSQL)**

  Structured Query Language (SQL) is an ANSI and ISO standard developed for relational systems [ANSI89]. OSQL has been discussed in the last 2 years within the SQL standards group. Preliminary enhancements for SQL to support object concepts might include facilities such as: user-defined data types; generalization hierarchy for tables with the ability for a subtable to inherit the columns, key attributes, and integrity constraints from a more general table definition; and features that support encapsulation with the use of assertions, triggers, and external procedure calls.

o **Query Optimization and Object Encapsulation**

In principle, in an ODBMS, encapsulation precludes knowledge of, and direct access to, an object's instance variables. On the other hand, in a relational database management system, query processing generally assumes far fewer restrictions on access to fields in a record. For instance, specification of access methods for query optimization requires access to, and knowledge of, information about type, length, and range of values for the attributes of a relation. In an ODBMS, using the message passing mechanism to retrieve large numbers of objects may impose additional overhead that degrades performance. Retrieving objects by means other than message passing, such as through use of inverted indexes, might improve performance. Differences in response times will likely be even greater in the case of very large object databases. However, accessing objects without using message passing protocols seems to violate the principle of encapsulation, allowing an object's instance variables to be directly accessed. This issue may have implications not only on ODBMS architecture but also on the development of the Object Data Model.

o **Kinds of Indexes for Objects**

Indexing objects (discussed in sec. 6.3) raises questions about what kinds of indexes might be useful. For instance, it may be desirable to index objects not only on the basis object ID and instance variable value, but also to create indexes on values returned by methods that have been invoked in response to a message. The indexes may be for objects of one class or several different classes.

o **Object Boundary**

An object has structural relationships with other objects (see sec. 5.1). For instance, CARGO_AIRCRAFT may have a relationship with objects representing its CARGO_ITEMs. This relationship may be implemented by storing the Object IDs of the related CARGO_ITEMs in an instance variable of the CARGO_AIRCRAFT object. The issue of object boundary concerns an object's scope. Does the object consist of only its own immediate internal state, e.g., the values of its own instance variables? Or, does the object also include the internal state of the objects it has relationships with? That is, does retrieving the CARGO_AIRCRAFT object mean that the objects representing its cargo should also be retrieved? Determining the extent or scope of the CARGO_AIRCRAFT object is part of the object boundary issue. The answer may depend on the semantics of a particular application. This question has important implications for performance, access methods, and concurrency control e.g., determination of locking granularity. Object boundary is also a possible issue in implementing composite objects and database security.

- **Integration of Programming Language and Database Languages (achieving seamlessness)**

The OOPL and the DBMS are generally not integrated, and may be considered separate components. An ODBMS may also have to interface with application development tools that use different OOPLs, such as Smalltalk or C++. An important issue is ensuring that the ODBMS and the different OOPLs share a common data model, and that they also share a single database language for defining the database schema and for performing data manipulation operations. There should also be a common query language which can be used both in application programs and by external users.

- **Encapsulation and Database Security**

The property of object encapsulation and its relationship to security is an important issue. The question is whether encapsulation is related to specification of database, or object views?

The above issues will remain as active research topics for several more years.

8.2 Conclusions

The major focus of this report is to describe object concepts and identify features for ODBMS.

The research and development of ODBMS is far from complete. No commercial ODBMS or prototype system, at present, supports all of the features identified in this report.

The projected future for object database systems is that ODBMS will first be widely used in system development environments with data-intensive applications. These environments could include CAD/CAM, hypermedia, engineering and expert systems, etc. The relational databases and the SQL standard will still likely be used in conventional types of applications.

It is possible that a new collection of federated heterogeneous databases will emerge to tie together object-oriented, relational, and high-throughput transaction-oriented systems. However, it will likely take another 5 to 10 years before such technology is commercially accepted.

The object paradigm will be spreading not only to database systems, but to all software engineering activities including user interfaces, requirements specification, logical design, physical design, and testing. Exciting research in this technology will challenge computer scientists for the next decade.

9.0 REFERENCES AND RELATED READINGS

[ANDR87] Andrews, T., and Harris, C. "Combining Language and Database Advances in an Object-Oriented Development Environment," Proceedings OOPSLA 1987.

[ANDR88] Andrews, T. Using an Object Database to Build Integrated Design Environments, Ontologic Inc. Technical Report, 1988.

[ANSI89] ANSI "American National Standard - Database Language SQL with Integrity Enhancement," Number X3.135-1989, American National Standards Institute, 1989.

[BANC89] Bancilhon, F. et al. "The Object-Oriented Database Manifesto," Draft paper for the Conference on Deductive and Object-Oriented Databases in Kyoto, Japan, December 1989.

[BANE87] Banerjee, J. et al. "Semantics and Implementation of Schema Evolution in Object-Oriented Databases," SIGMOD Record, Vol 17, No. 3, 1987, pp 311-322.

[BATO88a] Batory, D. "Concepts for a Database System Compiler," Proceedings of the 1988 ACM Principles of Database Systems Conference, Austin, TX, March 1988.

[BATO88b] Batory, D. "GENESIS: An Extensible Database Management System," IEEE Transactions on Software Engineering, 14,11, November 1988.

[BJOR89] Bjornerstedt, A., and Hulten, C. "Version Control in an Object-Oriented Architecture," in Kim, W., and Lochovsky, F. (eds.) Object-Oriented Concepts, Databases, and Applications, ACM Press, Addison-Wesley Publishing Company, 1989.

[BLAK87] Blake, E., and Cook, S. "On Including Part Hierarchies in Object-Oriented Languages with an Implementation in Smalltalk," in ECOOP '87 Conference Proceedings, June 1987.

[BOBR88] Bobrow, D. G. et al. Common Lisp Object System Specification, X3J13 Technical Report 88-002R, June 1988.

[CANN82] Cannon, H. "Flavors, A Non-hierarchical Approach to Object-Oriented Programming," Draft Document 1982.

[CARD90] Cardelli, L. "A Semantics of Multiple Inheritance," in Readings In Object-Oriented Database Systems, Zdonik, S., and Maier, D. (eds.) Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1990.

[CARE86] Carey, M. et al. "The Architecture of the EXODUS Extensible DBMS," Proceedings of ODBS, 1986.

[CHEN76] Chen, P.P.S. "The Entity-Relationship Model - Toward a Unified View of Data," ACM Transactions on Database Systems, 1:1, March 1976, 9-36.

[CODD81] Codd, E.F. "Data Models in Database Management," SIGMOD Record, Vol. 11, No. 2, Feb 1981.

[COPE84] Copeland, G., and Maier, D. "Making Smalltalk a Database System," SIGMOD Record 1984, pp 316-326.

[DAHL66] Dahl, O., and Nygaard, K. "Simula - An Algol-Based Simulation Language," Communications of the ACM, 9, 9, 1966, 671-678.

[DAYA87] Dayal, U., and Smith, J. "PROBE: A Knowledge-Oriented Database Management Systems," in Brodie, M., and Mylopoulos, J. (eds.) On Knowledge Base Management Systems, Springer-Verlag, 1987.

[DAYA90] Dayal, U. et al. "Simplifying Complex Objects: The PROBE Approach to Modelling and Querying Them," in Readings In Object-Oriented Database Systems, Zdonik, S., and Maier, D. (eds.) Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1990.

[FISH87a] Fishman, D. et al. "Iris: An Object-Oriented Database Management System," ACM Transaction on Office Information Systems, Vol. 5, No. 1, 1987 pp 48-69.

[FISH89] Fishman, D. et al. "Overview of the Iris DBMS," in Object-Oriented Concepts, Databases, and Applications, edited Kim, W., and Lochovsky, F. ACM Press, New York, 1989, pp. 219-250.

[FORD88] Ford, S. et al. Zeitgeist: Database Support for Object-oriented Programming, Lecture Note 334, 1988.

[GARZ88] Garza, J., and Kim, W. "Transaction Management in an Object-Oriented Database Systems," SIGMOD Records, Vol 17, No. 3 September 1988.

[GOLD83] Goldberg, A., and Robson, D. Smalltalk-80, The Language and Its Implementation, Addison-Wesley, Reading, MA, 1983.

[GRAE87] Graefe, G., and DeWitt, D. "The EXODUS Optimizer Generator," SIGMOD Record, 1987, pp 160-172.

[GRAP87] Graphael, "G-Base Tools," The G-Base Manual written by Emma Callaghan, 1987.

[HAMM81] Hammer, M., and McLeod, D. "Database Description with SDM: A Semantic Database Model," ACM Transactions on Database Systems, 6,3 (1981), 351-386.

[HARD87] Hardwick, M. "Why ROSE is Fast: Five Optimizations in the Design of the Experimental Database System for CAD/CAM Applications," SIGMOD Record, 1987.

[HARD88a] Hardwick, M., and Spooner, D. The ROSE Data Manager: Using Object Technology to Support Interactive Engineering Applications, RPI Report 1988.

[HARD88b] Hardwick, M., and Spooner, D. "ROSE: An Object-Oriented Database System for Interactive Computer Graphics Applications," Lecture Notes: Advances in Object-Oriented Database Systems, 334, Springer-Verlag, 1988.

[HARD88c] Hardwick, M. User Manual for ROSE, a Database System for Engineering Design, Interactive Computer Graphics and Other Object-Oriented Applications, RPI Report 1988.

[HORO83] Horowitz, E. Fundamentals Of Programming Languages., Computer Science Press, Rockville, MD, 1983.

[HUDS87] Hudson, S.E., and King, R. "Object-Oriented Database Support for Software Environments," SIGMOD Record, Vol. 16, No. 3, December 1987.

[ISO89] ISO "Information Processing Systems - Open Systems Interconnection - Remote Database Access - SQL Specialization," ISO Working Draft, Document ISO/JTC1/SC21/WG3 N844, June 2, 1989.

[JOSE89] Joseph, J.V. et al. "Object-Oriented Databases: Design and Implementation," Draft Report from Texas Instruments, 1989.

[KIMW87a] Kim, W. et al. "Composite Object Support in an Object-Oriented Database System," in <u>OOPSLA '87 Proceedings</u>, 1987.

[KIMW88a] Kim, W. et al. "Integrating an Object-Oriented Programming System with a Database System," in <u>OOPSLA '88 Proceedings</u>, 1988, pp 142-152.

[KIMW88b] Kim, W., and Chou, H. "Versions of Schema for Object-Oriented Databases," <u>Proceedings 14th Very Large Databases</u>, August 1988.

[KIMW89] Kim, W. et al. "Features of the ORION Object-Oriented Database System," in <u>Object-Oriented Concepts, Databases, and Applications</u>, edited Kim, W., and Lochovsky, F. ACM Press, New York, 1989, pp. 250-282.

[LECL88] Lecluse, C., Richard, P., and Velez, F. "O2, an Object-Oriented Data Model," <u>SIGMOD 88 Conference Proceeding</u>, pp 424-433.

[LIND87] Lindsay, B., McPherson, J., and Pirahesh, H. "A Data Management Extension Architecture," <u>Proceedings of the 1987 SIGMOD Conference</u>, San Francisco, CA, May 1987.

[LISK77] Liskov, B., Snyder, A., Atkinson, R., and Schaffert, C. "Abstraction Mechanisms in CLU," <u>Communications of the ACM</u>, (20, 8), 1977, 564-576.

[LYNG86] Lyngbaek, P., and Kent, W. "A Data Modeling Methodology for the Design and Implementation of Information Systems," <u>Proceedings of ODBS</u>, 1986, pp 6-17.

[MAIE86a] Maier, D., Otis, A., and Purdy, A. "Object-Oriented Database Development at Servio Logic," <u>Database Engineering</u>, 18:4, December 1986.

[MAIE86b] Maier, D. et al. "Development of an Object-Oriented DBMS," <u>Proceedings of the OOPSLA</u>, Sept 1986.

[MAIE86c] Maier, D., and Stein, J. "Indexing in an Object-Oriented DBMS," <u>Proceedings of the ODBS</u>, 1986.

[MAIE87] Maier, D., and Stein, J. "Development and Implementation of an Object-oriented DBMS," in Shriver, B., and Wagner, P. (eds.) Research Directions in Object-Oriented Programming, MIT Press, Cambridge MA 1987, pp 355-392.

[MANO86] Manola, F.A., and Dayal, U. "PDM: An Object-Oriented Data Model," International Workshop on ODBS, September 1986,

[MARI88] Mariategui, F., Eich, M., and Rafiqi, S. "The Object-Oriented Data Model Defined," Southern Methodist University Technical Report 88-CSE-28, April 1988.

[OBRI86] O'Brien, P. et al. "Persistent and Shared Objects in Trellis/Owl," Proceeding International Workshop on ODBS, September 1986.

[OBRI87] O'Brien, P. et al. "The Trellis Programming Environment," in OOPSLA Proceedings, 1987.

[OODB90] Object-Oriented Database Task Group, A Reference Model for an Object Database, Draft Report to X3/SPARC Database System Study Group (DBSSG), Available from Elizabeth Fong, NIST, Technology Bldg. A 266, Gaithersburg MD 20899, 1990.

[ONTO89] Ontologic Inc. "ONTOS Technical Overview," Viewgraphs used by T. Andrews, Ontologic Inc. Burlington, MA 1989.

[PENN87] Penney, D., and Stein, J. "Class Modification in the GemStone Object-Oriented DBMS," Proceedings of the OOPSLA Conference, 1987, pp 111-117.

[RICH87] Richardson, J.E., and Carey, M. "Programming Constructs for Database System Implementation in EXODUS," SIGMOD Record, Vol. 16, No. 3, Dec. 1987.

[ROWE86] Rowe, L. "A Shared Object Hierarchy," Proceeding International Workshop on OODBMS, September 1986.

[ROWE87] Rowe, L., and Stonebraker, M. "The POSTGRES Data Model," Proceeding 13th Very Large Databases Conference, September 1987.

[SCHW86] Schwarz, P. et al. "Extensibility in the Starburst Database System," Proceedings of the International Workshop on Object-Oriented Database Systems, Pacific Grove, CA, September 1986.

[SHIP81] Shipman, D.W. "The Functional Data Model and the Data Language DAPLEX," ACM Transactions on Database Systems, 6(1), March 1981.

[SKAR89] Skarra, A., and Zdonik, S. B. "Concurrency Control and Object-Oriented Databases," in Kim, W., and Lochovsky, F. (eds.) Object-Oriented Concepts, Databases, and Applications, ACM Press, Addison-Wesley Publishing Company, 1989.

[SMIT77] Smith, J.M., and Smith, D. "Database Abstractions: Aggregation and Generalization," ACM Transactions on Database Systems, 2(2) June 1977.

[STEF83] Stefik, M. et al. "Knowledge Programming in LOOPS: Report on an Experimental Course," AI Magazine, Vol. 4, No. 3, Fall 1983.

[STON86a] Stonebraker, M., and Rowe, L. "The Design of POSTGRES," Proceeding ACM SIGMOD Conference, June 1986.

[STON86b] Stonebraker, M. "Object Management in POSTGRES Using Procedures," Proceeding ODBS, 1986.

[STON87a] Stonebraker, M. et al. A Rule Manager for Relational Database Systems, Memo No. UCB/ERL M86/85, June 1987.

[STON87b] Stonebraker, M. The Design of the POSTGRES Storage System, Memo. No. UCB/ERL M86/85, June 1987.

[STON88] Stonebraker, M. et al. "The POSTGRES Rule Manager," IEEE Transaction on Software Engineering, Vol 14, No. 7, July 1988.

[STRO86] Stroustrup, B. The C++ Programming Language, Addison-Wesley, Reading, MA, 1986.

[UNGA87] Ungar, D., and Smith, R.B. "SELF: The Power of Simplicity," OOPSLA '87 Proceedings, Orlando, FL, 1987, 227-241.

[WEGN87] Wegner, P. "Dimensions of Object-Based Language Design," OOPSLA '87 Proceedings, Oct. 1987, pp 168-181.

[WEIN88] Weinreb, D. "An Object-Oriented Database System to Support in Integrated Programming Environment," Data Engineering Vol. 11, No. 2, 1988, pp 33-43.

[ZDON90] Zdonik, S. B., and David, M. (eds.) Readings in Object-Oriented Database Systems, Morgan Kaufmann Publishers, Inc. San Mateo, CA 1990.

APPENDIX - GLOSSARY

Some of the terms used in this report are found throughout the literature, occasionally with conflicting meanings. This Appendix contains short, informal definitions of key concepts and terms which are underlined in the report.

Abstract Data Type:

A programming technique that defines a data space, hiding procedures and details the programmer does not need to know to manipulate the data. The definition of an abstract data type consists of an internal representation along with a set of procedures required to access and manipulate the data.

Active Database:

A database system in which retrieval and update operations result in invocation of procedures. Such procedures, known as triggers, are associated with particular fields. When the field is accessed, the trigger is activated.

Attribute:

Attributes are properties of an entity. An entity is said to be described by its attributes. In a database, the attributes of an entity have their analogues in the fields of a record. In an object database, instance variables may be considered attributes of objects.

Class:

A generic description of an object type consisting of instance variables and method definitions. Class definitions are templates from which individual objects can be created.

Class Object:

A class definition. In many OOPL and ODBMS implementations, class definitions are objects that are instances of a generic class, or metaclass.

Class Hierarchy:

Classes can naturally be organized into structures (tree or network) called class hierarchies. In a hierarchy, a class may have zero or more superclasses above it in the hierarchy. A class may have zero or more classes below, referred to as its subclasses.

Class Library:

A set of related classes belonging to a specific domain. For example, a graphics library may exist, consisting of classes of graphical objects.

Code Reuse:

The ability to use a single piece of code for more than one purpose in a computer application. When a superclass definition is inherited by a subclass, the code associated with the superclass definition, including method definitions, is reused in the subclass. Code reuse has the effect of reducing the amount of code needed to implement an application.

Composite or Complex Object:

An object which is made up of other objects. Composite objects consist of collections of parts, each of which is itself an object. Each part is in an "Is-Part-Of" relationship with the object of which it is a component.

Concurrency Control:

A mechanism that regulates access to objects and prevents users from executing inconsistent actions on the database.

Data Abstraction:

A programming technique by which the internal representation and operations of an object are made only partially visible, allowing only certain information relevant to a particular application to be seen. The actual methods by which computations are performed remain hidden from external view. See also encapsulation.

Data Model:

The data model is a specification of the structure of the database, the operations, and the integrity rules.

Database Schema:

The complete set of individual schema definitions which describe the logical structure of a database. In an ODB, the database schema is expressed in the set of class definitions for a database.

Dynamic Binding:

Also known as run time binding or late binding. Dynamic binding refers to the association of a message with a method during run time, as opposed to compile time. Dynamic binding means that a message can be sent to an object without prior knowledge of the object's class.

Encapsulation:

The packaging of data and procedures into a single programmatic structure. In object-oriented programming languages, encapsulation means that an object's data structures are hidden from outside sources and are accessible only through the object's protocol.

Entity:

A collection of information items which can conceptually be grouped together and distinguished from their surroundings. An entity is described by its attributes. Entities can be linked, or have relationships to other entities.

Extensible Database Management Systems:

A class of DBMS incorporating additional data modeling capabilities together with data management services needed for application domains which cannot easily make use of conventional DBMS.

Extensibility:

The ability to dynamically augment the database schema. This includes addition of new data types and class definitions for representation and manipulation of unconventional data such as voice data, image data, and data associated with artificial intelligence applications.

Generalization:

Refers to the relationship between a superclass and its subclasses. A superclass is a generalization of its subclasses.

Handle:

A pointer to, or address of, an object. A handle is a unique, and nonchangeable reference to an object. In some systems, the term handle is interchangeable with the term object identity.

Inheritance:

A mechanism which allows objects of a class to acquire part of their definition from another class (called a super-class). Inheritance can be regarded as a method for sharing a behavioral description.

Instance:

An individual occurrence of an object.

Instance Variable:

An attribute of an object. A class definition may specify the set of instance variables that constitute the data structures for objects of the class.

Message:

See message passing.

Message Passing:

The means by which objects communicate. Individual messages may consist of the name of the message, the name of the target object to which its being sent, and arguments, if any. When an object receives a message, a method is invoked which performs an operation that exhibits some part of the object's behavior.

Method:

A method is the body of code executed in response to a message. The methods associated with a class definition effectively describe the behavior of all the instances of the class.

Multiple Inheritance:

The ability for a class to inherit from more than one superclass. Thus, a class may inherit instance variables and methods from multiple superclasses.

Object:

An object is the basic unit of computation. An object has a set of "operations" and a "state" that remembers the effect of operations. Classes define object types. Typically, objects are defined to represent the behavioral and structural aspects of real world entities.

Object ID (object identity):

A permanent unique identifier that is assigned to each object. The identifier is independent of the value of the instance variables of the object, and remains constant despite any change in the object's state. Object Identity is sometimes used interchangeably with the term handle.

Object Server:

An Object Server is the software system which supports transaction management and storage management functions for objects.

Part Hierarchy:

A hierarchy of component objects which form parts of a composite object. A composite object will be made up of objects which may themselves have components. This is distinguished from a class hierarchy which consists of classes related through inheritance.

Persistence:

A property of data or objects implying that it has a lifetime greater than the process which created it.

Persistent Object Store:

The object database, or ODB.

Polymorphism:

Polymorphism refers to being able to apply a generic operation to data of different types. For each type, a different piece of code is defined to execute the operation. In the context of object systems, polymorphism means that an object's response to a message is determined by the class it belongs to.

Protocol:

The set of messages an object will respond to. The term protocol can sometimes be used interchangeably with the term public interface. In an ODBMS, protocols are specified in class definitions.

Referential Integrity:

In a relational database, referential integrity means that no record may contain a reference to the primary key of a nonexisting record.

Run Time Binding:

See Dynamic Binding.

Shadowing:

The definition of a method in a class description to replace a method that would otherwise be inherited from a superclass. When a message is sent to an object that is an instance of the subclass, the method defined in the subclass is invoked. The shadowed method in the superclass is not invoked.

Specialization:

Refers to the relationship between a subclass and its superclasses. A subclass is a specialization of its superclasses.

State:

The set of values for the instance variables of an object. When the values of any of the object's instance variables change, the object's state is altered.

Subclass:

When a class inherits the instance variables and methods from another class, it is referred to as its subclass.

Superclass:

The class from which the instance variables and methods of a subclass are inherited.

NIST-114A
(REV. 3-89)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

BIBLIOGRAPHIC DATA SHEET

1. PUBLICATION OR REPORT NUMBER
NIST/SP-500/179

2. PERFORMING ORGANIZATION REPORT NUMBER

3. PUBLICATION DATE
April 1990

4. TITLE AND SUBTITLE
Object Database Management Systems: Concepts and Features

5. AUTHOR(S)
Christopher E. Dabrowski, Elizabeth N. Fong, and Deyuan Yang

6. PERFORMING ORGANIZATION (IF JOINT OR OTHER THAN NIST, SEE INSTRUCTIONS)
U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
GAITHERSBURG, MD 20899

7. CONTRACT/GRANT NUMBER

8. TYPE OF REPORT AND PERIOD COVERED
Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (STREET, CITY, STATE, ZIP)
Same as item #6

10. SUPPLEMENTARY NOTES

[ ] DOCUMENT DESCRIBES A COMPUTER PROGRAM; SF-185, FIPS SOFTWARE SUMMARY, IS ATTACHED.

11. ABSTRACT (A 200-WORD OR LESS FACTUAL SUMMARY OF MOST SIGNIFICANT INFORMATION. IF DOCUMENT INCLUDES A SIGNIFICANT BIBLIOGRAPHY OR LITERATURE SURVEY, MENTION IT HERE.)

The last decade has seen the emergence of object concepts and their infusion into information systems technology. This phenomenon began with the advent of programming languages that included object concepts. More recently, object concepts have been merged with database management system technology, resulting in the production of some object database management systems. As a result, the term object database management system (ODBMS) is now becoming a recognized and important topic in the database community. The purpose of this report is to provide managers and software analysts a state-of-the-art review of object concepts and to describe features associated with object database management systems.

12. KEY WORDS (6 TO 12 ENTRIES; ALPHABETICAL ORDER; CAPITALIZE ONLY PROPER NAMES; AND SEPARATE KEY WORDS BY SEMICOLONS)
class; database; database management system; object; object-oriented; object database management system; ODBMS.

13. AVAILABILITY
[X] UNLIMITED
[ ] FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NATIONAL TECHNICAL INFORMATION SERVICE (NTIS).
[X] ORDER FROM SUPERINTENDENT OF DOCUMENTS, U.S. GOVERNMENT PRINTING OFFICE, WASHINGTON, DC 20402.
[ ] ORDER FROM NATIONAL TECHNICAL INFORMATION SERVICE (NTIS), SPRINGFIELD, VA 22161.

14. NUMBER OF PRINTED PAGES
63

15. PRICE

ELECTRONIC FORM

ANNOUNCEMENT OF NEW PUBLICATIONS ON COMPUTER SYSTEMS TECHNOLOGY

Superintendent of Documents
Government Printing Office
Washington, DC 20402

Dear Sir:

Please add my name to the announcement list of new publications to be issued in the series: National Institute of Standards and Technology Special Publication 500-.

Name ____

Company ____

Address ____

City ____________________ State ______________ Zip Code ____________

(Notification key N-503)