Thai Journal of Development Administration Vol. 30, No. 1, January - March 1990

INTRODUCTION TO OBJECT - ORIENTED DATABASE SYSTEMS

Raweewan Auepanwiriyakul*
Kathleen Swigger**

Introduction

The purpose of this paper is to present an introduction to object-oriented database systems. It describes the differences between object-oriented database systems and more conventional database systems, and provides a rationale for selecting object-oriented database approach over existing database models. Object-oriented database systems are divided into two categories, behavioral and structural object-oriented systems. Object-oriented database systems have a special feature that makes them a better systems. This special feature is an inheritance between a class and its superclass in the class hierarchy.

The first section is an introduction to object-oriented database systems. The second section describes the differences between the behavioral and structural approach in object-oriented design. The last section describes different inheritance problems and possible solutions to these problems. The inheritance problems discussed in this section include name conflicts which occur within multiple inheritance and incomplete inheritance.

Object-Oriented Database Systems

The concept of an object and a class was first introduced into programming language in 1962 with the programming language Simula [NYG86]. Simula referred to classes as named activities and objects as processes. This concept of a class/object data structure later formed the basis of several object-oriented programming languages such as Smalltalk, Flavors, LOOPS, and object-Lisp [WOE86]. Following the emergence of object-oriented programming languages, several proposals were made for using the object-oriented concept within a database environment. Examples of such systems include Gem-Stone [COP84, MAI85, MAI86, PEN87], Iris [DER85, DER86, FIS87, LYN86], ORION [BAN87, CHO87, KIM87], PROBE Data Model (PDM) [MAN86], ENCORE [SMI87, ZDO85], Object Oriented Programming System (OOPS) [SCH88], Cactis [HUD86, HUD87], and Statice [WET88].

An object-oriented system, regardless of whether it is a programming language or a database system, represents conceptual entities as objects. An object can represent anything from a simple number to a series of complex entities. Each object consists of its own private memory and contains the values of its instance variables.

The behavior of an object is encapsulated in procedures called " methods. " Methods are procedures or code that manipulate other objects and return values of instance variables. Methods are part of the object itself and are not visible from outside the object. Objects communicate with each other by sending messages to themselves and other objects. Messages, together with any arguments attached to the messages, present the public interface of an object. For each message, there is a corresponding method that executes the message. Similar

* Instructor, School of Applied Statistics, NIDA
** Associate Professor, Computer Science Department, UNT

objects are grouped into a class. All objects of the same class have the same instance variables and methods. Objects that belong to a class are called instances of that class. A class can have none, one, or more than one superclasses. A class inherits every instance variable and method from its superclass. It can have additional instance variables and methods. This subclass/superclass relationship is represented by a class hierarchy [CHO87]. Some object-oriented database systems support multiple inheritance; that is, a subclass can have multiple superclasses. A class hierarchy is also represented by a lattice. For those systems that do not support multiple inheritance, a class hierarchy is represented as a tree.

The idea of a class/subclass data structure, coupled with inheritance of attributes and values, appears to be one solution to many of the problems that have plagued more traditional database models. Several authors [SCH88, COP84, SMI87, RAM88, HUD87] have attempted to catalog these problems, and a brief review of these criticisms are listed as follows :

1. Most traditional database systems (hierarchical, relational, and network) do not support sophisticated data types that describe the structure of a complex object in both a natural and efficient manner. Traditional database systems supply the database manager with a fixed number of simple data types such as integer, real, and character. Traditional database systems lack the ability to define new, more complicated data types.

2. Traditional database management systems cannot express complex objects. An object is often divided into several database objects. The data structuring capabilities of existing techniques do not adequately support the complexities and variations that occur in real data.

3. Operational semantics of complex objects cannot be expressible. In a traditional database management system, a user cannot include operations of the object in the database; he can only specify them in his application program.

4. The existing database systems do not support a hierarchy of types and inheritance of properties along a hierarchy.

5. Existing database systems force the user to specify constraint checking in the application program, not in the update command. Since every data item can have constraints attached to it, the update command should include constraint checking to preserve integrity.

6. Traditional database systems do not support a data manipulation language that is capable of arbitrary computation on database objects. The user usually must learn two languages; the data manipulation language and a general-purpose language such as C or PASCAL. These two languages may have some different data structures and programming paradigms.

7. Most traditional database systems lack efficient functions for complex object storage and retrieval. Because complex objects are stored as a series of database objects, a number of queries are needed to retrieve a single object from the database. Similarly, a number of queries are needed to store a single object.

Object-oriented database systems can overcome many of the limitations listed above. Because object-oriented database systems allow the user to specify new data types and operations to those data types, the limitations listed above under items 1, 3, 5 and 7 are satisfied. Also, object-oriented database systems allow the user to represent any type of entity. Such systems support class hierarchy, particularly the subclass/superclass relationship. Not only do object-oriented database systems overcome most of the problems associated with traditional systems, they also provide many additional features [MAI86, MAI86a, DER86, FIS87]. Some of these features are summarized as follows:

1. They allow the designer to represent complex design entities in a more direct manner than the traditional database management systems.

2. They enhance efficiency. Because all operations are performed in the database, data does not have to be transferred from the database to the application program. Moreover, a single method corresponds to a number of database queries and updates. Thus, communication between database and application programs are reduced.

3. They increase data dependency. Application programs access data in object-oriented database systems by sending messages. Therefore, unnecessary details are removed from the application program. If the structure of the database changes, the application program does not need to be changed as long as the messages are not changed.

4. They facilitate information hiding. Application programmers do not have to worry about the details of retrieving data, finding their data, and checking constraints, because these items are coded into the operations in the database management system.

5. By storing an operation in the database, a system administrator can ensure that there exists only one copy of that operation. Every application program can be forced to use the same set of operations. Thus, application programs that send the same message with similar arguments will get the same result.

6. They insure integrity of the data. By restricting database access to sending messages to specify objects, a database administrator can reduce the risk of intentional or unintentional corruption or disclosure of data by an application program. The system can also check the consistency of the data in the same operation.

7. They allow the designer to specify object identity to distinguish the object from all other objects. In the relational data model, the properties of an entity must be sufficient to distinguish it from other entities. However, properties of an entity may not be sufficient to identify it in the real world.

As a result of the advantages listed above, many authors have proposed database systems that use object-oriented concepts in their data model. These systems are intended to meet the needs of several new database applications such as office information system, knowledge-based systems, CAD/CAM applications, geographic information systems, and hardware and software design systems [FIS87, MAN86, CHO87].

Behavioral and Structural Object-Orientations

Object-oriented database systems can be further divided into two categories, structural and behavioral object-orientations [DIT86]. The behavioral object-oriented database systems are based on either the concept of object-oriented programming language or the functional data model. The behavioral approach defines objects according to their features and a set of operations (methods) that act upon these objects. Instances of the object type can be accessed by sending a message to the object. One message corresponds to one method. The internal structure of the object is known only to the methods. The interface between objects is the set of messages that manipulate that object. Examples of behavioral object-oriented database systems include GemStone [MAI86, PEN87, COP84, MAI85], Iris [DER85, LYN86, DER86, FIS87], PDM [MAN86], ORION [CHO87, KIM87, BAN87], ENCORE [SMI87, ZDO85], and OOPS [SCH88].

One group of behavioral object-oriented databases uses a functional data model to describe its application [DER85, LYN86, DER86, FIS87, MAN86]. For example, in the DAPLEX system [SHI81] an instance variable is a function that maps a class or an object

to a value. Iris [DER85, DER86, LYN86, FIS87] and PDM [MAN86] use a similar model, but allow the user to define methods using only a set of predefined functions and instance variables. Methods in the Iris and PDM systems do not belong to a particular class or object, but rather take one or more classes (or objects) as their arguments.

Other behavioral object-oriented databases borrow concepts from the object-oriented programming paradigm, particularly in their use of abstract data types coupled with the idea of " persistent " objects [MAI86, PEN87, COP84, MAI85, CHO87, KIM87, BAN87, SMI87, ZDO85, SCH88]. A system called ORION [CHO87, KIM87, BAN87] also includes the concept of shared-value and default-valued instance variables. A shared-value instance variable is, as the name implies, a variable that takes on the same value as another object. This concept is similar to the idea of a class variable as it is defined in object-oriented programming languages. The default-valued instance variable allows the user to specify a default value for any unassigned variable.

Behavioral object-oriented databases usually specify methods in two ways. One approach is to develop a special purpose database language that will allow the user to send messages to objects. These languages are usually compiled and stored in the database itself. The internal form is later interpreted in the same way that compiled queries are stored and executed in a relational database. One of the main advantages of this type of approach is that the language can be tailored to the database management system. Unfortunately, one of the disadvantages of using this approach is that the user will have to learn two languages; a database language that is used to query the database and a general-purpose programming language that is used to write the application programs. Moreover, the database manager must also write a compiler or interpreter for the database language. Behavioral object-oriented systems using this approach are GemStone [MAI86, PEN87, COP84, MAI85], PDM [MAN86], Iris [DER85, LYN86, DER86] and OOPS [SCH88].

The second way to specify methods is through the use of an existing programming language. Advantages to using this approach are that the user needs to learn only one language, and the database manager does not have to write a separate compiler. A behavioral object-oriented system that uses this approach is ORION [CHO87, KIM87, BAN87].

Whenever an method is defined in an object-oriented database, there is a question of who " owns " the method. Behavioral object-oriented systems generally use one of two approaches to assign ownership of methods. Ownership can be associated with a class (type) with the assumption that each object belongs to a specific class. This approach can also be thought of as a form of data abstraction and has been used by object-oriented systems such as ORION, GemStone, OOPS, and ENCORE.

The second approach for determining ownership of methods assumes that the methods belong to no one. Objects and classes are merely tokens that are passed like parameters to methods. Methods can act upon multiple objects or classes. Behavioral object-oriented database systems using this approach are Iris and PDM.

The structural object-oriented database systems are database systems that use a data model that allows the user to define data structures to represent entities of any complexity. The data model often includes generic operators to deal with complex objects. These operators are provided by the system and do not belong to any particular class of objects. Every object in the database can use these predefined operators. This concept is in contrast to the behavioral object-orientation approach in that the behavioral approach allows users to specify the class along with the operators (methods) that manipulate the objects of that class. Some examples of the structural object-oriented database systems include Postgres [STO86, ROW86,

KEM87], Statice [WEI88], Object-Oriented Data Model (OODM) [ZHA88], and Cactis [HUD86, HUD87].

Postgres [STO86, ROW86, KEM87] is a structural object-oriented database system built on top of the relational database system called INGRES. Postgres supports complex objects by using an extensible type system for defining new columns for relationship, new operations on these columns, and new access methods.

Statice [WEI88] is a structural object-oriented database system based on DAPLEX. Under this system, whenever a user creates a class and specifies its attributes and their types, the system automatically creates an accessor function for each attribute. An accessor function takes an object as its argument and returns the value of an attribute of that object. Statice also provides a set of system functions that can create and update every object in every class.

The Entity-Relationship model (E - R model) of Chen [CHE76], the Relational Model/Tasmania (RM/T) of Codd [COD79], and the Semantic Data Model (SDM) of McLeod and Hammer [HAM78] have directed the development of a system called Object-Oriented Data Model (OODM) [ZHA88]. This system consists of two parts, a conceptual schema and data operations. The conceptual schema represents classes, objects and relationships between these classes. Data operations, which are provided by the system, include functions for defining schema, creating the database, and manipulating objects.

Cactis [HUD86, HUD87] is based on a principle called active semantics and can support complex functionally-defined data. Under Cactis, attributes of an object can be intrinsic or derived. Derived attributes have an attribution rule attached to them, while intrinsic attributes do not. These rules allow attributes to be derived from other attributes within a given instance and from the values contained in related instances. Cactis provides a number of data mainpulation primitives such as creating and deleting class objects, retrieving and replacing attribute values, etc. Each object in Cactis is designed to respond to a standard predefined set of messages. After a user creates classes, Cactis provides the actual methods which respond to these messages for each class in the system. All objects respond to the same set of messages, but objects of different classes will respond to the same message in different ways.

There are several approaches that have been used to develop a structural object-oriented database systems. These approaches are based on the relational database model, the functional data model, and active semantics.

Inheritance

The idea that classes should inherit variables and values from higher-ordered (super) classes was first introduced by the language Simula [MEY86]. From its first inception, inheritance has been widely used in other object-oriented programming languages and database systems. The basic idea is that a new class may be defined as an extension of a previously defined class.

In most object-oriented database systems, objects are grouped into classes by similarity. Classes are then arranged in a class hierarchy. A class hierarchy is a hierarchy of classes in which an edge between a pair of nodes represents the IS-A relationship; that is, the lower-level node is a specialization of the higher-level node. For each pair of classes in a class hierarchy, the higher-level class is called a superclass of the lower-level class, and the lower-level class is called a subclass of the higher-level class. The instance variables and methods that are specified for a class are inherited by all its subclasses. Additional properties

may be specified for each of the subclasses. If a class has more than one superclass, then it is said to have multiple inheritance.

There have been several proposals and implementations of the inheritance property in different object-oriented programming languages. Versions of multiple inheritance were described in [BAN87, SNY86, MIN87], and incomplete inheritance in [SNY86].

One of the major research issues relating to the question of inheritance concerns multiple inheritance and the problem of name conflicts. Since a class may inherit instance variables from two or more superclasses, inherited instance variables may contain name conflicts between the superclass and subclasses. In most systems, name conflicts between a class and its superclasses are resolved by giving precedence to the definition within the class over that in the superclasses. Name conflicts among superclasses usually are resolved by giving priority to the first superclass in the list of superclasses of a given class. This solution is used by ORION [BAN87]. An alternative conflict resolution strategy is found in Extended Smalltalk [SNY86] which gives an error message whenever the system notices a name conflict. A third solution is to first flatten the inheritance graph to a linear chain, eliminate duplicates, and then treat the result as a single inheritance. The systems that use this strategy are Flavors and CommonLoops [SNY86]. Name conflicts between a class and its superclass can also occur in single inheritance systems and are resolved by using the same techniques described above.

The problem of incomplete inheritance is examined by Snyder [SNY86], who suggests that one solution to the problem is to exclude inherited methods from the subclass's own external interface. This technique might be both reasonable and useful if inheritance is viewed as an implementation technique as in object-oriented programming languages. According to Snyder, a class can inherit some or all methods from another class (superclass). Snyder also suggests that the way to implement stack and deque is to define a class called stack which inherits from the class called deque, but excludes the extra methods that are not applicable for stack. Stack is not a specialization of deque because it does not have all the deque methods; it just has some methods that are already defined for deque. The system that provides incomplete inheritance is Common Objects [SNY86]. Although Snyder's system of incomplete inheritance deals with partial inherited methods, he does not explain whether he allows partial inherited instance variables.

Summary

As this section indicates, object-oriented databases have many advantages over more traditional database systems. Because of their greater flexibility, object-oriented databases are more suitable for certain classes of problems.

However, before one begins to design an object-oriented database, he must first make certain kinds of decisions. First, the designer must decide whether to implement the object-oriented database using a behavioral or structural approach. If the designer decides to use a behavioral approach, then he must further decide whether to use a special-purpose database language or an existing programming language to specify methods. Also, the designer must answer the question of who " owns " an method. The final question involves the use of multiple inheritance and the resolution of name conflicts. If the designer selects multiple inheritance, then he must also decide on a method to resolve conflicts between two inherited variables, and between two inherited methods.

CHAPTER BIBLIOGRAPHY

[BAN87] Banerjee, J., et al., " Semantic and Implementation of Schema Evolution in Object-Oriented Databases, " *Proceedings of Association for Computing Machinery Special Interest Group on Management of Data,* 1987, 311-322.

[CHE76] Chen, P.P.S., " The Entity-Relationship Model: Toward a Unified View of Data, " *ACM Transactions on Database Systems,* 1976, 9 - 36.

[CHO87] Chou, H., et al., " Data Model Issues for Object-Oriented Applications, " *ACM Transactions on Office Information Systems,* 1987, 2 - 26.

[COD79] Codd, E. F., " Extending the Database Relational Model to Capture More Meaning, " *ACM Transactions on Database Systems,* 1979, 397 - 434.

[COP84] Copeland, G. and D. Maier, " Making Smalltalk a Database System, " *Proceedings of ACM SIGMOD,* 1984, 316 - 325.

[DER85] Derrett, N., W. Kent, and P. Lyngbaek, " Some Aspects of Operations in an Object-Oriented Database, " *Database Engineering,* 1985, 302 - 310.

[DER86] Derrett, N., et al., " An Object-Oriented Approach to Data Management, " *Proceedings of Thirty-First IEEE Computer Society International Conference,* 1986, 330 - 335.

[DIT86] Dittrich, K.R., " Object-Oriented Database Systems: the Notion and the Issues, " *Proceedings of International Workshop on Object-Oriented Database System,* 1986, 2 - 6.

[FIS87] Fishman, D.H., et al., " Iris: An Object-Oriented Database Management System, " *ACM Transactions on Office Information Systems,* 1987, 48 - 69.

[HAM78] Hammer, M. and D. McLeod, " The Semantic Data Model: a Modelling Mechanism for Data Base Application, " *Proceedings of ACM SIGMOD,* 1978, 26 - 35.

[HUD86] Hudson, S. E., and R. King, " CACTIS : A Database System for Specifying Functionally-Defined Data, " *Proceedings of the Workshop on Object-Oriented Databases,* 1986, 26 - 37.

[HUD87] _____ _____, " Object-Oriented Database Support for Software Environments, " *Proceedings of ACM SIGMOD,* 1987, 491 - 503.

[KEM87] Kemper, A., P.C. Lockemann, and M. Wallrath, " An Object-Oriented Database System for Engineering Applications, " *Proceedings of ACM SIGMOD,* 1987, 299 - 310.

[KIM87] Kim, W., et al., " Composite Object Support in an Object-Oriented Database System, " *Proceedings of ACM Conference on Object-Oriented Programming Systems, Languages and Applications,* 1987, 118 - 125.

[LYN86] Lyngback, P., et al., " A Data Modeling Methodology for the Design and Implementation of Information Systems, " *Proceedings of International Workshop on Object-Oriented Database System,* 1986, 6 - 17.

[MAI85] Maier, D., A. Otis, and A. Purdy, " Object-Oriented Database Development at Servio Logic, " *Data Engineering,* 1985, 294 - 301.

[MAI86] Maier, D., et al., " Development of an Object-Oriented DBMS, " *Proceedings of ACM Conference on Object-Oriented Programming Systems, Languages and Applications,* 1986, 472 - 482.

[MAI86a] Maier, D., " Why Object-Oriented Databases can Succeed Where Others have Failed, " *Proceedings of International Workshop on Object-Oriented Database Systems,* 1986, 227.

[MAN86] Manola, F., and U. Dayal, " PDM: An Object-Oriented Data Model, " *Proceedings of International Workshop on Object-Oriented Database System,* 1986, 18 - 25.

[MEY86] Meyer, B., " Genericity Versus Inheritance, " *Proceedings of ACM Conference on Object-Oriented Programming Systems, Languages and Applications,* 1986, 391 - 405.

[MIN87] Minsky, N.H., and D. Rozenshtein, " A Law-Based Approach to Object-Oriented Programming, " *Proceedings of ACM Conference on Object-Oriented Programming Systems, Languages and Applications,* 1987, 482 - 493.

[NYG86] Nygaard, K., " Basic Concepts in Object-Oriented Programming, " *ACM SIGPLAN Notices,* October 1986, 128 - 132.

[PEN87] Penney, D.J., and J. Stein, " Class Modification in the GemStone Object-Oriented DBMS, " *Proceedings of ACM Conference on Object-Oriented Programming Systems, Languages and Applications,* 1987, 111 - 117.

[ROW] Rowe, A.L. and M. Stonebraker, " The Design of Postgres, " *Proceedings of ACM SIGMOD,* 1986, 340 - 355.

[RAM88] Ramamoorthy, C.V., and P.C. Sheu, " Object-Oriented Systems, " *IEEE Expert,* 1988, 9 - 15.

[SCH88] Schlageter, G., et al., " OOPS - An Object Oriented Programming System with Integrated Data Management Facility, " *Proceedings of the Fourth International Conference on Data Engineering,* 1988, 118 - 125.

[SHI81] Shipman, D., " The Functional Data Model and the Data Language DAPLEX, " *ACM Transactions on Database System,* 1981, 140 - 173.

[SMI87] Smith, E.K., S.B. Zdonik, " Intermedia : A Case Study of the Differences Between Relational and Object-Oriented Database Systems, " *Proceedings of ACM Conference on Object-Oriented Programming Systems, Languages and Applications,* 1987, 452 - 465.

[SNY86] Snyder, A., " Encapsulation and Inheritance, " *Proceedings of ACM Conference on Object-Oriented Programming Systems, Languages and Applications,* 1986, 38 - 45.

[WEI88] Weinreb, D., et al., " An Object-Oriented Database System to Support an Integrated Programming Environment, " Document submitted for publication.

[WOE86] Woelk, D., W. Kim, and W. Luther, " An Object-Oriented Approach to Multimedia Databases, " *Proceedings of ACM SIGMOD,* 1986, 311 - 325.

[ZDO85] Zdonik, S., " Object Management Systems for Design Environments, " *Database Engineering,* 1985, 259 - 266.

[ZHA88] Zhao, L. and S.A. Roberts, " An Object-Oriented Data Model for Database Modelling Implementation and Access, " *The Computer Journal,* 1988, 116 - 124.