ACM Crossroads Student Magazine
The ACM's First Electronic Publication

Objects Objects Everywhere

by *Todd R. Manion*

Introduction

Object oriented programming (OOP) and methodology has radically altered the way we program. Many beginning computer science classes are now taught in an object oriented language, such as Java or C++. Object oriented ideas are influencing not only how we program, but also how we think about other concepts such databases. Rob and Coronel [3] view the Object Oriented Database (OODB) historically, as the merging of two mainstream thoughts: object oriented programming and database management systems. The merging of these two technologies is the direct result of the increasingly difficult task of dealing with complex data requirements and the issues of modeling real world data with relational database technology. Issues such as the increasing use of multimedia in databases, the size of databases, and the complexity of today's databases, requires solutions far beyond relational databases. Object Oriented Database Management Systems (OODBMS) are a direct result of the emerging database technologies from the reorganization of information systems already in use and existence [3].

The purpose of this article is to describe characteristics of the object oriented database (OODB) and how it relates to the object oriented data model (OODM) and an object oriented database management system (OODBMS). This article is a summary of the concepts and ideas laid out in Peter Rob and Carlos Coronel's book titled, *Database Systems: Design, Implementation, and Management*. Underlying ideas native to object oriented programming languages will be addressed and clarified to maintain congruency throughout the overview. Overall aspects of object oriented databases will be addressed in relation to the traditional Entity-Relation model of a database. In addition, object oriented database management systems will be viewed in relation to the object oriented database and the prescribed standards agreed upon at the *First International Conference in Deductive and Object-Oriented Databases* in Kyoto, Japan. Finally, the pros and cons of the object oriented database and object oriented database management system will be evaluated leading to conclusions and suggested ideas for future research/ development.

Underlying Fundamentals: The Traditional Entity-Relationship Model

The traditional database model is the Entity-Relationship (ER) model. In this model, data is organized into rows (tuples) and columns (fields). In turn, the tuples are organized into tables. Rows are specific instances of an entity. Columns are characteristics of that entity. A separate table would model the relationship between two entities. In this table, each row would be a relationship, and each column would be the Unique ID of the two entities in the relationship.

Using a Structured Query Language(SQL) the two entities could be cross-referenced through the relationship table. The joining, through the table, provides a relationship between the two entities.

Object Oriented Data Model

To fully understand the OODB and how it differs from relational models, one must understand some terms, which fit the constraints imposed by the underlying OODM, namely what is an object. The subtle differences that apply to database concepts enabling the OODB to be a much stronger modeling technique than others, and in some cases stronger than a relational model, is important. Thus, one needs a level ground upon which to build the OODB upon the OODM. At the highest level of abstraction, according to Rob and Coronel, an object is modeling a real-world situation or entity by having unique identification, properties specific to itself, and the ability to work in conjunction with objects both of and not of the same specification [3]. An entity in an ER model does not provide this behavior.

Object Identity (OID) provides the first aspect of an object, unique identity. An OID is assigned by the system, and it is unable to be changed. Therefore, no two objects created can have the same OID. This idea should not be confused with a primary-key, since a primary-key is composed of user entered values, while an OID is assigned by the system. Removing an object from the system, removes an OID from the system and by definition the system will never assign another object the removed object's OID [3].

An object is defined by the attributes that describe the real-world entity that it is modeling. Object-oriented terminology commonly refers to these attributes as instance variables. Attributes can be of any base type supported by a programming language. In addition to data attributes, objects use functional attributes or methods in order to fulfill an additional constraint imposed by the OODM: interaction [3]. Methods allow data within the object to be accessed and allow it to access states or attributes of other objects.

Characteristics of the OODM

In addition to the above definition of an object, Rob and Coronel say an OODM must support at a minimum the following characteristics [3]:

1. The OODM must be able to provide complex object representation.
2. The OODM must be extensible. In other words, it must provide support for defining new data types and methods that are capable of operating on that object.
3. The OODM must support encapsulation or information hiding. It must be able to hide how the data is represented within the object and how each method is implemented from other objects and other entities outside of itself.
4. The OODM must exhibit inheritance. Objects will exist in a hierarchy relationship. A hierarchy relationship is a relationship where an object either is or stems from a root object. The objects in an OODB will stem from a root database object, or a sub-object from that root database object. This provides the ability to take on the attributes (data) and operations of other objects above it in the hierarchy.
5. The OODM must support OIDs as described earlier.

The Object Oriented Database

Entities and Tuples

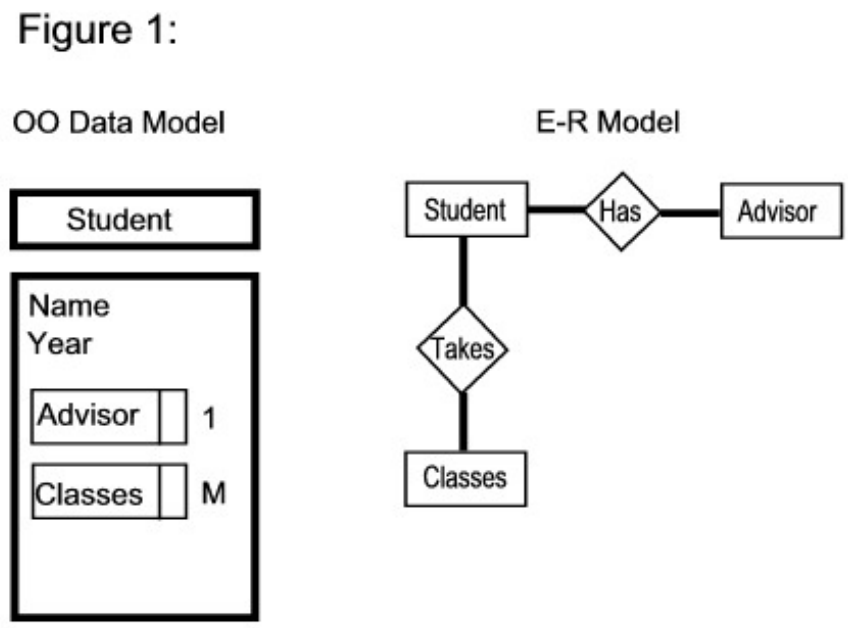

FIGURE 1: Object oriented data model compared to the ER model.

Figure 1 compares the OODM with the ER model. The two models depict a Student who *has an* Advisor and *takes* Classes. The OODM is a class containing the attributes Name and Year. It then serves as a container class that contains an Advisor object/entity and a Classes object/entity. On the other hand, the ER model has a three entity tables: Student, Advisor, and Classes. In addition, the ER model must have two relationship tables to join the information in a meaningful manner. As one can see, the ER technique needs two more entities in order to provide the relationships that the OODM approach inherently suggests.

Entity Sets and Tables

The OODMs idea of a class resembles the ER model's idea of an entity set or table. Like an object, OODM classes are more powerful then the ER model idea of an entity set or table. A class not only describes the data structure, but describes the behavior of the class objects. Features such as methods, which allow the description of behavior of objects, give an OODB full Abstract Data Type (ADT) capabilities allowing an increase in the semantics of the object being modeled. This full ADT capability also allows for encapsulation and inheritance support. These capabilities provide mechanisms for triggers (type constraints) in databases, which only a few SQL databases support [3].

Relationships

Like any data model, an important property is how it represents relationships among the different components of data. The OODB has two types: inter-class (as shown in Figure 1) or class hierarchy. This is contrasted against the ER value-based relationships. Value-based relationships are established on equal planes or equal data types between two tables. In sharp contrast, relationships in the OODB are OID based. This allows independence between the object's state and the relationship [3].

Versioning

Another difference between traditional and E-R based databases is the OODBs support of versioning. Just as a document can have multiple versions so can a database. This is especially a

desired trait to a system such as a CAD or a CASE. One is able to load a system and change aspects to determine how this would affect the overall system. Then the original system can be reloaded intact [3].

Data Access

Traditionally, data access can be viewed as a two level storage model [2]. It needs a link between the virtual/main memory of the computer and the secondary storage device in order to access the data contained in the database. The user needs the data in order to analyze and use it. In order to get the data, the user makes a query through SQL. SQL accesses the data on a secondary storage medium. It then transforms and type checks the data before it transfers the data to the user (see Figure 2).

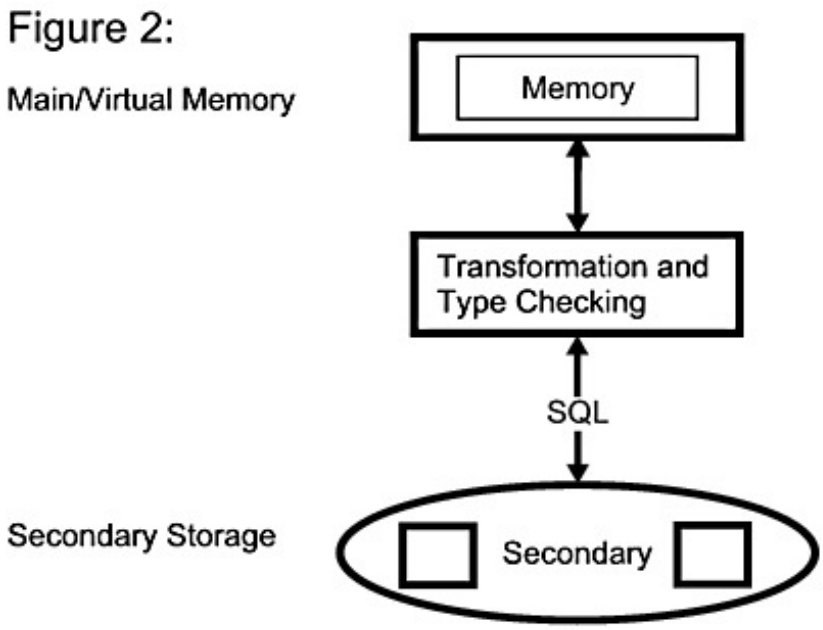

FIGURE 2: Traditional Memory Model.

The OODB eliminates the two-level storage view and provides a single-level view. An OODB does not need SQL or a transformation/type checking phase in order to bring the data from secondary storage into memory. The user instead instantiates an object. This object is the link from memory to the information on the secondary storage. The user then interacts with the object which is in memory to receive data from secondary storage (see Figure 3).

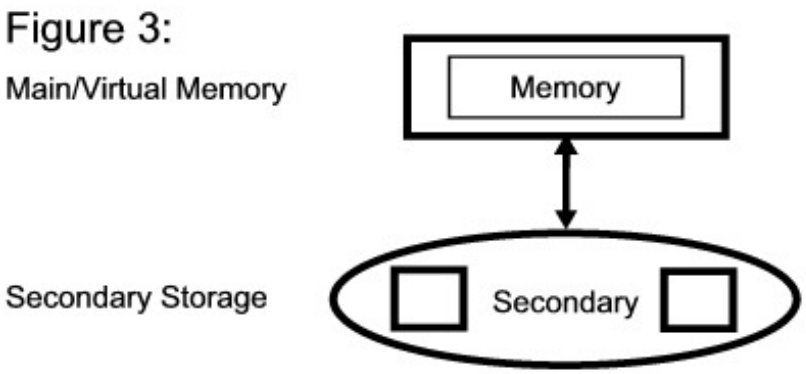

FIGURE 3: OODB Memory Model.

Another aspect of data access one must address is the issue of late versus early binding. In a traditional ER model, data type is bound early or at the definition of the table structure [2]. In contrast, an OODB allows for late binding of data types or binding of data types at run-time. In such an environment, an object's instance variable's type is not known until it is actually utilized, thus allowing any type of data to be assigned [3]. In other words, if one has two different student objects (as prescribed in Figure 1), the attribute Year can be used as a character value or a numerical value. One does not have to stay with the prescribed data type as in the traditional model. Late binding in addition to data types provides a needed aspect to allow polymorphism or the choosing of an object's method implementation at run-time [3].

A weakness of the OODM is the absence of ad hoc access to data that SQL provides. A proposal is underway, but there is no fruition at this point [3].

OODM + DBMS = OODBMS

As stated in the introduction, the point of this article is to simply provide an overview of features and aspects of OODB features and how they relate to traditional ER models. On the other hand, one cannot address OODBs without providing a management solution, which is an object oriented database management system (OODBMS). An OODBMS actually manages the data access and the querying of the data from the databases. It manages multiple users trying to access the data at the same time, provides disaster recovery services, and manages all the databases in its system. An OODBMS is an important component, since it provides and manages the constraints for the database.

Features such as persistence, transaction and concurrency control, and administration/security are as of yet in relation to OODBs on a set standard [2]. To answer this dilemma the "Object-Oriented Database System Manifesto" was introduced at the *First International Conference in Deductive and Object-Oriented Databases* in Kyoto, Japan. It provided the following musts for an OODBMS [1] (Note: Clarification is provided from [3]):

1. *OODBMSs should support complex objects.*
2. *OODBMSs should support object identity.*
3. *OODBMSs should encapsulate thine objects.*
4. *OODBMSs should support types or classes.*
5. *OODBMS classes or types should inherit from their ancestors.*
6. *OODBMSs should not bind prematurely.* It will support late binding.
7. *OODBMSs should be computationally complete.* Basic programming language notions are

supported in the Database Manipulation Language.
8. *OODBMSs should be extensible.*
9. *OODBMSs should remember thy data.* Objects keep their data in memory, unlike a database. That data is then lost on shutdown, but this is not acceptable a way must be provided for storing objects in secondary storage.
10. *OODBMSs should manage very large databases.*
11. *OODBMSs should accept concurrent users.* The OODBMS must support the same level of concurrency as present databases.
12. *OODBMSs should recover from hardware and software failures.* The OODBMS must support the same level of protection from failures as present databases.
13. *OODBMSs should have a simple way of querying data.* A method for efficient querying must be available, similar to SQL.

The Manifesto is the first attempt at describing a standard on which OODBMSs should be based. It is an important first step toward an agreement on the minimum an OODBMS should support. Once a standard is in place, OODBMSs and OODBs will become more mainstream. In addition, since it is the most significant list of requirements to be assembled, most OODBMSs are measured against it.

OODBMS Pros and Cons

According to Rob and Coronel, OODBMSs provide many benefits far and above traditional database models [3]. But there are weaknesses that the OODBMS has as well and they also should not be overlooked.

Pros

OODBMSs provide many advantages. These advantages are important because they solve many of the problems traditional systems cannot solve. First, the amount of information that can be modeled by an OODBMS is increased, and it is also easier to model this information. Furthermore, OODBMSs provide complex objects that allow ease of integration for multimedia, CAD, and other such specialized databases. OODBMSs are also able to have higher modeling capabilities through extensibility. With an OODBMS, one would be able to add more modeling capabilities, thus being able to model even more complex systems. This extensibility provides a solution for incorporating future and existing databases into one environment.

In addition to modeling advantages, OODBMSs have system advantages as well. In an OODBMS, versioning is available to help model various changes to systems. With versioning, one would be able to revert to previous data sets, and compare the current sets to the previous.

Reusability of classes plays a vital role in faster application development and maintenance. Generic classes are powerful, but more importantly they can be reused. Since classes can be reused, redundant material does not need to be designed. This leads to faster production of applications and easier maintenance of those applications and databases [3].

Cons

While there are many advantages to OODBMSs, there are also many disadvantages. Traditional ER systems have been around for a long time and a change would stray from established ideas. It would require people to think differently, and in some cases relational users lack the OO foundations needed to work with OODBMSs. Educating people on the OO foundations is a very painstaking process. It would require a significant amount of time, money, and other resources. Also, because of the change even more time would be required to move the data into the new OODBMSs.

Another disadvantage is there needs to be a way for the traditional systems and the OODBMSs to communicate and work together. Traditional systems and OODBMS must understand each other and the relations they are representing. Again, this would take time, money, and resources. Presently there is no such system for communication and understanding.

In addition, there does not exist an ad hoc query language such as SQL. While it is easier to make complex queries with OODBMSs, no query language exists. Furthermore, there are no standards for design and implementation in place and none in the foreseeable future. OODBMSs could solve problems of traditional systems, but a standard needs to be agreed upon [3].

Future Developments

Future developments for the OODB must include an ad hoc querying language for the average user. This language should provide for OODBs what SQL provides for traditional databases. Also, a standard for design, notation, implementation must be agreed upon. Future developments for the OODB could include an easier accessing method from the Internet and integration of ideas such as XML or something similar. One such initiative is the W3QL or the World Wide Web Query Language [4]. This initiative would allow one to query the web as if it is a database. Because of the vast amounts of information, an object oriented approach may prove useful.

Conclusions

There is no doubt that object oriented ideas are here to stay and they have influenced traditional relational models. The merging of the two thoughts, object oriented programming and database management systems, provides the basis for object oriented databases. Being able to represent data and relations, versioning, and simplification of data access are some of the main characteristics of OODBs. OODBMSs must include the thirteen main features delineated by the "Object-Oriented Database System Manifesto." While there are many pros and cons to OODBMSs, they are providing ways that problems can be solved and mechanisms by which traditional solutions can be incorporated into future solutions. Future development is now an even playing field as a standard for OODBs has yet to be set. Once a standard is set OODBMSs could become the database standard. As with all change, it will not come quickly, but with the introduction of the Internet and incredible amounts of data to be sifted through, solutions such as the OODB will need to be put in place to manage it successfully.

References:

1

M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. The Object-Oriented Database System Manifesto. In *Proc. of the First International Conference on Deductive and Object-Oriented Databases*, Kyoto, Japan. 1990.

2

Connolly, Thomas and Carolyn Begg, *Database Systems: A Practical Approach to Design, Implementation, and Management.* Addison Wesley Longman Limited, Edinburgh Gate, 1999, pp. 762-770.

3

Rob, Peter and Carlos Coronel, *Database Systems: Design, Implementation, and Management.* International Thomson Publishing, Cambridge, MA, 1997, pp. 545-600.

4

D. Konopnicki and O. Shmueli. *Information gathering in the world-wide web: The w3ql query language and the w3qs system.* ACM Transactions on Database Systems, Sept. 1998.

Biography

Todd R. Manion (tmanion@mission.mvnc.edu) is a senior level undergraduate student working on his BSc degree in Computer Science at Mount Vernon Nazarene College. He currently serves as Vice President of Academic Life for the student body and as President of the local ACM chapter. Recently, Todd interned at Microsoft Corporation and plans to return to industry after graduation.