International Journal of Advance Research and Innovation
Vol. 5(3), Jul-Sept 2017, pp. 17-25
Doi: 10.51976/ijari.531704
www.gla.ac.in/journals/ijari

Article Info
Received: 25 Jun 2017 | Revised Submission: 20 Jul 2017 | Accepted: 28 Jul 2017 | Available Online: 15 Sept 2017

Software Professionals Use Object Oriented Data Modeling Instead of Traditional Relational sData Modeling

*Vijay Singh**

ABSTRACT

The purpose of this paper is to explain why object oriented data modeling is more popular than relational data modeling. A data model is a logic organization of the real world objects (entities), constraints on them, and the relationships among objects. Relational model is very simple since data in represented in the form of relations that are depicted by use of two-dimensional tables. Rows in the table represent records and Columns represent attributes of the entity. The basic concept in the relational model is that of a relation. In object-oriented model main construct is an object. As in relational model, there are relations similarly we have objects in OO data modeling. So first thing in OO model is to identify the objects for the systems. Examining the problem statement can do it. Other important task is to identify the various operations for these objects. It is easy to relate the objects to the real world entity. The object-oriented approach has proved to be especially fruitful in application areas, such as the design of geographical information systems which have a richly structured knowledge domain and are associated with multimedia databases.

Relational data modeling is different from Object Oriented data modeling because it focuses solely on data while object oriented data models focuses on both the behavior and data aspects of your domain. OODBMS are faster than relational DBMS because data isn't stored in relational rows and columns but as objects. Objects have a many to many relationship and are accessed by the use of pointers.

Keywords: *Data Model; Relational Model; Object-Oriented Model; Modeling; Object; Multimedia Databases.*

1.0 Introduction

In today's world, Most of the applications use an Object Oriented Data Modeling as their data store while using an object oriented programming language for development. If the applications which are developed by object oriented programming languages use relational data modeling then to store objects, it have to be flattened into tables and when retrieved from the database the object has to be reassembled from the parts from different tables.

This causes certain inefficiency as there is need to map the objects to tuples in the database and vice versa.

In relational data modeling, there is a problem "impedance mismatch" caused by having to map objects to tables and vice versa.

Relational Data Model does not support for complex objects such as documents, video, images, spatial and time series-data. It does not provide efficient and effective integrated support for things like text searching within fields. It requires a query language like SQL while there is no need for query language in object oriented data model.

OO data modeling allows us to define our own types of objects through topological, spatial, and general relationships, which can help capture how these objects interact with other objects.

The OO modeling will be conducted in Unified Modeling Language (UML) scheme style, with a possibility to use CASE tool.

Data modeling runs through conceptual, logical and technical stages.

The purpose of this paper is to provide answers to the following questions

- What is the Data Modeling?
- What is a Relational Data Modeling?
- What is an Object Oriented Data Modeling?
- Why is the Object Oriented data modeling more used in comparison of relational Data Modeling?

Department of Computer Science, KIT, Kanpur (UP) India (E-mail: vijay.techno05@gmail.com)

- What are benefits of using an Object Oriented data modeling over Relational data modeling?

2.0 Data Modeling

A **data model** is an abstract model that describes how data are represented and accessed. Data models formally define data elements and relationships among data elements for a domain of interest.

Data Modeling is a way to structure and organize data so it can be used easily by databases. Unstructured data can be found in word processing documents, email messages, audio or video files, and design modeled and made ready for this system can be identified in various ways, such as according to what they represent or how they relate to other data. The idea is to make data as presentable as possible, so analysis and integration can be done with as little effort as necessary. It is a method used to define and analyze data requirements needed to support the business processes of an organization and by defining the data structures and the relationships between data elements.

2.1 Data modeling techniques are used-

- To manage data as a resource (migrate and merge)
- For designing computer databases.
- to better cope with change, by allowing to make changes into the model, that will automatically
- induce changes in the database and programs

A data model is composed of three levels of modeling:

2.1.1 Semantic model describes the semantics of a domain (for example, it may be a model of the interest area of an organization or industry), i.e. define the meaning of data within the context of its interrelationships and constraints with other data. It is an abstraction which defines how the stored symbols relate to the real world.

Thus, the semantic model must be a true representation of the real world. A semantic model consists of entity classes, representing kinds of things of significance in the domain, and relationships assertions about associations between pairs of entity classes. A semantic model specifies the kinds of facts or propositions that can be expressed using the model. In that sense, it defines the allowed expressions in an artificial 'language' with a scope that is limited by the scope of the model;

2.1.2 Logical model describes the system information, as represented by a particular data manipulation technology type: e.g. flat file system, hierarchical DBMS (IMS), network DBMS (IDMS, IDS2), relational DBMS (DB2, ORACLE, SQL SERVER) and Object Oriented DBMS (Ontos, DB/Explorer ODBMS). A logical model consists of descriptions of entities (called « segments » in hierarchical DBMS, « records » in network DBMS, « tables » in relational DBMS and « objects » in Object Oriented DBMS).

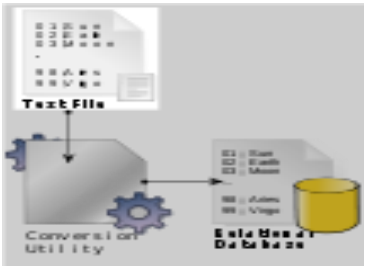
2.1.3 Physical model describes the physical means by which data are stored in a particular DBMS product (flat files, XML files, IMS, IDS2, IDMS, ORACLE, DB2, ...) . This is concerned with partitions, CPUs, table spaces, and the like. The significance of this approach is that it allows the three perspectives to be relatively independent of each other.

Storage technology can change without affecting either the logical or the semantic model. The structure (entities/attributes) can change without (necessarily) affecting the semantic model. In each case, of course, the structures must remain consistent with the other model.

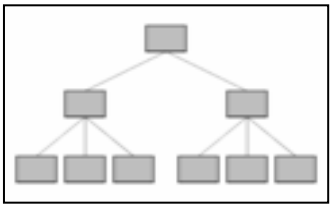
The structure implemented in a particular DBMS may be different from a direct translation of the entity classes and attributes, but it must ultimately carry out the objectives of the semantic entity class structure. Early phases of software development projects emphasize the design of a semantic data model. Such a design can be detailed into a logical data model. In later stages, this model may be translated into physical data model.

2.2 Types of data models

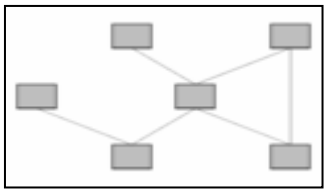
2.2.1 Flat model: This may not strictly qualify as a data model. The flat model consists of a single, two-dimensional array of data elements, where all members of a given column are assumed to be similar values, and all members of a row are assumed to be related to one another.



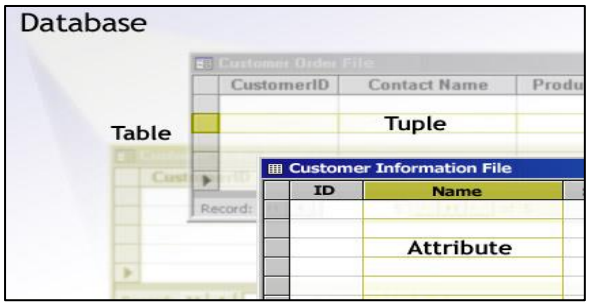
2.2.2 Hierarchical model: In this model data is organized into a tree-like structure, implying a single upward link in each record to describe the nesting, and a sort field to keep the records in a particular order in each same-level list.



2.2.3Network model: This model organizes data using two fundamental constructs, called records and sets. Records contain fields, and sets define one-to-many relationships between records: one owner, many members.



2.2.4 Relational model: is a database model based on first-order predicate logic. Its core idea is to describe a database as a collection of predicates over a finite set of predicate variables, describing constraints on the possible values and combinations of values.



2.2.5 Object oriented model: In this model data is organized into objects and relationships among the objects.

A data model is an abstract model that describes the representation and usage of data. Effective data model is required to build rapid and scalable applications, since the data model drives the application development.

In the object-oriented context, the data are modeled as units of objects and the data model describes the logical organization of the real world objects, or conceptualizing the abstractions as objects, with constraints on them, and the establishing relationships among the objects. To overcome the impedance mismatches at various levels during application development, an object-oriental data model, employing object-oriented databases and there by providing a fine abstraction to

2.3 Properties of relational tables

- Values are atomic.
- Each row is unique.
- Column values are of the same kind.
- The Sequence of columns is insignificant.
- The sequence of rows is insignificant.
- Each column has a unique name.
- Fields (Columns) In The Table Store Attributes. Each Attribute Has A Specific Domain.
- Tuples (Or Records Or Rows) In The Table Store Information. Each Tuple Is A Unique Instance Of An Object.
- Tables Are Composed Of A Set Of Tuples. A Table Is Also Called A Relation.
- Table is a collection of relevant data relating to one type of real world objects.
- Column is a specific place for one type of data relating to one type of real world objects.
- Domain is a Set of all possible values for a specific column.
- Row is a collection of data describing one real world object.
- Primary Key is a Column, which uniquely identify any one row.
- Each record represents a logical entity (e.g. a student)
- Each field represents an attribute of the logical entity

- Each table has a *primary key*, one field (or a combination of fields) that has a unique value for each and every record in the table

Certain fields may be designated as keys, which mean that searches for specific values of that field will use indexing to speed them up. Where fields in two different tables take values from the same set, a join operation can be performed to select related records in the two tables by matching values in those fields. Often, but not always, the fields will have the same name in both tables. The Relational database model is based on the Relational Algebra. It stores both

- Data about real world objects (entities) in tables
- Relationships between the tables

Table 1: Student Table

ID	Last name	First name	Grade	Age
1	Singh	Sunakshi	A	18
2	Rathore	Vijay	B	19
3	Jadon	Ajay	A	20
4	Sinha	Anurag	C	18

Id Is The Primary Key In This Table (Two Students May Share Either A Last Or First Name)

2.3.1 Relating tables

- Tables Can Be Related (Joined) Together Based On Their Keys
- The Idea Is To Decompose Into Separate Tables With No Redundancy And To Provide A Capability To Reassemble With No Information Loss

2.4 Advantages of RDBMS

- Eliminate Unnecessary Duplication of Data
- Enforce data integrity through constraints
- Changes to Conceptual Schema Need Not Affect External Schema
- Many Tools Are Available To Manage the Database

2.5 Disadvantages of RDBMS

- Until recently, no support for complex objects such as documents, video, images, spatial or time-series data. (ORDBMS are adding support these).
- Often poor support for storage of complex objects. (Disassembling the car to park it in the garage)
- Still no efficient and effective integrated support for things like text searching within fields.
- To store objects (e.g., drawings) in a relational database, the objects have to be ‘flattened’ into tables
- e.g., a digital representation of a parcel must be separated from the behaviour of other parcels
- Complex objects have to be taken apart and the parts stored in different tables
- When retrieved from the database, the object has to be reassembled from the parts in different tables

3.0 Object Oriented Data Model

An object oriented data model is a database model in which information is represented in the form of objects as used in object-oriented programming. Object data model add database functionality to object programming languages.

They bring much more than persistent storage of programming language objects. Object DBMSs extend the semantics of the C++, Smalltalk and Java object programming languages to provide full-featured database programming capability, while retaining native language compatibility.

A major benefit of this approach is the unification of the application and database development into a seamless data model and language environment.

As a result, applications require less code, use more natural data modeling, and code bases are easier to maintain. Object developers can write complete database applications with a modest amount of additional effort.

According to Rao (1994), "The object-oriented database (OODB) model is the combination of object-oriented programming language (OOPL) systems and persistent systems.

The power of the OODB comes from the seamless treatment of both persistent data, as found in databases, and transient data, as found in executing programs."

An object-oriented database management system (OODBMS), sometimes shortened to *ODBMS* for *object database management system*), is a database management system (DBMS) that supports the modeling and creation of data as objects. This includes some kind of support for classes of objects and the inheritance of class properties and methods by subclasses and their objects. There is currently no widely agreed-upon standard for what constitutes an OODBMS, and OODBMS products are considered to be still in their infancy. In the meantime, the object-relational database management system (ORDBMS), the idea that object-oriented database concepts can be superimposed on relational databases, is more commonly encountered in available products. An object-oriented database interface standard is being developed by an industry group, the Object Data Management Group (ODMG). The Object Management Group (OMG) has already standardized an object-oriented data brokering interface between systems in a network.

In their influential paper, *The Object-Oriented Database Manifesto*, Malcolm Atkinson and others define an OODBMS as follows:

An object-oriented database system must satisfy two criteria: it should be a DBMS, and it should be an object-oriented system, i.e., to the extent possible, it should be consistent with the current crop of object-oriented programming languages. The first criterion translates into five features: persistence, secondary storage management, concurrency, recovery and an ad hoc query facility. The second one translates into eight features: complex objects, object identity, encapsulation, types or classes, inheritance, overriding combined with late binding, extensibility and computational completeness.

In contrast to a relational data model where a complex data structure must be flattened out to fit into tables or joined together from those tables to form the in-memory structure, object oriented data model have no performance overhead to store or retrieve a web or hierarchy of interrelated objects. This one-to-one mapping of object programming language objects to database objects has two benefits over other storage approaches: it provides higher performance management of objects, and it enables better management of the complex interrelationships between objects. This makes object oriented data model better suited to support applications such as financial portfolio risk analysis systems; telecommunications service applications, World Wide Web document structures, design and manufacturing systems, and hospital patient record systems, which have complex relationships between data.

An Object oriented programming concepts such as encapsulation, polymorphism and inheritance are enforced as well as database management concepts such as the ACID properties (Atomicity, Consistency, Isolation and Durability) which lead to system integrity, support for an *ad hoc* query language and secondary storage management systems which allow for managing very large amounts of data. The Object Oriented Database Manifesto [Atk 89] specifically lists the following features as mandatory for a system to support before it can be called an OODBMS; Complex objects, Object identity, Encapsulation , Types and Classes , Class or Type Hierarchies, Overriding, overloading and late binding, Computational completeness , Extensibility, Persistence, Secondary storage management, Concurrency, Recovery and an Ad Hoc Query Facility.

An object oriented data model should be able to store objects that are nearly indistinguishable from the kind of objects supported by the target programming language with as little limitation as possible. Persistent objects should belong to a class and can have one or more atomic types or other objects as attributes. The normal rules of inheritance should apply with all their benefits including polymorphism, overriding inherited methods and dynamic binding. Each object has an object identifier (OID) which used as a way of uniquely identifying a particular object. OIDs are permanent, system generated and not based on any of the member data within the object. OIDs make storing references to other objects in the database simpler but may cause referential integrity problems if an object is deleted while other objects still have references to its OID. An OODBMS is thus a full scale object oriented development environment as well as a database management system. Features that are common in the RDBMS world such as transactions, the ability to handle large amounts of data, indexes, deadlock detection, backup and restoration features and data

recovery mechanisms also exist in the OODBMS world.

A primary feature of an object oriented data model is that accessing objects in the database is done in a transparent manner such that interaction with persistent objects is no different from interacting with in-memory objects. It is very different from using a relational data model in that there is no need to interact via a query sub-language like SQL nor is there a reason to use a Call Level Interface such as ODBC, ADO or JDBC. Database operations typically involve obtaining a database root from the OODBMS which is usually a data structure like a graph, vector, hash table, or set and traversing it to obtain objects to create, update or delete from the database. When a client requests an object from the database, the object is transferred from the database into the application's cache where it can be used either as a transient value that is disconnected from its representation in the database (updates to the cached object do not affect the object in the database) or it can be used as a mirror of the version in the database in that updates to the object are reflected in the database and changes to object in the database require that the object is fetched from the OODBMS.When database capabilities are combined with object-oriented (OO) programming language capabilities, the result is an object database management system (ODBMS).

Today's trend in programming languages is to utilize objects, thereby making OODBMS ideal for OO programmers because they can develop the product, store them as objects, and can replicate or modify existing objects to make new objects within the OODBMS. Information today includes not only data but video, audio, graphs, and photos which are considered complex data types. Relational DBMS aren't natively capable of supporting these complex data types. By being integrated with the programming language, the programmer can maintain consistency within one environment because both the OODBMS and the programming language will use the same model of representation. Relational DBMS projects using complex data types would have to be divided into two separate tasks: the database model and the application. As the usage of web-based technology increases with the implementation of Intranets and extranets, companies have a vested interest in OODBMS to display their complex data. Using a DBMS that has been specifically designed to store data as objects gives an advantage to those companies that are geared towards multimedia presentation or organizations that utilize computer-aided design (CAD).

Some object-oriented databases are designed to work well with object-oriented programming languages such as Ruby, Python, Perl, Java, C#, Visual Basic .NET, C++, Objective-C and Smalltalk; others have their own programming languages. ODBMSs use exactly the same model as object-oriented programming languages.

3.1 Application of object oriented data models

Areas where Object Oriented Data Modeling is heavily used.

- Engineering and spatial databases.
- Telecommunications.
- Scientific areas such as high energy physics.
- Real-time systems.
- Multimedia applications
- Molecular Biology.
- Geographic Information Systems (GIS)
- Computer Aided Design (CAD)

3.2 List of object oriented database management systems

- DB4O
- Object Store
- O2
- Gemstone
- Versant
- Ontos
- DB/Explorer ODBMS
- Ontos
- Poet
- Objectivity/DB
- EyeDB

3.3 Standards

The Object Data Management Group (ODMG) was a consortium of object database and object-relational mapping vendors, members of the academic community, and interested parties. Its goal was to create a set of specifications that would allow for portable applications that store objects in database management systems. It published several versions of its specification. The last release was ODMG 3.0. By 2001, most of the major object database and object-relational mapping vendors claimed conformance to the ODMG Java Language Binding.

Compliance to the other components of the specification was mixed. In 2001, the ODMG Java Language Binding was submitted to the Java Community Process as a basis for the Java Data Objects specification. The ODMG member companies then decided to concentrate their efforts on the Java Data Objects specification. As a result, the ODMG disbanded in 2001.Many object database ideas were also absorbed into SQL: 1999 and have been implemented in varying degrees in object-relational database products. In 2005 Cook, Rai, and Rosenberger proposed to drop all standardization efforts to introduce additional object-oriented query APIs but rather use the OO programming language itself, i.e., Java and .NET, to express queries. As a result, Native Queries emerged. Similarly, Microsoft announced Language Integrated Query (LINQ) and DLINQ, an implementation of LINQ, in September 2005, to provide close, language-integrated database query capabilities with its programming languages C# and VB.NET 9. In February 2006, the Object Management Group (OMG) announced that they had been granted the right to develop new specifications based on the ODMG 3.0 specification and the formation of the Object Database Technology Working Group (ODBT WG). The ODBT WG plans to create a set of standards that incorporates advances in object database technology (e.g., replication), data management (e.g., spatial indexing), and data formats (e.g., XML) and to include new features into these standards that support domains where object databases are being adopted (e.g., real-time systems). On January 2007 the World Wide Web Consortium gave final recommendation status to the XQuery language. XQuery has enabled a new class of applications that managed hierarchical data built around the XRX web application architecture that also provide many of the advantages of object databases. In addition XRX applications benefit by transporting XML directly to client applications such as XForms without changing data structures.

3.5 Object-oriented database advantages

3.5.1 Complex data object

Using an object-oriented database for data storage brings powerful advantages to applications that use complex object models, have high concurrency requirements, and large data sets. It is difficult, time consuming, expensive in development, and expensive at run time, to map the objects into a relational database and performance can suffer. Object-oriented database solutions are designed to handle the navigational access, seamless data distribution, and scalability often required by these applications. When data handling requirements are simple and suitable to rigid row and column structures an RDBMS might be an appropriate solution. However, for many applications, today's most challenging aspect is controlling the inherent complexity of the subject matter itself - the complexity must be tamed. Tamed in a way, that enables continual evolution of the application as the environment and needs change. For these applications, an object-oriented database is the best answer.

3.5.2 Complex (inter-) relationships

If there are a lot of many-to-many relationships, tree structures or network (graph) structures then object-oriented database solutions will handle those relationships much faster than a relational database.

3.5.3 No mapping layer

It is difficult, time consuming, expensive in development, and expensive at run time, to map the objects into a relational database and performance can suffer. object-oriented database solutions store objects as objects - yes, it's as easy as object database solutions are designed to store many-to-many, tree and network relationships as named bi-directional associations without having the need for JOIN tables. Hence, object database solutions save programming time, and objects can be stored and retrieved faster.

3.5.4 Fast and easy developement, ability to cope with continous evolution

The complexity of telecommunications infrastructure, transportation networks, simulations, financial instruments and other domains must be tamed. Tamed in a way, that enables continual evolution of the application as the environment and needs change. Architectures that mix technical needs such as persistence (and SQL) with the domain model are an invitation to disaster.

Versant's object-oriented databases solutions let you develop using objects that need onlzy contain the domain behavior, freeing you from persistence concerns.

4.0 Comparisons of Object Oriented Data Models to Relational Data Models

There are concepts in the relational database model that are similar to those in the object database model. A relation or table in a relational database can be considered to be analogous to a class in an object database. A tuple is similar to an instance of a class but is different in that it has attributes but no behaviors. A column in a tuple is similar to a class attribute except that a column can hold only primitive data types while a class attribute can hold data of any type. Finally classes have methods which are computationally complete (meaning that general purpose control and computational structures are provided while relational databases typically do not have computationally complete programming capabilities although some stored procedure languages come close.

Below is a list of advantages and disadvantages of using an Object Oriented Data Models over a Relational Data Models with an object oriented programming language.

4.1 Advantages

4.1.1 Composite objects and relationships:

Objects in an OODBMS can store an arbitrary number of atomic types as well as other objects. It is thus possible to have a large class which holds many medium sized classes which themselves hold many smaller classes, ad infinitum. In a relational database this has to be done either by having one huge table with lots of null fields or via a number of smaller, normalized tables which are linked via foreign keys.

Having lots of smaller tables is still a problem since a join has to be performed every time one wants to query data based on the "Has-a" relationship between the entities. Also an object is a better model of the real world entity than the relational tuples with regards to complex objects. The fact that an OODBMS is better suited to handling complex, interrelated data than an RDBMS means that an OODBMS can outperform an RDBMS by ten to a thousand times depending on the complexity of the data being handled.

4.1.2 Class hierarchy:

Data in the real world is usually having hierarchical characteristics. The ever popular Employee example used in most RDBMS texts is easier to describe in an OODBMS than in an RDBMS. An Employee can be a Manager or not, this is usually done in an RDBMS by having a type identifier field or creating another table which uses foreign keys to indicate the relationship between Managers and Employees. In an OODBMS, the Employee class is simply a parent class of the Manager class.

4.1.3 The need for a query language:

A query language is not necessary for accessing data from an OODBMS unlike an RDBMS since interaction with the database is done by transparently accessing objects. It is still possible to use queries in an OODBMS however.

4.1.4 No impedance mismatch:

In a typical application that uses an object oriented programming language and an RDBMS, a significant amount of time is usually spent mapping tables to objects and back. There are also various problems that can occur when the atomic types in the database do not map cleanly to the atomic types in the programming language and vice versa. This "impedance mismatch" is completely avoided when using an OODBMS.

4.1.5 No primary keys:

The user of an RDBMS has to worry about uniquely identifying tuples by their values and making sure that no two tuples have the same primary key values to avoid error conditions. In an OODBMS, the unique identification of objects is done behind the scenes via OIDs and is completely invisible to the user. Thus there is no limitation on the values that can be stored in an object.

4.1.6 One data model:

A data model typically should model entities and their relationships, constraints and operations that change the states of the data in the system.

With an RDBMS it is not possible to model the dynamic operations or rules that change the state of the data in the system because this is beyond the scope of the database. Thus applications that use RDBMS systems usually have an Entity Relationship diagram to model the static parts of the system and a seperate model for the operations and behaviors of entities in the application. With an OODBMS there is no disconnecting between the database model and the application model because the entities are just other objects in the system. An entire application can thus be comprehensively modelled in one UML diagram.

4.2 Disadvantages

4.2.1 Schema Changes:

In an RDBMS modifying the database schema either by creating, updating or deleting tables is typically independent of the actual application. In an OODBMS based application modifying the schema by creating, updating or modifying a persistent class typically means that changes have to be made to the other classes in the application that interact with instances of that class. This typically means that all schema changes in an OODBMS will involve a system wide recompile. Also updating all the instance objects within the database can take an extended period of time depending on the size of the database.

4.2.2 Language Dependence:

An OODBMS is typically tied to a specific language via a specific API. This means that data in an OODBMS is typically only accessible from a specific language using a specific API, which is typically not the case with an RDBMS.

4.2.3 Lack of Ad-Hoc Queries:

In an RDBMS, the relational nature of the data allows one to construct ad-hoc queries where new tables are created from joining existing tables then querying them. Since it is currently not possible to duplicate the semantics of joining two tables by "joining" two classes then there is a loss of flexibility with an OODBMS. Thus the queries that can be performed on the data in an OODBMS is highly dependent on the design of the system.

5.0 Conclusions

The gains from using an OODBMS while developing an application using an OO programming language are many. The savings in development time by not having to worry about separate data models as well as the fact that there is less code to write due to the lack of impedance mismatch is very attractive. In my opinion, there is little reason to pick an RDBMS over an OODBMS system for *new* application development unless there are legacy issues that have to be dealt with.

References

[1]. OODBMS Facts. Barry & Associates. http://www.odbmsfacts.com

[2]. Atkinson, Malcolm. The Object-Oriented Database Manifesto. In Proceedings of the First International Conference on Deductive and Object-Oriented Data-bases, Kyoto, Japan, 1989, 223-40, www.cs.cmu.edu/People/clamen/OODBMS/Manifesto/htManifesto/Manifesto.html

[3]. McFarland, Gregory, Andres Rudmik, and David Lange. Object-Oriented Database Management Systems Revisited. http://www.dacs.dtic.mil/techs/oodbms2/

[4]. S.N. Woodfield, The Impedance Mismatch Between Conceptual Models and Implementation Environments, in ER'97 Workshop 4 Proceedings, 1997

[5]. Database For Objects (DB4O) - http://www.db4o.com

[6]. M. Winston, R. Chaffin, and D. Herrmann, A Taxonomy of Part-Whole Relations, Cognitive Science, 11, 1987, 417-444.

[7]. PolePosition open source database benchmark - http://www.polepos.org/