DEVELOPMENT OF DATA PERSISTENT FRAMEWORK FOR MODELLING ENTITY AND ENTITY RELATIONSHIP USING DB4O OBJECT DATABASE WITH JAVA

KEN YAP CHEE KIAN

MASTER OF COMPUTER SCIENCE

FACULTY OF ENGINEERING & SCIENCE
UNIVERSITI TUNKU ABDUL RAHMAN
OCTOBER 2013

ABSTRACT

DEVELOPMENT OF DATA PERSISTENT FRAMEWORK FOR MODELLING ENTITY AND ENTITY RELATIONSHIP USING DB4O OBJECT-ORIENTED DATABASE WITH JAVA

Ken Yap Chee Kian

Object database is studied with some experiments performed on the DB4O database and a persistent object framework with data access library is developed. The structure of persistent object of the framework conceptually mapped to a relation of relational data model; this allow direct adaptation and reorganization of existing relational design to OODB. The relationships of entities in runtime is modeled by two memory references from two respective persistent objects in Relationships persistent object; which is much similar to introducing a table for Many-to-Many relationship in relational design. In the framework, the associated members in a relationship can be retrieved, however multiple connected relationships retrieving which is similar to Join in RDB is not supported by the framework. An application level design strategy is suggested associated with the usage of the framework and instability is found in using DB4O static method in deletion of persistent object in DB; removal of collection item instead of deletion is suggested to improve the situation the problem, i.e. not a satisfactory fix. The specific achievements of the framework appears to be cut down of design and development effort of application, and suitable for integration with computer aided design tools for rapid application development.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I would like to thank my supervisor Mr. Sugumaran for his effort and patience in guiding me in this project. I would like to thank my family for giving me care so that I can be healthy and be able to complete my Master course while studying and working.

APPROVAL SHEET

This dissertation/thesis entitled "***DEVELOPMENT OF DATA PERSISTENT FRAMEWORK FOR MODELLING ENTITY AND ENTITY RELATIONSHIP USING DB4O OBJECT-ORIENTED DATABASE WITH JAVA***" was prepared by Ken Yap Chee Kian and submitted as partial fulfillment of the requirements for the degree of Master of Science in Computer Science at Universiti Tunku Abdul Rahman.

Approved by:

____

Mr. Sugumaran
Professor/Supervisor
Department of Computer Science
Faculty of Engineering & Science
Universiti Tunku Abdul Rahman

DECLARATION

I hereby declare that the dissertation is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at UTAR or other institutions.

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS AND SYMBOLS

OODB	Object-Oriented Database
OO	Object-Oriented
JCBC	Java Database Connectivity
API	Application Programming Interface
DB	Database
SQL	Structured Query Language
CPU	Central Processing Unit
DBMS	Database Management System
ER	Entity Relationship
NoSql	Not Only Sql

CHAPTER 1

INTRODUCTION

Computational machine and devices are pervasive in our society now, information in various form is available to the public connected to the internet. More and more people are using mobile devices in their daily activities, there is a great need emerged to drive technology in serving this great number of people, supporting them in their tasks as well as their pleasure in using an application. The advancement is expected to change the way people work, play, and social, it definitely change the economy in a very dynamical and unpredictable way.

This period is also a time to witness the power of distributed technology, a distributed system is a system, operating system, application that appear to the user as a single system with the knowledge that in fact there are many connected parts that are working behind the scene. It increased the computational powers and gives optimal utilization of its resources such as disk space and memory. What is worth noting about distributed technology is what we called the cloud computing in our present day. Virtualization technology certainly contributed to the success path of cloud computing. Cloud computing brings the computing of an entreprise or business to a third party owned cloud infrastructure that highly utilized distributed technology. It affects businesses in many dimensions, businesses are given the options to ward off many laborous efforts in maintenance, scaling, problem support that normally required by its owner, and now owner do not have the need to invest heavily on hardware, software or services. They can now host their system at cloud provider like Amazon Cloud Computing and put their resources, concentrations and focus on their core business and application development and leave the rest to the cloud provider.

When the user uses an information system or an application, information that is presented to the user is normally come from database. Database is a “base” for data stored in a persistent physical storage such as a hard disk, so that data is persisted beyond the time electrical current is cut off from the machine or device. A front end system can be an interface for the user who interacts with the data it wrapped behind, it normally do not presenting the data in “raw” form, it may have altered, transformed, and formatted the data it get from the database before it is presented to the user so that information is adapted to the context of the application requirements. For example, a bank manager at a branch may need to access his client contact information through the bank internal information system, but at the same time, the bank business processing unit may also need to look into the client credit risk exposure in order to make decisions to approve or reject the client application to the bank financial products. To calculate associated risk, the bank backend system may have to make request to external sources of database as well.

To make the system scale to support more users, there are also many technological breakthrough in database technology. However, this is not the only factor involved to make advancement in database technology. There are many aspects that the users of the database system expect depending on the nature of the application. Application chooses database technology that best suites its needs, one database technology may benefits an application in certain ways, but may not be suited for another kind of application. In fact, when one database technology is used in one system, not all criteria may be satisfied, and there is a trade-off that the decision maker may be aware of when choosing a particular kind database technology. For example, MongoDB, a so-called NoSQL database may be good for website like eBay for scaling capability, but it may not be suitable for some internal business information system that strongly demand transactional functionality of database that is seen lack in MongoDB in current state.

Though, in general, there are some features that a database should possess, at least minimally. Firstly, it should provide persistency because that is the main function and purpose of database. It must also have the ability to delete, retrieve, update, and create data in the physical storage and, it should provide security, so that data is protected from alteration, modification, and accessed by unauthorized person, system, or program. An example of this kind of vulnerabilities is SQL injection, where the attacker can add additional SQL statement to an existing and retrieving the data from the database, and present it on the attacker browser.

The way that a database satisfied the above written features is totally depending on the vendor of database technology. Therefore, classification can be made for how different databases was build. How the data are represented, the storage file system, how records relationship is represented physically, are important design factors that led to the advantages of a particular database or the disadvantage as well. If execution time is a concern in a very request intensive multi-user application, the database organization that increases disk searching will be on the disadvantage side, as mechanical dynamics of scanning the circular disk takes relatively longer time than CPU execution time. One particular database may not have the ability to model a different kind of data that one industry demands because it cannot represent it physically. A good database design will shield the user from the need to know the physical operations details in order to work with the database, thus it provides the user with the logical view of the data semantics, without supplying any low-level details in commanding the operations of database. On one part of modern database development, the physical and logical separating has been a continuous effort, for example, the emergence of declarative SQL standard for relational database, which optimization strategy of a query is auto-selected and circumvent the need to indicate the indexing details in accessing.

How the data are represented, designed, organized, maintained in the database is as important as how easy the database is adapted to change. Can the database support the cost incurred in the growth of the data and the evolution of data format at the same time maintaining other imposing rules? Or the case where the database only allowed a fixed format and length of data that by default not expecting it to be changed? The growth of data and its evolution have important implications to the overall benefits of having a database. There are many other respects that we can take a look at database, performance wise, or utility wise to gauge whether it served the purpose it meant to be.

Apart from performance or utility aspects like whether it has security implemented, the database has to make it easy for human to work with it. As database developers or application developers will need to spend efforts, hours and expert knowledge in using the database to build system. If it is not robust, not user-friendly, requiring big effort input and time-consuming, it will incur cost for a business, for example resulting in project delayed, lack of human resources and its implications. Therefore, the maintenance and development by human is an important aspect of database itself.

This project is about using an object-oriented database known as DB4O to develop application data persistent framework in modelling persistent entity and their relationships, it is NOT an application Model-View-Controller framework.

Object-oriented database is a new paradigm of database as compared with relational database. Object-oriented database is entirely fashioned by the application programming language, there is no need to have a middleware such as JDBC to talk to the database and the application. Object-oriented database may or may not have its own query language, as it can be allowed to use the application language to select data. It is reported that object-oriented database is superior than relational database in performance wise.

Entity aforementioned in our context are conceptual abstraction used to represent a real world entity such as person, object or event. The conceptual abstraction can be used to design a database or application. The relationships of the entities in the design normally can tell us about business's process graphically.

The framework aforementioned on data persistency is a programming framework that can be used to generalize a persistent object derived from conceptual entity, i.e. the data that are meant to be stored in database. This framework is believed to provide rapid application development, unification and consistency for the system, and the ability to model complex object and evolution of its structure, these are among other benefits.

In the development of this project, each important designing decision point is selectively recorded in this report. This chapter will continue with the problem statement and research methodology, relevant knowledge of object-oriented database which also include DB4O is written in the second chapter of literature review. The technical of how to work with DB4O is also covered in the second chapter. The framework developed is described in the third chapter. The forth chapter described implementation of a web application using the framework. The last chapter gives the conclusion on the project.

1.1 Problem Statement

No discovery of opportunity or no attention given to a discovered opportunity to build, improve a system should be considered a problem. The first problem can be dealt with by examining more closely of a system, or experimenting on the system so that the unknown part can be made known. The problem that this project aiming belong to the latter, a discovered opportunity to improve an application that uses database. Therefore, to verify whether this opportunity is indeed an opportunity that improves a system, work must be put forward to justify it.

Before precisely describing the problem, which is seen as an improvement opportunity, there are some facts on object-oriented database DB4O that leads to the discovery of the opportunity.

In DB4O, the data model schema may not need to be introduced to the database before it is being inserted with a first new data of a class. It is in program runtime that a new object with its schema can be persisted to the database that initially, have no knowledge of the structure of the object. Therefore, we can skip the schema definition design work effort and relay it to the application programming. The database design becomes the work of application developers and not touching the database administration system, this is to say that database design is one with the design of application. Comparing with relational database, it is true because we are able to identify that the object correspond to the relational tuple, and the class of an object corresponds to the schema/metadata of the relation or table. With object-oriented database, object can now directly persisted to the database without breaking down to smaller units.

Object-oriented features such as inheritance and encapsulation can be applied into the database with DB4O database, these features can be used to build a framework for persistent object, which is object that is not transient and requires persistency in the context of the application. This framework is NOT related or tied to any business processes of the application, rather it is a general data persistence framework that is independent of the business processes and business process can build on top of this framework.

1.1.1 What Motivates the Development of the Framework?

There exist common internal and external operations subjected to all persistent object

We assume that we have many various persistent objects derived from different classes. And, we assume that there are some common static operations for all of such objects, and these common operations are related to persistency. Giving this scenario, an experience developer would motivate the design of a static library class to accommodate the static operation that accepts persistent object as argument. But in a typed system of Java, this operation is broken into many pieces of method signatures as each object varies in their classes and are orthogonal to each other, and the signature of the method changes as it takes in the type of the persistent object as a specification. This violate the principle of software reuse. Persistent operations exist for persistent object and it should be common to all persistent object. Therefore, there is a need to develop a framework for persistent object for DB4O, so that the principle of software engineering is not violated. Business entity that interacts in the processing of program, and is deriving from the storage, should be a subclass of the persistent object class, which endowed the entity with some behaviors and states that are useful in persistency, accessing and other conveniences.

Factorization of Entity Relationships

The standard tutorial guide available in the public access on DB4O currently gives us a picture of how relationships are modelled in object-oriented database. The picture is an object that contains the reference pointing to the other object or vice-versa, or both pointing to each other (bi-directional). When the relationship is moved from memory to storage, the references are swapped with the object identification, which is unique to each object putting into storage. Vice-versa, when an object is loaded into the memory from storage, the identification is replaced with object's memory reference. The link allows one to traverse from one object to the other object or collection of objects that had some kinds of relationship between. However, this is somehow restricted in the way data is accessed because one may have to sequentially or direct access to one object in order to get to the other object if the latter object has no identity to be searched and is uni-directional in the link. A tree structured object network will need an amount of work to access relevant object to the leaf of the tree. Therefore, it is restricted in this sense. There is an opportunity that we can factor out the relationship to be an independent persistent object containing two objects as a binary relationship. By this, the database has an additional entry point of accessing in the sense that relationships can be first accessed in order to get to the left or right-handed object.

1.2 Project Aim and Objectives

The need for the framework is certain. The aim of the project is to design and build a data persistent framework using Java and DB4O object-oriented database, the framework is a layer between the object database and the application.

1.2.1 Objectives

1. To find out more about the object database by experimenting on it.
2. Find satisfying or optimal decisions in designing a persistent object.
3. The persistent object data structure must be able to model complex entity.
4. The persistent object must be able to model entity that evolve in their structure over time.
5. Model relationships as a persistent object.
6. Find out all the “how” to build the framework.
7. Find the drawback/shortcoming of the designed system.

CHAPTER 2

LITERATURE REVIEW

Object-oriented databases emerged as the convergence of several research threads. It was induced by many other fields such as software engineering, artificial intelligence, programming language that applying object-oriented technology.

Object-oriented is a paradigm of programming language, it is a product of modern computer development. Programming language has developed from low-level to high-level language, but it is not to say that the past achievement is having lower credential. Low-level language like assembly language are not discarded because it is old, it is still being applied to certain area such as micro-controller programming. High-level language popularly known as C language was designed by Dennis Ritchie in 1973, it is the programming language for developing UNIX operating system. Before this there were some other language like Pascal, Cobol, and SmallTalk. Pascal and C is a structure programming language, but SmallTalk is already an object-oriented language. The first object-oriented language is the Simula. Simula started as a project in 1962. The goal was to build a tool to describe discrete event system, or network, and a language to simulating real world. In 1964, Simula 1 had been implemented on Univac 1107. It was used to control administrations, airports, planning, transport, or social systems. Therefore, we can see that the first object-oriented programming was born when programing a real world problem, which does not only compose of set of functions, or structure based algorithms. This language has introduced classes, inheritance and objects that are instances of classes. Classes allow to link functions (methods) to objects. The object-oriented version of C came in 1986 as C++. A very prominent object-oriented language Java came in the 1994, now it is a proven technology that is widely used in industry such as banking. And, it is usually a language that is used in teaching object-oriented programming in universities. Newer popular programming language like C Sharp or .NET are fashioned in object-oriented as well.

The relationship of data use in application written in a particular language with the language itself is a rich knowledge. This knowledge is also gathered over time with the development of programming language and database. It started where there is no database for all the maintaining and administration, it started with file system or data directly resided in the programming file, this is in the beginning state of modern computer development. There is a strong coupling of data with the application, this is not a good design as we know it now. The organization details of the data in the file will directly affect the way programming is written, if data is reorganized, program will have to be rewritten. So the first effort has been to separate application and data into application and a well-defined database and organized data in certain ways that achieve invariant in terms of the coding change with respect to data reorganization or data change. However, data independence is not achieved in the beginning within the database system. This is achieved as the operation of data is relayed to the database management system. Therefore, this has established database management system as another system working together with the application.

Database is a collection of data and metadata on how data is organized that is stored and is used to support application that request data or request for a change, modification, creation of data, analysis of data from the database management system (DMBS). A database is not generally portable across different DBMS.

There were two popular data models in the early of database development, a network model called CODASYL and a hierarchical model called IMS. They emerged in the 60s when computers became cost-effective for private companies. These two models put more emphasis on modelling the data into the physical organization and direct access, and less emphasis on the event of change of data organization. In these model, pointers linked various kind of records together, by following the path of the link, record can be accessed, it is a navigational approach data model. When data organization is changed, often the access and modification scheme has to be rewritten. The user of the database have to supply the knowledge of the physical in order to work with it, and it makes it difficult and less robust when there is a change in the physical. Although these models did make impressive progress at that time, as it provides data application separation by factorized the data management functionality out from the application, it is still restricted in certain way pertaining to data independence.

CODASYL Data Definition Language Commitee in its 1978 Journal of Development increased the degree of data independence by separation of external and internal operations. Data independence is the ability to modify a schema definition in one level without affecting a schema definition in the next higher level. In 1970, a paper by E.F. Codd [CODD] addresses the data independence problem with proposing a relational approach. The relational model disconnects the physical storage of data from its logical schema. Relational approach apply elementary relation theory to systems which provide shared access to large banks of formatted data. It lays down the theory of relational database based on relational algebra which has been led to the invention of SQL query language, which has be widely received and adopted as a standard in the 80s and ever since the relational database boom in the business market until now.

With relational database, content based query that directly accessing any record type is facilitated. Foreign key allow establishing relationships between records to be retrieved. Primary key field is used for identification of a record within a relation. As more and more applications are developed in object-oriented language with relational database, soon a problem known as impedance mismatch was noticed. Impedance mismatch in database context is the mismatch between the data manipulation language of the database and the general-purpose programming language in which the rest of the application is written [BM]. As it does not match, there are some loss of information occurs at the interface, for example, object behaviour would be lost, object data member name is replaced, object class is lost by transmission of signal to the database. When the signal flow from the database to the application through the interface, it anticipates some amount of work to reconstruct from a relation to an object that is interacting in the application. The work is often a cost in development, the cost for object- relational mapping which has a lot of configuration details. On one hand it increases the development burden of developer and the other hand it slow down the performance.

In 1976, A new database model called Entity-Relationship, or ER, was proposed by P. Chen. This model allows designers to focus on data application, instead of logical table structure. An entity diagram is illustrated in the following.

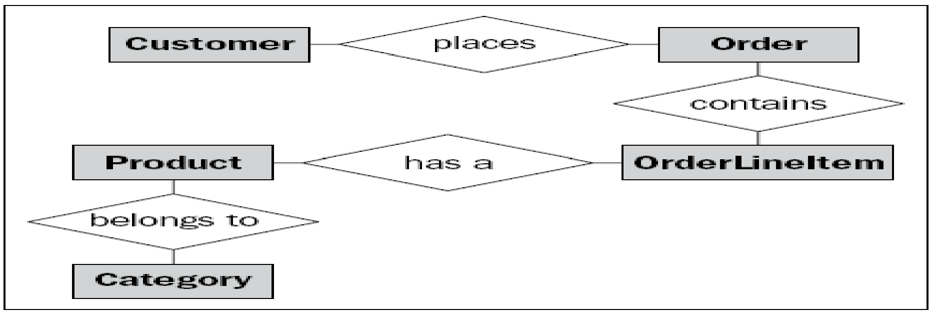


Figure 1. ER Diagram.

In the 1980s, when SQL becomes a standard in relational database, it was realized that it is not practical to model some data types in physics, multimedia, CAD and some other domain that has complex data types. Object-oriented database had its presence in 1980s to remove the inconvenience of the so-called object-relational impedance mismatch. This is along with the rise of object-oriented programming where data used in the program is considered to belong to an object. There is also another stream of development to overcome object relational impedance mismatch offer by object-relational mapping tools or product of an effort to incorporate object-oriented features in relational database. The incorporated relational database is called object relational database, it is often the extension of existing relational database. Object relational mapping provides a mapping between the relational and the object model, so that the relational details is made transparent to the programmer.

Object-oriented programming gain further popularity in 2004 when easy to use and affordable open source database management system like DB4O emerged. Other databases in the 2000s are some post-relational database known as NoSQL databases, they are fast and do not require fixed table schemas, with no join operation and able to scale horizontally. This kind of database is used to support a large scaled web applications such as facebook.

2.1 Preliminary

Data is a fact, something upon which an inference is based, it is simply value or sets of values.

Data Item is smallest named unit of data that has meaning in the real world, it is a single unit of values.

Entity is something that has certain attributes or properties which may be assigned values. The value themselves may be either numeric or nonnumerical.

Entity Set is formed by set of entities with similar attributes.

Field is a single elementary unit of information representing an attribute of an entity.

Record is group of related data items treated as a unit by an application program. It is a collection of field values of a given entity.

File is the collection of records of the entities in a given entity set.

Information is used for data with given attributes, or in other words, meaningful or processed data.

Database is an organized body of related information.

Database Management System is a software system that facilitates the creation and maintenance and use of an electronic database.

Persistent Data refer to data that exist from session to session, persistent data are usually stored in a database on disk or tape.

Database Server is a collection of programs that enable users to create and maintain a database.

2.2 What Makes A Database Management System

This section outlines some features of a database management system.

1. Persistence

Pertaining to the storage of data.

2. Concurrency

Ensures correctness of competing multiple access to the same data, the access are treated in a coherent and reliable way summed up commonly as a acronym of ACID

ACID:

(a) Atomicity

Transactions execute entirely or not at all, if one of the operations is going to fail, the whole transaction is totally disregarded.

(b) Consistency

Transactions move the database from one consistent state to the next consistent state, if consistency is violated, the whole transaction is rolled back to the consistent state before the execution.

(c) Isolation

No partial effects of incomplete transactions are visible.

(d) Durability

Successfully-completed transactions are permanent, cannot be undone.

3. Integrity

The accuracy and consistency of data.

4. Security

Restricted access of the data.

5. Data Independence

Delarative queries and update that not affected by changes in the storage structure and access methods.

6. Backup and Recovery

2.3 Object-Oriented Concept

Class

A class represents a concept, and an object on the other hand represents the embodiment of a class, it is a blueprint for an object, the data type of an object.

Object

Object is data structures consisting of data fields and methods, it is a specific instance of a class. It can refer to a physical object, such as computer, vehicle, or person. It is a data abstraction that is defined by an object name as a unique identifier, valued attributes (instance variables) that give a state to an object, and methods or routines that access the state of the object. The state of the object is actually a set of values of its attributes. The specified methods are the only operations that are allowed to carried out on the attributes in the object. An object is usually drawn as a rectangle having an object name and its properties as shown below

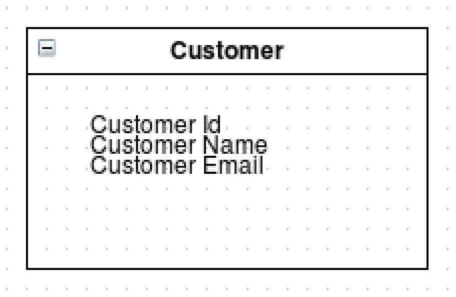


Figure 2. Class Diagram

Object-Oriented Programming

It is a programming paradigm that use objects and their interactions to design applications and computer programs. It is a methodology that focuses on data rather than processes, with programs composed of self-sufficient modules called objects, each having the ability to manipulate its own data structure. This is as opposed to the conventional model, in which program is seen as a list of tasks or subroutines to perform.

2.3.1 Features of Object-Oriented Programming

(a) Inheritance

Inheritance is generally known as a generalization or specialization relationship, in which the definition of a class can be based on other existing class. A child (subclass) class can be created from existing ones, and inherits

its properties from its parents class (superclass). It is illustrated in the following diagram

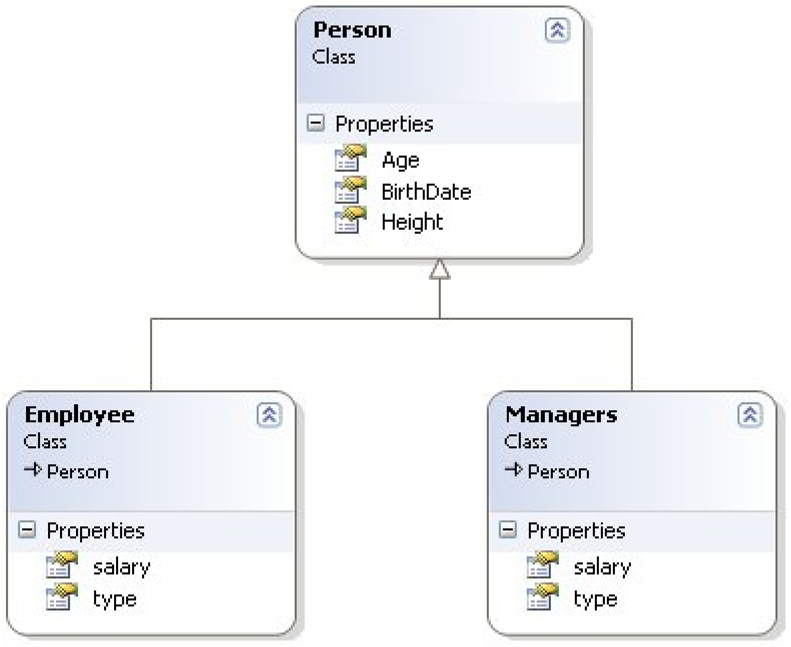


Figure 3. Inheritance Relationship of Class

In above diagram, employee class and manager are the subclass of person, person can be considered a generalization of employee and manager, and they are considered specialization of the person.

(b) Polymorphism

The ability of objects to respond differently to the same message or function call.

(c) Encapsulation

It is the principle of separating the implementation of a class from its interface and hiding the implementation from its clients and only providing external interfaces for manipulation of the data, it is the process of binding data and methods together.

2.4 Object-Oriented Database Management System

Object database management system is a database management system and it is also an object-oriented system as well, it combines the features of an object-oriented language and a database management system.

Object Database store object directly without having to break the object into data level and store it in relational manner. Therefore, the persisted data matches the object more closely, it is stored similar as serialization of object except that it provides additional database features. Serialization turns an object into stream of data thatcan be read from and written to a file, but it has some limitations and does not support remote access over a network.

Object Database can model well the relationships between objects, which are inheritance, association and aggregation, and these relationships are associated with constraints.

2.4.1 Features of Object-Oriented Database System

Object Identifier

Object identifier is discussed by Khoshafian and Copeland [KC] relating to fundamental differences between relational database and object-oriented database. Object Identifier is used by the system uniquely identify objects in database, when the object is no longer in memory, the memory reference is no longer can be used to express the associations of objects. Therefore object identifier is needed whenever object is persisted beyond program execution. However, this identifier is transparent to the programmer, and it is not accessible or used across the application. It is different from primary key, primary key is visible and can be altered.

Computational Complete

The operation on the database is restricted by the sql query, object database however is not restricted in the way it interact with the data, as it is modeled in the programming language that is able to formulate any expressions in the query.

Support All Features of Object-Oriented Programming:

It supports encapsulation, inheritance, overriding, late-binding.

Persistence Mechanism

<u>(a) Persistence by Class</u>

Declare all objects of a class to be persistence.

<u>(b) Persistence by Creation</u>

Extend the syntax for creating objects to specify that an object is to be a persistent.

<u>(c) Persistence by Marking</u>

An object that is persist beyond program execution is marked as persistent before program termination.

<u>(d) Persistence by Reachability</u>

Object becomes persistent if they are referred to from a declared object.

2.4.2 Comparison of Object and Relational Database

The relational model so far has been the most widely used and successful in model data, it represents data entities as tabular form known as relations or tables, the relationships between relations are defined by primary keys and foreign keys. The final persistence representation is a normalization that removed redundancy in improving storage efficiency, data integrity, etc.

Object-oriented programming has become a paradigm that most developers will not consider to code in procedural language nowadays, as the major programming platform like .NET from Microsoft and Java platform are all object-oriented based.

The mismatch between relational data model and object-oriented application model becomes evident and it creates additional burdens for developers. The impedance mismatch is a results of the need to convert the relational data model to assemble object in their application. The conversion is needed as the semantics of the application is lost when it is persisted. Data access of relational database is using SQL query, sql query has its good as a declarative language that is easy to manipulate, and it can be used to retrieve relationships by joining table. It has a strong mathematical foundation behind and it is a standard that conformed by various vendor, this making it popular in the market.

Yet, in object-oriented database there is no data access standard or protocol, as opposed to SQL language of relational database. However in terms of performance, object-oriented is superior as it does not suffer from the slowdown from joining tables, this is because the relationships in objects retrieved from object database is navigational, it provides a direct reference to the next object in a particular relationships, unlike relational where it needs to reconstruct the relationships by join operation.

Another factor put object-oriented database in a favorable position is that it can more suited to represent semi-structure data, multiple-valued data and even more complex data model, where it is being apply in some applications like CAD application, engineering design. It offers flexibility in modeling data.

Object-oriented database also promised to improve software quality and efficiency. One in such enticing benefits is reusability. Object classes can be stored and used across all projects, this can reduce the effort in development cost and more effort can be devoted to improve other areas such as correctness and robustness.

Object oriented database is developing into maturity in future that may overtake relational database, which is fully developed.

2.5 Existing Object Data Persistent Framework

Hibernate (http://www.hibernate.org) and iBatis (http://ibatis.apache.org) are two data persistent framework available in the market. Hibernate is picked here for illustrations of a data persistent framework.

Hibernate is a data persistent framework integrated with the object-relational mapping. The data persistency is realized in the logical model of the application. The following diagram shows its position in a developed system.

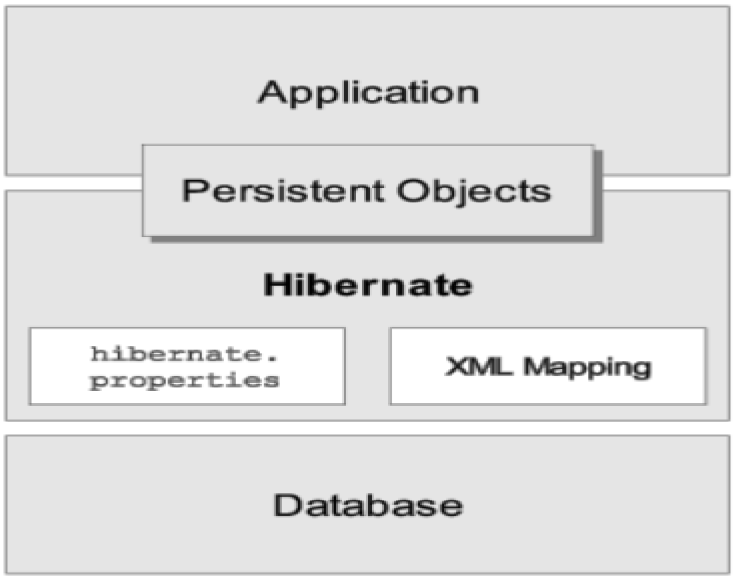


Figure 4. Hibernate.

In the above, persistent objects is both shared by application and the framework, Hibernate keeps these objects as cache and keep track of any changes that are made to the persistent object, transient object can be in the memory of application, but have no references in the hibernate system.

By using Hibernate,

- Many database operations that are frequently written such as opening and closing database can be omitted, as it is all managed by Hibernate. Therefore it reduced the code lines for programmer.
- Programmer do not have to concern with the data type of the underlying database, programmer only concern with the attribute of the persistent object.
- Increase performance by providing cache of objects.

- By simple parameter modification, database can be migrated to another easily.

2.5.1 Hibernate Session and Object Relational Mapping

Hibernate provide an object by class named Session, it takes care of the connection to any relational database. Some of the methods it provides are

- load/get
- save/persist
- update/merge
- delete

These method directly deals with persistent object, for example

```
public void updateBranch() {
        Configuration config = new Configuration().config
        ure();
        SessionFactory sessionFactory = config.buildSes
        sionFactory();
        Session session = sessionFactory.openSession();
        Transaction trans = session.beginTransaction();
        try {
                EmpBranch branch = new EmpBranch();
                branch.setBranchName("branch_0005");
                session.persist(branch);
                trans.commit();
        } catch (Exception e) {
                trans.rollback();
                e.printStackTrace();
        } finally{
                if( session.isOpen() )
                session.close();
        }
}
```

In above the EmpBranch object class is defined as

```
public class EmpBranch implements java.io.Serializable{
        private Long branchId;
        private String branchName;
        public EmpBranch(){
        }
        public Long getBranchId(){
                return this.branchId;
        }
        public void setBranchId(Long branchId){
                this.branchId = branchId;
        }
        public String getBranchName(){
                return this.branchName;
        }
        public void setBranchName(String branchName){
                this.branchName = branchName;
        }
    }
```

The table that corresponds to the EmpBranch persistent object is

```
    create table EMP_BRANCH
    (
        BRANCH_ID NUMBER not null primary key,
        BRANCH_NAME VARCHAR2(50),
        TIMESTAMP DATE
    )
```

In order to map the class that the persistent object is derived to the table, there is a mapping configuration in the form of xml. Following the same example, it is given as

```
    <?xml version="1.0" encoding="utf-8"?>
    <!DOCTYPE  hibernate-mapping  PUBLIC  "-//Hibernate/Hi
    bernate      Map   ping          DTD          3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-      map
    ping-3.0.dtd">
    <hibernate-mapping>
    <class name="com.hibernate.vo.EmpBranch"
    table="EMP_BRANCH"
    lazy="true"
    dynamic-insert="true"
    optimistic-lock="version"
    batch-size="5">
    <id name="branchId" type="java.lang.Long">
```

```
<column name="BRANCH_ID" precision="22" scale="0"
/>
<generator class="sequence">
<param name="sequence">seq_sales</param>
</generator>
</id>
<property                    name="branchName"
type="java.lang.String">
<column name="BRANCH_NAME" length="50"/>
</property>
<set      name="empDepartments"      inverse="true"
lazy="true"          cas    cade="save-update">
<key>
<column    name="BRANCH_ID_EMP"    precision="22"
scale="0" not-     null="true"/>
</key>
<one-to-many         class="com.hibernate.vo.EmpDepart
ment"/>
</set>
<version    name="timestamp"    column="timestamp"
type="java.util.D ate"></version>
</class>
</hibernate-mapping>
```

In the xml document above, the property tag refers to the object data member and the column tag refers to the table column, therefore the branch-Name from the object is mapped to the table column BRANCH_NAME.

2.6 Working with DB4O

In this section, we introduce how to work with the DB4O database, it also includes the experiments performed on the database. With various persistency mapping like Hibernate, DB4O provides a better transparent persistency that is more efficient. DB4O is designed to require no database administration. It requires less effort and offer rapid development for engineering an application as the need to construct object schema, object mapping can be eliminated in DB4O. The database can be open and operate as an embedded server or a network server to support various kind of applications.

Open Database

DB4O database is a single file, the database can be open by calling a static method, specified the path to the file as an argument, if the file does not exist on the path, then it will create a new file on the path. Example:

```
ObjectSet db = Db4o.openFile(“/home/user1/db.yap”);
```

Persist Object

Unlike relational manner, there is no mention of any table name and columns name in db4o object database. And, there is no need tointroduce schema or work out object relational mappings. In db4o, Object Definition Language is not necessary to introduce object schema into the database, as the schema is just simply the object class definition itself. We may or may not have to manually assign an object property to be the identifier of the object. Example given as follow,

```
Person person = new Person();
person.setName(“Peter”);
person.setAge(“24”);
db.store(person);
```

Retrieving Object

DB4O offers a few ways of retrieving objects. It is given as

(a) By Example

By passing a new object of a subclass with or without setting the properties of the object, matching objects stored in the database can be retrieved. For example,

```
Query query = db.queryByExample(new
Person(“Peter”));
ObjectSet result = query.execute();
```

(b) By SODA Query

SODA Stands for Simple Object Data Access, it provides APIs for performing complex queries on the database. For example,

```
Query query = db.query();
query.constrain(X_Person.class);
query.descend(“Name”).constrain(“Peter”);
ObjectSet result = query.execute();
```

(c) By Native Query

A native query is written in the language of application, it is not text based that needed interpretation by the persistent engine. By passing in the predicate object with the matching expression, it returns the matching objects. For example,

```
List persons = db.query(new Predicate(){
        public boolean match(X_Person person){
                return person.getAge() > 21;
        }
}
```

Update Object

In order to update, the object must be retrieved from the database while keeping the database open, if the object is allocated in the memory before database open, the object will be treated as a new object and not as updating. The object is considered a new object because the database does not keep track of its object ID and its memory reference anymore. Unlike relational database, there is no need to explicitly specify which member data to be updated, it reduces the possible errors of writing sql statement. For example,

```
ObjectSet result = (ObjectSet) db.get(new
Person(“Peter”);
if( result.hasNext() )
        Person peter = (Person) result.next();
peter.setAge("25");
db.store(peter);
```

Delete Object

In order to delete, we select the object out from the database into memory, similar to updating, for example,

```
ObjectSet result = (ObjectSet) db.get(new
Person("Peter"));
if( result.hasNext() )
        Person peter = (Person) result.next();
db.delete(p);
```

DB4O Client Server Access

DB4O can be started as a service that listens for and accept connections, operation such as data retrieval, updates, deletions can be requested by the client connected to the service. Networking mode and embedded mode are supported by DB4O database.

<u>(a) Embedded Mode</u>

In embedded mode, the database and application are in the same virtual machine, there is no need to specify the IP Address and port number.

<u>(b) Networking Mode</u>

It puts DB4O database service in a distributed environment that can be accessed from computers, PDAs, handheld devices or cell phones. Example below shows how DB4O can be wrapped as a server.

```
public class RunServer implements Runnable{
        private boolean stop = false;
        public void run(){
        synchronized(this){
        ObjectServer server =
		DB4O.openServer("/home/user1/db.yap",
		"8123"); //port number 8123
		server.grantAccess("user1", "password");
		try {
			while( !stop ){
				System.out.print("Server running...");
				this.wait(60000);
			}
		} catch( Exception e ){
			e.printStackTrace();
		}finally{
			server.close();
		}
		}
		}
	}
```

To connect to the database, the client can call method in the following manner.

```
	private ObjectContainer client;
	client = DB4O.openClient(IPAddress, 8123, "user1",
	"password");
```

2.7 Experiments On DB4O Database

2.7.1 Experiment 1 on Deletion of Object

This experiment is to obtain findings on the deletion of objects and its side-effects if existed. An object may be referenced by multiple objects that are stored in the database, what is the outcome when the object is deleted?

<u>Experiment 1 Steps:</u>

1. An object class is designed given below.

```
public X_Object{
        List<X_Object> objectList = new ArrayList<X_Ob
        ject>();
        X_Object obj;
}
```

2. Two objects are created in the program.

```
X_Object obj1 = new X_Object();
X_Object obj2 = new X_Object();
```

3. The second object reference is added to the first object List collection type.

```
obj1.objectList.add(obj2);
```

4. The reference of second object is assigned to the first object instance variable named obj.

```
obj1.obj = obj2;
```

5. Save/store both objects.

```
db.store(obj1);
db.store(obj2);
```

6. Delete the second object

```
db.delete(obj2);
```

<u>Outcome and Validation for Experiment 1</u>

The second object cannot be retrieved, it returned null that is true as it is deleted. The first object can be retrieved, and the reference to the second object "Obj1.obj" is null, which is also correct and acceptable as second object is removed. Similarly the first object instance variable "objectList" should had zero size, as the second object reference is no longer existed. But it was not so, the size of 1 remained and when we iterate over the list collection that is "obj1.objectList", it ran into exception. This can be understand as the reference is made null, but the size of the collection is not corrected by the database. Therefore, the data are not cleanse and there is side-effect in directly using "db.delete(obj2)" for removing reference item in a collection. We saw the side-effects of deletion on collection type. This problem can be mitigated by enforcing a protocol in programming, which every deletion of object in a collection has to make a removal of that item in the collection before doing the deletion, for example

```
obj1.objList.remove(obj2);
db.delete(obj2);
db.store(obj1);
```

2.7.2 Experiment 2 on Schema Change of Database Class

Experiment 2 Steps

1. An object class is designed.

```
public class X_Object{
        private String property = “PROP_VALUE”;
        public String getProperty(){
                return property;
        }
}
```

2. An object is created in the program and saved.

```
X_Object object = new X_Object();
db.store(object);
db.close();
```

3. The program and database are terminated and the class is altered into as follows.

```
public class X_Object{
        private String property = “PROP_VALUE”;
        private String property2 = “PROP_VALUE2”;
        public String getProperty(){
                return property2;
        }
        public void setProperty2(String property){
                this.property2 = property2;
        }
        }
```

4. Start the program and database with fresh class loaded and retrieving the object.

Outcome and Validation for Experiment 2

The object stored can be retrieved, the changed method can be used and “property2” is accessed and return null. Null value returned is acceptable because the first object did not store this property when it was first created. If we try set the value to the second property within the scope by

```
        obj.property2 = “ANOTHER_VALUE”;
```

or using

```
        obj.setProperty2(“ANOTHER_VALUE”);
```

It is found that it cannot be stored and it is always be null thoughout the program for that particular object. For the objects that are created and persistent after the schema change, the instance variable is having presence in the created persistent objects.

CHAPTER 3

RESEARCH METHODOLOGY

The project research methodology involved gathering knowledge on database and object-oriented programming. I am collecting information on the development trail of database, and information on the development trail of programming. I put these time-variant developments of the two together and observe if there is any part of the two that convergence into each other. This information allow me to see the connection between database and application programming language. I am looking into specifically object-oriented database and comparing it with partnership of relational database with an application development in object-oriented language, it is noticed that the gap has been the so-called impedance mismatch, some questions to ask in the research are how to minimize the loss of information as a results of impedance, and where object-oriented database is different from relational in partnership with object-oriented application. Experiments to be conducted on DB4O database to verify some of its behaviors, questions that this project is interested to obtain from the experiments are

- Findings on the deletion of objects and its side effects if existed.
- Findings of database on changes on the class of object, either the changes comes from method signatures, method implementation method or the form of class from adding or removing instance data.

Question like whether the object can still be retrieved on class changed? What happen to the old persisted object if it can be retrieved? The comparison of Hibernate data persistent framework with the framework to be developed is important part of this project.

3.1 Tools and Resources

1. Programming Language: Java 1.6
2. Server: Glassfish Server (optional)
3. Hibernate version 4
4. Object Database: DB4O 8.0
5. Operating System: Linux Fedora

CHAPTER 4

THE DESIGNED FRAMEWORK

4.1 Generalization of Entity

An entity in the real world is represented by an object in object-oriented application. A general class to model an object belong to any entities is defined. Naming of PO is used to represent persistent object. The following diagram shows the implementation class of persistent object.

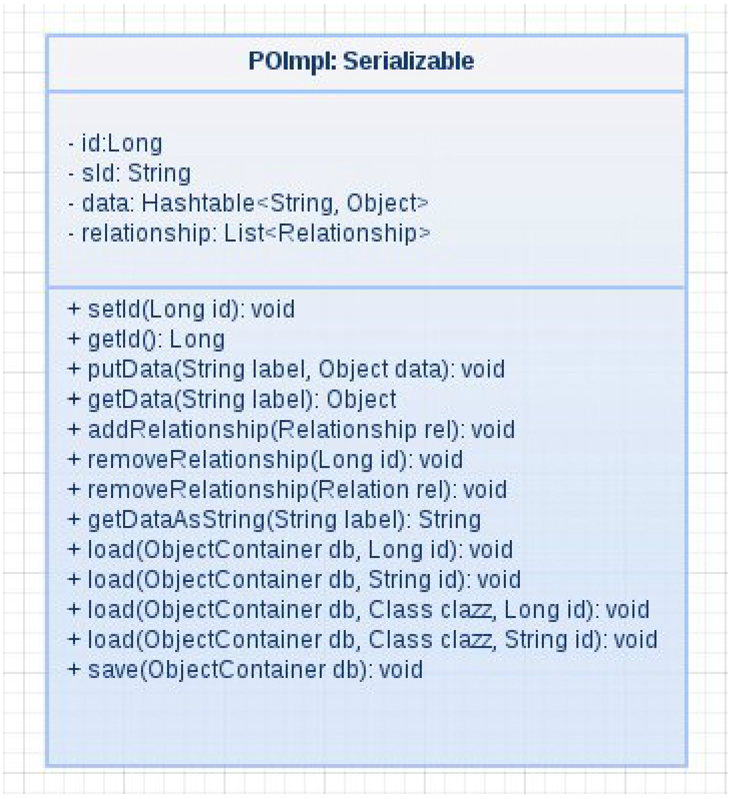
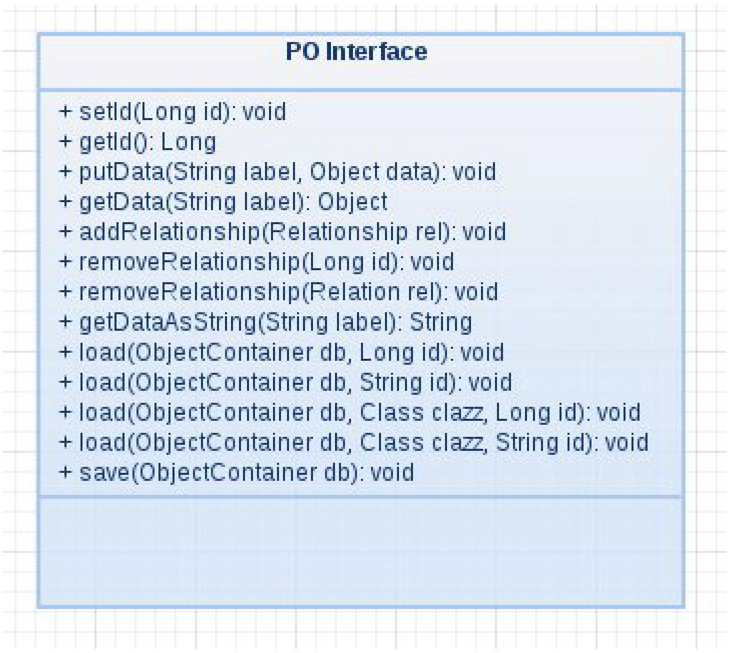

POImpl: Serializable
- id:Long
- sId: String
- data: Hashtable<String, Object>
- relationship: List<Relationship>
+ setId(Long id): void
+ getId(): Long
+ putData(String label, Object data): void
+ getData(String label): Object
+ addRelationship(Relationship rel): void
+ removeRelationship(Long id): void
+ removeRelationship(Relation rel): void
+ getDataAsString(String label): String
+ load(ObjectContainer db, Long id): void
+ load(ObjectContainer db, String id): void
+ load(ObjectContainer db, Class clazz, Long id): void
+ load(ObjectContainer db, Class clazz, String id): void
+ save(ObjectContainer db): void

Figure 5. Persistent Object Class.

The persistent object implements the interface of persistent object given below



```
PO Interface

+ setId(Long id): void
+ getId(): Long
+ putData(String label, Object data): void
+ getData(String label): Object
+ addRelationship(Relationship rel): void
+ removeRelationship(Long id): void
+ removeRelationship(Relation rel): void
+ getDataAsString(String label): String
+ load(ObjectContainer db, Long id): void
+ load(ObjectContainer db, String id): void
+ load(ObjectContainer db, Class clazz, Long id): void
+ load(ObjectContainer db, Class clazz, String id): void
+ save(ObjectContainer db): void
```

Figure 6. Persistent Object Interface.

With respect to the persistent object class presented above, some of the its details are outlined in the following.

<u>The "id" instance variable</u>

The id is ID is used for identification of the persistent object, it is optional, but it is strongly advised to reserve it as it can be used for retrieving purpose similar to primary key in relational database. The auto generation of the id is not provided by the database, as opposed to the fact that in relational database id is allowed to auto generate or auto increment, therefore, the application designer can provide the function to generate id for the persistent object. This role is given to the application designer, whether the id is unique throughout all objects, or it can be unique to just within objects of the same class.

<u>The "data" instance variable</u>

Instance variable named data is used to store value of data of any type, as java.util.Object is the parent class of all primitive and user defined object. The

String type in the Hashtable is for labeling of the data similar to the column name in relational table, it labels the data value and becomes part of data that saved into database. The instance variable data concept in this project embodied a row in the relational table, except that the data label is saved along with the data value. Collection type Hashtable is used instead of HashMap, this is because Hashtable provides synchronization that allows multiple write access to be executed in orderly manner without causing corruption of data, and only acceptable state of data is visible to the subsequent accessing thread.

<u>The "relationship" instance variable</u>

Instance data member named relationship is used to store the relationships between different persistent objects, note that there is no foreign key involved in this technique. It can store relationships of persistent object belonging to the same class or different classes as long as it belongs to the subclass of the persistent object class named RelationImpl.

<u>Method "getData"</u>

Method named getData return the data in Object form, the programmer needs to be aware of its type before it can be cast and subject to its permissible operations according to its type. Because of this overhead of type knowledge required in advance before writing the code, it is encouraged here to adopt a design strategy that all data processing to be done in business application layer to use this method to access the data. In this strategy, if possible all necessary computations are encouraged to perform before it is persisted in the database, by this, we assume that all data that are persisted is a final processed product and always ready to be retrieved. So that in retrieving in presentation layer of application we can always use getDataAsString which require no knowledge of the type of the data.

Method “getDataAsString”

Method named getDataAsString allows all types of data to be cast to String that is generally for retrieving purpose, it can be written as following.

```
public String getDataAsString(String label) throws
Exception{
        if( data == null )
                return null;
        Object obj = record.get(label);
        if( obj == null )
                return null;
        if( obj instanceof String )
                return (String) obj;
        else if( obj instanceof Integer || obj instanceof
                Double || obj intanceof Boolean )
                return obj + “”;
        else
                return obj.toString();
}
```

For data belonging to user defined class, a “toString” method or an overriding version of “toString” method can provide the String representation of the object. Method named getData return the data in Object form, the programmer needs to be aware of its type before it can be cast and subject to its permissible operations according to its type.

Method “load”

The method loads the data into the persistent object from the database.

Method “save”

The method save is used to persist the object by passing the DB4O object container as argument. It is written in the following.

```
public void save(ObjectContainer db){
        X_DBUtil.save(this, db);
}
```

The save method implementation is provided by the database utility class that accept the current persistent object and the passed in object container as arguments. The save method is common to both update and saving of a new object operations. The saving of persistent object is simply by calling the save method as follows

```
obj.save(db);
```

4.1.1 Extension of PO Type

All persistent object modelling a real world business entity in the framework is a subclass of the persistent object class named POImpl depicted below.

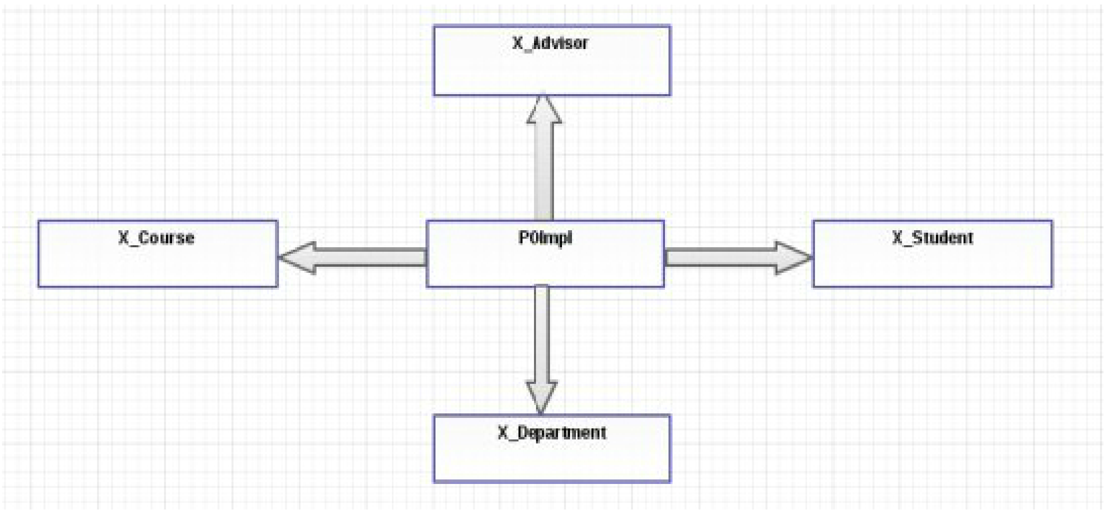


Figure 7. Differentiation of Persistent Object.

All persistent object subclass can be either manually written as simple as the following.

```
public X_POSubClass extends POImpl{
        //declaration of some other data member or
        methods specific to the subclass
}
// if having interface of subclass then
public X_POSubClassImpl extends POImpl implements
SubClass{
        //declaration of some other data member or
        methods specific to the subclass implementation
}
```

The above code can also be auto-generated by programming aided design tools and group it under the same packages directory. For using programming aided design tool, in simple case, the name the subclass say as above “X_POSubClass” is a parameter provided to the tool.

4.2 Entity Relationships

The Relationship object is an extension of persistent object class POImpl such that it is treated the same way as a persistent object depicted below.

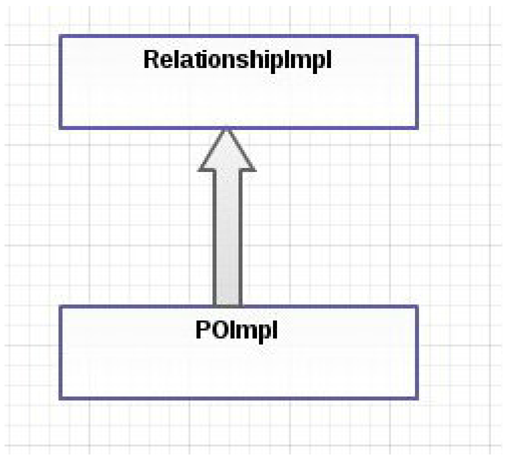


Figure 8. Relationship as Persistent Object.

This allows data concerning the relationships can be stored in the instance variable named data in the persistent object. The class for relationships of persistent objects is illustrated below.

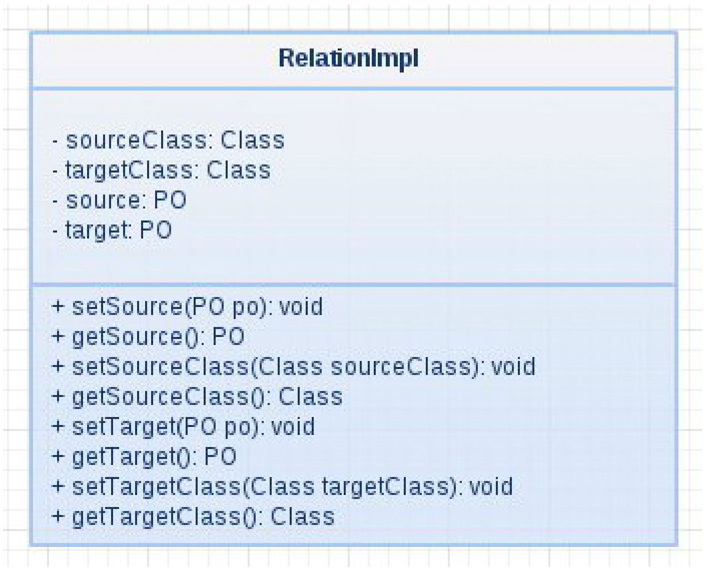


Figure 9. Relationship Persistent Object Class.

The relationship object binds only two objects having some relationship together, given two reference storage variable for the two objects, the framework introduced an order to indicate vector in the relationship rather than having less information to differentiate the two objects. The order is given by naming the two object reference variables into “Source” and “Target”. For example, a borrow record in a library, the borrow relationship can put student and book objects together as follows

```
public static void borrow throws q
Exception(ObjectContainer db, X_Student          student,
X_Bio book)
{
        X_StudentBio_Borrow_Relationship
        borrowRel = new
        X_StudentBio_Borrow_Relationship();
        borrowRel.setSource(student);
        student.addRelation(borrowRel);
        borrowRel.setTarget(book);
        book.addRelation(borrowRel);
        borrowRel.setBorrowDate(new Date());
        ...
        borrowRel.save(db);
}
```

Data concerning the relationship of borrow such as borrow date can be stored as data on this relationship persistent object. In the framework, the student in the example can be named as a “source” while book as a “target”, source can be used to represent an object that initiated an action say borrow applying on the target such as book. This provides a mindset that makes it easy in designing the system based on the framework.

With respect to the persistent object class presented above, some of the its details are outlined in the following.

Source and target

The source and target instance data member are to hold a persistent object that is taking part in a relationship.

sourceClass and targetClass

The instance variable named sourceClass and targetClass are used to store the class of the source and target persistent object. It can be used to validate the object with the correct type before it is to be stored into the relationship persistent object demonstrated as follows

```
public void setSource(F_PO source) {
        if(sourceClass.isInstance(source))
                this.source = source;
}
```

Method “setSourceclass” and “setTargetclass”

The two methods are used to store the source class and target class, it should be fixed in design time and not be able to change externally, therefore private access modifier is used.

4.2.1 Extension of Relationship Persistent Object

Similar to the persistent object class, any other relationships of persistent object can be modelled by extending the relationship persistent object class. Name of the relationship class is encouraged to convey the two entities involved in the relationships, a rule is suggested below in naming a relationship class

System Differentiation Symbol+“_”+First Entity Name+Second Entity Name+“_”+Relationship

A model of student advisor supervises a student, for example will have a class of “X_AdvisorStudentRelationship” extending the “RelationshipImpl”. The class is implemented as below

```
public class X_AdvisorStudent_Relationship extends
RelationshipImpl
{
        private X_AdvisorStudent_Relationship(){}
        public static X_AdvisorStudent_Relationship get(){
                X_AdvisorStudent_Relationship rel = new
                X_AdvisorStudent_Relationship();
                rel.setSourceClass(X_Advisor.class);
                rel.setTargetClass(X_Student.class);
                return rel;
        }
}
```

The constructor is delared as private so that the object cannot be instantiated without going through the static method named “get” that endowed the object with the correct source and target class before it can be used in the program. The get() method is to be called to instantiate the relationship persistent object because any persistent object must check against the type before it can be set into the relationship, by setting the source and target class, checking can be done when needed. The above class can be auto-generated by computer aided design tools accepting only two String valued persistent object class name, as the relationship name can be derived from two object class name.

Steps to store the relationship is given below.

1. Create or retrieve two persistent object from the database.

```
X_Advisor advisor = DBUtil.get(advisorId,
X_Advisor.class, db);
X_Student student = DBUtil.get(studentId,
X_Student.class, db);
```

2. Create the relationship persistent object.

```
X_AdvisorStudentRelationship rel =
X_AdvisorStudentRelationship.get();
```

3. Put the two persistent object into the relationship respectively.

```
rel.setSource(advisor);
rel.setTarget(student);
```

4. Put the relationship persistent object into the advisor and student persistent object.

```
student.addRelationship(rel);
advisor.addRelationship(rel);
```

5. Save the relationship persistent object.

```
rel.save(db);
```

With the data structure of relationship persistent object, one-to-one, many-to-one, and many-to-many relationships can be expressed and treated the same. This is contrary to relational where one-to-one, many-to-one relationship is modeled by foreign key and many-to-many relationship is modeled by introduction of an additional table. It is a single treatment for all relationships.

<u>Ways to retrieve the relationship is given as</u>

1. Navigational

First retrieved the “source” or “target” persistent object, and get relationship persistent reference from the object retrieved, this is a navigational approach.

2. Relationship class with object value

Get the relationship persistent object by giving the class of the relationship persistent object subclass and specific value in the properties for identification.

3. Relationship class with source or target object reference

Get the relationship persistent object by giving the class of the relationship persistent object and the source or target object reference as property to retrieve corresponding results.

4. Relationship class with ID

Use an ID instance variable to uniquely determine a relationship persistent object.

5. ID

Without providing any class, using an ID that is unique across all persistent object for retrieving.

By method of 2,3,4 and 5, it can be seen that relationships can be factored out from the database by the framework, so that not only the framework can support navigational in retrieving relationship object via method 1, it can have another entry point to the object database.

4.3 Data Access Object Utility

A common utility methods is written for accessing and storing of data in object-oriented database. Some of the utility methods is given below.

1. Message: getPO(String id, Class<PO> class, ObjectContainer db)

Return: a unique persistent object by the ID provided.

2. Message: getPO(Class<PO> class, ObjectContainer db)

Return: ObjectSet<PO>

By providing a persistent object class or its subclass, it returns a collection of persistent objects of that particular class. The class resembles the table name of a relational sql statement.

3. Message: getPO(HashMap<String, Object> constrains, Class<X_PO> class, ObjectContainer db)

Return: ObjectSet<PO>

The first method searches the object based on ID attribute, this method searches the objects based on any other properties given in the HashMap where the key is the label of the data, value by type Object is the matched value for the results. However this method is restricted by only matching equality of the object attributes.

4. Message: getPOByExample(PO po, ObjectContainer db)

Return: ObjectSet<PO>

By providing an object as argument with or without setting the properties, matching objects can be retrieved from the database.

5. Message: getPOByNativeQuery(Predicate pred, ObjectContainer db)

Return: ObjectSet<PO>

This method allows a predicate that formulate the matching criteria in the programming language. Therefore, it is not restricted to only equality matching and making the query computational complete.

CHAPTER 5

Implementation, Applying the Framework

5.1 Demo: Back Office Management of Web Video Site

A simple web application is developed to demonstate the framework. The web application chosen is used for online video site to manage their collection of videos. The video site does not store or allow users to upload any video, it is not a sharing site, the video that is provided to the users is a link to some other video sharing sites such as Youtube. The video site pull video feed from Youtube and save the video ID to its own database and some other metadata of the video. It is not intended to be a giant database, as it only store minimal information of the video. The data that is stored can be used to retrieve the video from Youtube and relay it to the user. It is not liable for any legal implications of video sharing as it only provide link to the video source site. The video data that are stored are organized, processed so that video to be shown can have categories and make it easy for users to select video for watching pleasure.

5.2 Data Model

There are three entities in this demonstration, which are video category, series, and series video. Series, for example is a popular tv series. Series video for example, is the episode of the tv series. The video category categorized the tv series. Category and series is many-to-many relationship, and series contains one to many series video.

Three class files correspond to the three persistent object is introduced, one of it is given as below.

```
public class F_Category extends F_POImpl {
}
```

Two relationship class files are needed, they are given as below.

```
public class F_CategorySeries_Relationship extends
F_RelationshipImpl{
        public F_CategorySeries_Relationship(){
                setSourceClass(F_Category.class);
                setTargetClass(F_Series.class);
        }
}


public class F_SeriesSeriesVideo_Relationship extends
F_RelationshipImpl{
        public F_SeriesSeriesVideo_Relationship(){
                setSourceClass(F_Series.class);
                setTargetClass(F_SeriesVideo.class);
                }
        }
```

5.3 Results

Create Category Persistent Object

1. “Add Category” is clicked in the screen shown below.

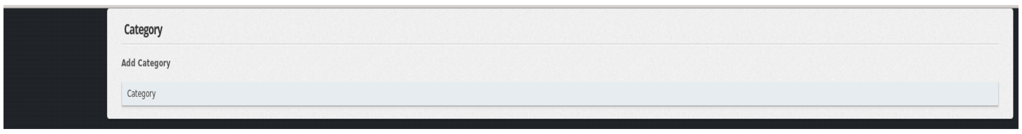
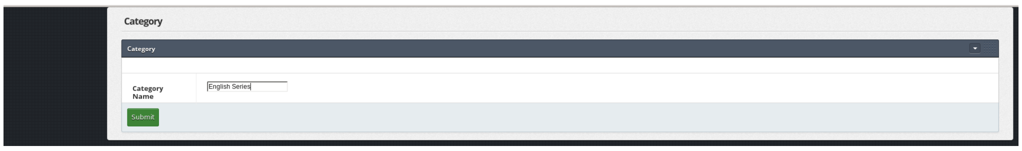


Figure 10. Add Category in Category Listing

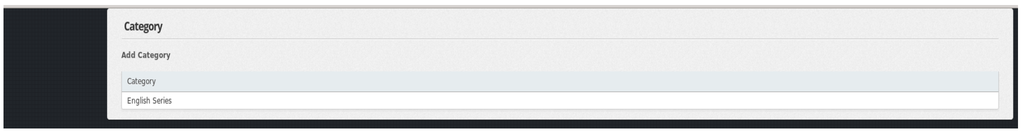
2. Data entry page for category appeared, “English Series” is typed as category, and submit.



Category Name: English Series

Figure 11. Data Entry for Category

3. Category is created and listed as shown below.



Category
English Series

Figure 12. Category Listing

The code for saving the category persistent object is given below.

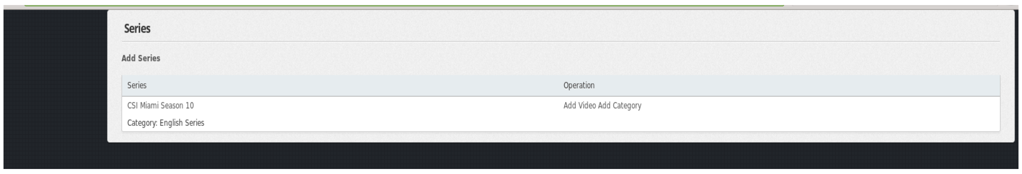
```
        String categoryName =
        req.getParameter("categoryName");
        if(categoryName == null)
                return;
```

```
F_Category cat = new F_Category();
cat.putData("Name", categoryName);
ObjectContainer db = F_DBUtil.getDB();
cat.save(db, true);
db.close();
```

In above, setting true to the method save generates a new identity for persistent object.

Series Listing

1. Series named “CSI Miami Season 10” is created and listed, and it is assigned to the category “English Series” previously saved.



Series

Add Series

Series	Operation
CSI Miami Season 10 Category: English Series	Add Video Add Category

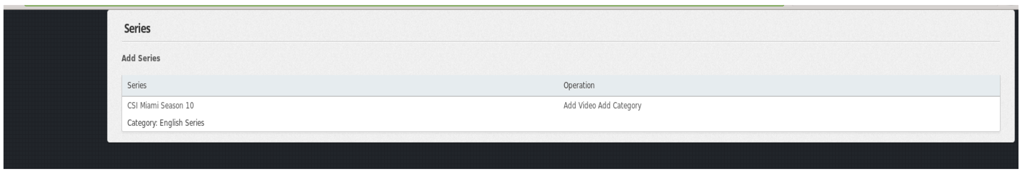
Figure 13. Series Listing

2. The code for saving the relationship of “Series” and “Category” is given below.

```
F_PO series = F_DBUtil.getPO(db, seriesId,
F_Series.class);
F_PO category = F_DBUtil.getPO(db, categoryId,
F_Category.class);
if(category == null || series == null)
        return;
F_CategorySeries_Relationship catSeriesRel = new
F_CategorySeries_Relationship();
catSeriesRel.setSource((F_POImpl)category);
catSeriesRel.setTarget((F_POImpl)series);
catSeriesRel.save(db, true);
series.addRelationship(catSeriesRel);
category.addRelationship(catSeriesRel);
series.save(db);
category.save(db);
db.close();
```

<u>Pull data from Youtube and Add Video to Series</u>

1. "Add Video" in the "Operation" column is clicked as shown in the below screen.



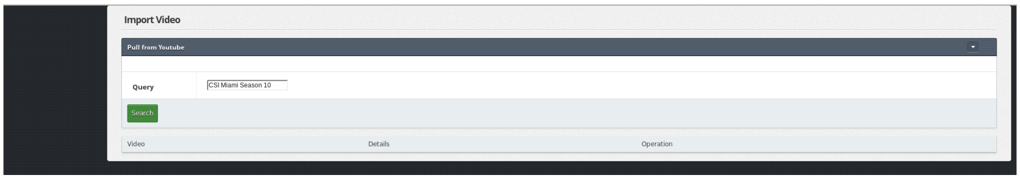
Series

Add Series

Series	Operation
CSI Miami Season 10 Category: English Series	Add Video Add Category

Figure 14. Add Series

2. "Import Video" screen appeared, and type in "CSI Miami Season 10" and click search.



Import Video

Pull from Youtube

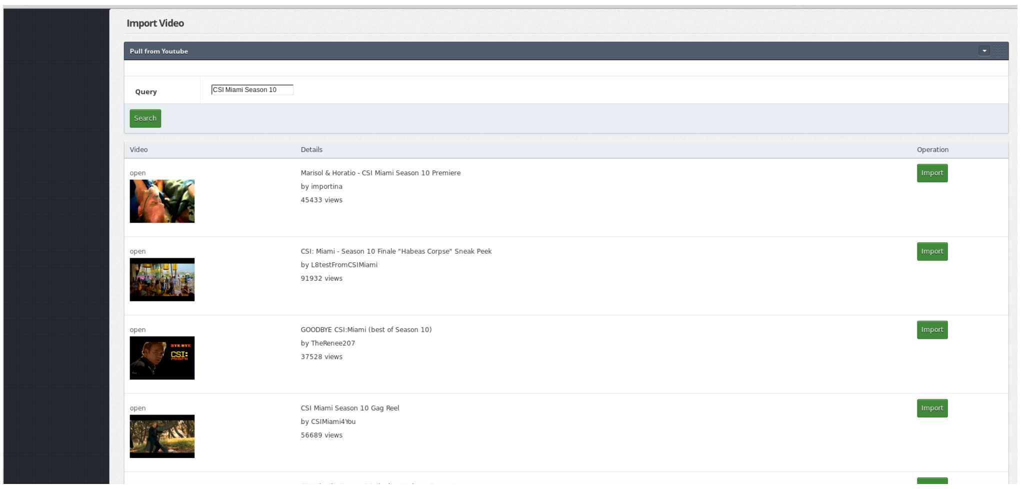
Query: CSI Miami Season 10

Search

Video	Details	Operation

Figure 15. Search Video from Youtube

3. It query the Youtube, returned results are processed for display as following.



Import Video

Pull from Youtube

Query: CSI Miami Season 10

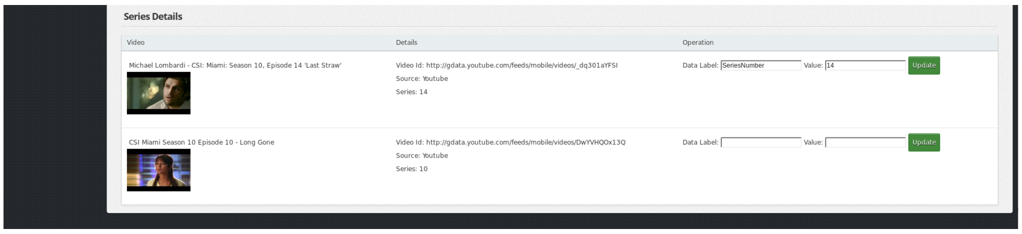
Search

Video	Details	Operation
open	Marisol & Horatio - CSI Miami Season 10 Premiere by importina 45433 views	Import
open	CSI: Miami - Season 10 Finale "Habeas Corpse" Sneak Peek by L8testFromCSIMiami 91932 views	Import
open	GOODBYE CSI:Miami (best of Season 10) by TheRenee207 37528 views	Import
open	CSI Miami Season 10 Gag Reel by CSIMiami4You 56689 views	Import

Figure 16. Return Results from Youtube

Relevant videos can be imported into the the database, it will create a series video persistent object for the selected video and create a relationship persistent object for binding itself with the series of “CSI Miami Season 10”. Both persistent object will be saved.

5. The details of series “CSI Miami Season 10” as shown below. It contained a video by the title “CSI Miami Season 10 Episode 10 – Long Gone” that is just added into.



Series Details

Video	Details	Operation
Michael Lombardi - CSI: Miami: Season 10, Episode 14 'Last Straw'	Video Id: http://gdata.youtube.com/feeds/mobile/videos/_dq301aYFSI Source: Youtube Series: 14	Data Label: SeriesNumber Value: 14 Update
CSI Miami Season 10 Episode 10 - Long Gone	Video Id: http://gdata.youtube.com/feeds/mobile/videos/DwYVHQOx13Q Source: Youtube Series: 10	Data Label: Value: Update

Figure 17. Series Details Screen

5.4 Evaluation

The simple system demonstrated is developed in less than a week, and it is fairly easy building system using the framework. It can cope with changes, as from time to time the system might decided to save some other attributes or metadata of video returned from Youtube, and it might need also the flexibility to integrate from various stream of feeds coming from different video sources with various format consolidated into a single persistent object type.

CHAPTER 6

DISCUSSIONS AND CONCLUSION

.1 Comparison of the Framework and Hibernate

Database design work

In the framework, there is no design work such as introducing table structure into the database, or design details pertaining to foreign key placement in certain table. Whereas in Hibernate, not only the works needed as a tradition of a relational database designing, it also needed to do a lot of configurations and setup. Given the framework in place, more attentions can be focused on the application, which is a good thing.

Data modeling

The framework can model any data as long as it can be reduced to a programming class. Both Hibernate and the framework can represent object, but Hibernate is restricted in the sense that some object representation cannot be mapped into the underlying relational representation in full or it is difficult to do so.

Generalization of Persistent Object

All persistent object are having similar structure by extending the persistent object class in the framework, they share same method such as getData(String label), it also allows specialization by adding data or method into the class for each subclass of persistent object. In Hibernate, persistent object is not generalized.

Identification

The framework allow each object to have uniqueness, uniqueness can be across all persistent object, across the same persistent object class, or across those belong to the same parent class, it is all decided by the designer. This is not the case with Hibernate as it follows relational manner where uniqueness is only across the same table.

Factorization of relationships

Any relationships existed in the framework can be factorized, as the user can retrieve the relationships by giving the relationship class to the framework respective method, whereas in Hibernate, is following the relational manner, relationships cannot be fully factorized out.

	The Framework	Hibernate
Database Design Work	No actual work, only conceptual understanding	Large amount
Need Names, Types for Columns and etc...	No	Yes, need to provides specifications such as length/size of data as well
Object-Relational Mapping	No	Work needed
Data Modeling	Represent Object in fullness	Restricted
Generalization of Persistent Object	Yes	No
Identification	More choices offer to designer	Key based
Relationship Modelling	Yes	Yes
Factorization of Relationship	Yes	No
Joining Multiple Relationships	Not implemented, may be difficult	Fully enabled but associated with cost
Record Column	Variable length and flexible to	Fixed as table columns are fixed

Size	add or remove anytime needed	
Provides Common Persistent Operations	Yes	Yes
Support Integration with Rapid Development Tools	Because of its simplicity, it can easily support the tool	Difficult
Performance	Not tested, but believed to be better as it has less computation	Not tested

Table 1. Comparisons of Hibernate with the Framework.

6.2 Conclusion

The object database of DB4O is studied. Anomaly is found in the experiments where deletion of object leaves side-effects on the collection type object's data. The framework is developed. A design strategy is suggested to lower burden of the type knowledge in advance in using the framework. The framework does not yet support multiple relationship retrieving similar to JOIN in relational database. More future work is expected to make the framework better in many areas.

REFERENCES

1. [KC] S. Khoshafian and G.P. Copeland (September 1986). Object Identity. *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications, Portland.*
2. [BM] F. Bancilhon and D. Maier (1988). Multilanguage Object-Oriented Systems: New Answer to Old Database Problems? *Future Generation Computer II. K. Fuchi and L. Kott eds. North Holland, Amsterdam.*
3. A.Snyder (September 1986). Encapsulation and Inheritance in Object-Oriented Programming Languuages. *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications, Portland.*
4. T.M. Atwood (1985). An Object-Oriented DBMS for Design Support Applications. *Proceedings IEEE CoMPPINT 85.*
5. [CODD] E. F. Codd (June 1970). A Relational Model Of Data For Large Shared Data Banks, *Communications of the ACM, Vol. 13, No. 6.*
6. E.F. Codd (December 1979). Extending the Database Relational Model To Capture More Meaning, *Transactions On Database Systems, ACM, Vol. 4, No.*
7. M.M. Zloof (May 1975). Query By Example. *Proceedings of the NCC, AFIPS Press, Montvale, N.J.*
8. D. H. Fishman, et al. IRIS, 1987. An Object-Oriented Database Management System. *ACM Transactions on Office Information Systems, 5(1).*
9. Ambler, S. W. (1997). Mapping objects to relational databases, in building object applications that work. *SIGS Books.*