On Object-Oriented Data Model for Rapid Application Development

CS 691 – R&D Project Report

by

Prathab K
Roll No: 06329902

Under the guidance of
Prof. Deepak B. Phatak

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
November, 2007

Acknowledgments

I especially want to thank **Prof. Deepak B. Phatak**, for his guidance to work on this novel idea with his constant motivation and encouragement to convert the ideas into a quality software framework. I would like to express my gratitude to the **ADA lab** team members for their support and the computational facilities provided for the framework testing. I would like to appreciate the **Ekalavya team** members in assisting the case study and to shape the framework to cater the real needs.

Prathab K
M.Tech, Second Year
KReSIT,
IIT Bombay

Abstract

Modeling the application into tiers, allows developers to create a flexible and reusable application. But the problem arises as the same data occupies different data structures in each tier and thereby creating the impedance mismatches in the data flow. These impedance mismatches hinders the scalable and reusable application development and also consumes more development and maintenance cost. This demands a single way representation of data or unique way of accessing the data to make a seamless development experience. This report focuses on the object-oriented data model to overcome the impedance mismatches. The object-oriented design become so ubiquitous that it is unlikely any developer isn't using some object-oriented programming language dialect. An effective object-oriented data model is proposed for object-oriented applications, which will provide a seamless application development experience using objects.

Table of Contents

1.0 Introduction

1.1 Motivation

The wide range of software frameworks and tools enables rapid and smooth software development. But, maintenance of the developed software code becomes costly, since developer who needs to maintain the code requires knowledge on the frameworks and tools used. Auto-generated code by the tools and framework finds its own issues during software maintenance. Using several technologies or frameworks makes the information flow in the software more complex, creating an impedance mismatch, a representation of same data as different data structures, at respective layers in the system. Consider the scenario of web-application creation with RDBMS as a backend, Object-oriented programming as middle tier and a scripting programming language to render the presentation. In this scenario one can see the impedance mismatch of data in terms of tables, objects, and tags. To overcome this problem, mapping tools are used which eventually adds the development and maintenance cost. There is also impedance mismatch in developers thinking apart from the information impedance mismatch which plays a crucial role in software quality. Though all these issues can be well-addressed by disciplined software development process, it consumes high development cost and human resources.

Emphasis on software quality and reusability allows for abstraction, modularization, encapsulation, and distribution of units of program code. With the prevalent of impedance mismatch in information flow and developers thinking does not allow in reaching the goal especially with limited development bandwidth and developer resources.

This motivates us to develop a framework focusing on the object-oriented data model and leveraging the object-oriented capabilities to build scalable applications rapidly by overcoming the impedance mismatches. This report describes how data can be modeled effectively in term objects and how it aids in the application development.

1.2 Outline

This report is organized with brief introduction on impedance mismatches in section 2, which briefly describes on object-relation and model-implementation impedance mismatch. In section 3, the proposed object-oriented data model is described and details how the database abstraction is handled. Also, it details about object types and how the object relationships are captured. The section 4 describes on the rapid application development using the proposed object-oriented data model. Section 5 concludes the insights on object model and the future work on the proposed model. Appendix provides screenshots of object creation with code snippets and performance comparison graphs on object database.

2.0 Impedance Mismatches

In software context, the term ‘impedance mismatch’ usually refers to the data flow across the tiers where the mismatches arises in representation of same data as different data structure in each tier. This usually caused by mapping the data to different data structure to leverage the information at each layer. Also, the impedance mismatches even present between analysis and implementation environments. Developers does not pay much attention to these impedance mismatches and usually overcome the problems by employing appropriate mapping tools, which adds up to the development and maintenance cost.

2.1 Object-Relational Impedance Mismatch

The object-relation impedance mismatch is set of conceptual and programming difficulties, encountered in using a Relation Database Management Systems (RDBMS) as the persistence storage and object-oriented programming language to access the persistence data. More precisely, the class definitions are mapped in a straightforward way to databases relational schemata. The mismatch arises when the developer attempts to implement the object-oriented programming concepts for classes of objects, inheritance, and polymorphism, which are not supported by relational database systems.

The object-relational mapping tools like Hibernate, provides an impedance matching to this problem. However, from the schema evolution perspective, it incurs the maintenance cost as the developer needs to aware what are changes to be done on both sides (tables and classes) to keep the functionality intact.

2.2 Conceptual Model-Implementation Impedance Mismatch

Apart from the major object-relation impedance mismatch, there exists impedance mismatch between conceptual models and implementation environments [1]. This mismatch occurs as application often structured into layers of tiers and each tier has its own data model with its own way of accessing it. For example, a web-application typically structured into three tiers as the presentation tier, the middle or business tier, and the data tier. The presentation tier usually deals with mark up languages like HTML, XML, or scripting languages like JSP, ASP. The middle tier usually leverages the advantage of some object-oriented language. And, the data tier deals with storing the data in a relation database. The data representation switches in each tier and the way of accessing it too. To bridge the gap between the conceptual models and implementation environment, requires a thought on single representation of data, or as unified data model, or with unique way of accessing the different representation of data.

3.0 On Object-Oriented Data Model

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 develop the middle and presentation tier in terms of objects is described in this section.

3.1 Database Abstraction

3.1.1 Object-Oriented Database

Object-oriented database enables to represent the information in the form of objects. With the database management principles and object-oriented programming capabilities, forms an object-oriented database management system (OODBMS). Object-oriented database bridges the data modeling and programming in an object-oriented environment, thereby averting the object-relational impedance mismatch. Object database permits developers to model complex data easily and capture the relationships in a natural way.

We used db4o [2], an open source object database, as the OODBMS. We have created a fine abstraction around the object database to provide transparent persistence operations, by abstracting the data as object types. The architectural stack diagram represented below provides a generic view in modeling the application in terms of objects.

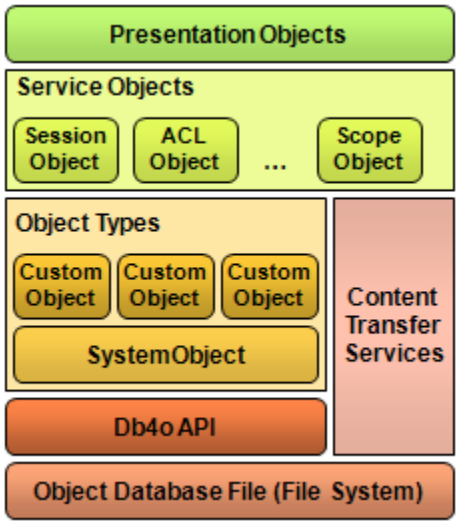

Figure 1: Architectural Stack Diagram for Object-Oriented Data Model

3.1.2 Object Types

The data are modeled as object types. An object type represents a class with associated attributes and the meta-data like relationships, constraints, etc., of the class. So the system or application is viewed in terms of object types and its relationships. Abstracting

the modeling in terms of object types makes the developers thought process easier and allows to capture the meta-data information while data modeling itself.

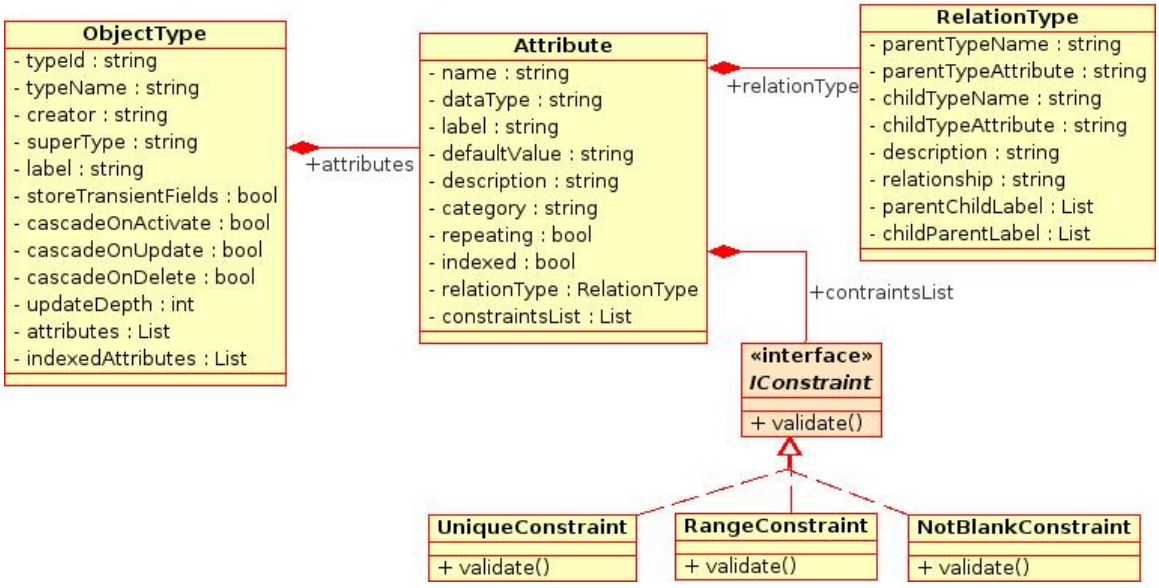

Figure 2: Object Type Class Diagram

Every object type should inherit the parent object type 'SystemObject' which contains the system related attributes like object id, creation and modification information, etc., and methods to store and retrieve the data. By inheriting the SystemObject type, objects will have a seamless storage and retrieval operations. Inheritance of object type allows evolution of types easier and increase the application scalability. See Appendix-A on object type creation along with storage and retrieval of objects.

3.2 Capturing Object Relationships

An object-oriented system can be characterized as a system of cooperation objects. Capturing object relationships provides the semantic information of the data. Capturing relationships among the objects makes the application more adaptive, intelligent, and effective. The object relationships are categorized based on the degree of sharing and degree of life time dependency [3]. The table below represents how the object relationships are modeled in object-oriented programming level.

Relationship Type	Object Type Modeling
One-to-one	Declaring as attribute of the type *Eg: Customer has Invoice*
One-to-many	Declaring as List/Set data structure to model the ‘many’ association *Eg: Group contains Users*
One-to-self	Declaring as List/Set data structure to model the ‘self’ association *Eg: Group contains (other) Groups*
Many-to-many	Declaring as List/Set data structure (in both types) to model the ‘many’ association *Eg: Employee associated with different Projects, Projects contains Employees*
Owned (Composition)	Declaring as the respective object type (having life time dependency)
Referenced (Aggregation)	Referencing the respective object type (with their referential identity)

The relationships are captured during the object type creation process itself. The ‘Relational Type’ object has the respective attributes to capture the information about parent and child type. User can associate labels to indicate the relationships between the parent-child and child-parent type. See Appendix-A on associating relationships between object types.

4.0 Rapid Application Development

The object-oriented data model enables to build applications rapidly since data model plays an important role in application development. We integrated the object type creation and modification with eclipse IDE, which cuts down the development time, involved in creating and maintaining the data objects of the application. Every data object

extends 'SystemObject' which encapsulates the persistence operations and there by providing a seamless database abstraction. Once the developer creates the data objects it is ready for use. This model aids the iterative development process and the evolving object type supports the application scalability. Developers can model in terms of objects matching the impedance across the application layers and delivering a quality product in stipulated duration.

The object-oriented model allows room for reusability and component based development. Also, capturing the relationships during the object creation allows to construction of ontological objects. Developers need to focus on the data and its relationships to build the application. The in-built service and presentation objects aid the rapid application development by leveraging the meta-data of object types.

5.0 Conclusion and Future Work

The adoption of an object-oriented data model allows designing and developing application rapidly. It reduces the development cost by not having to concern about separate data models and also less coding is involved due to the lack of impedance mismatches. The object model provides pluggable architecture for application development and increases the scope for reusability. Though the proposed object-oriented model helps to model the applications in a natural way, one has to apply the object-oriented design principles properly to utilize the model to the fullest in achieving the product quality.

The future work involves in creating the core service objects like session objects, access control list (acl) objects, etc., and presentation objects like wizard containers, dialog container, etc., thereby one can prototype and develop the application rapidly. To increase the reusability, the framework will support the notion of components, where developers have flexibility in extending and overriding the components to build a quality product.

6.0 Appendix

A – Object Type Creation

The object type creation and modification is integrated with eclipse IDE as a plug-in. Developer can create the object types using the respective wizard and the corresponding entity class will be auto generated. The following screen shots shows the process of object type creation with attributes and its relationship.

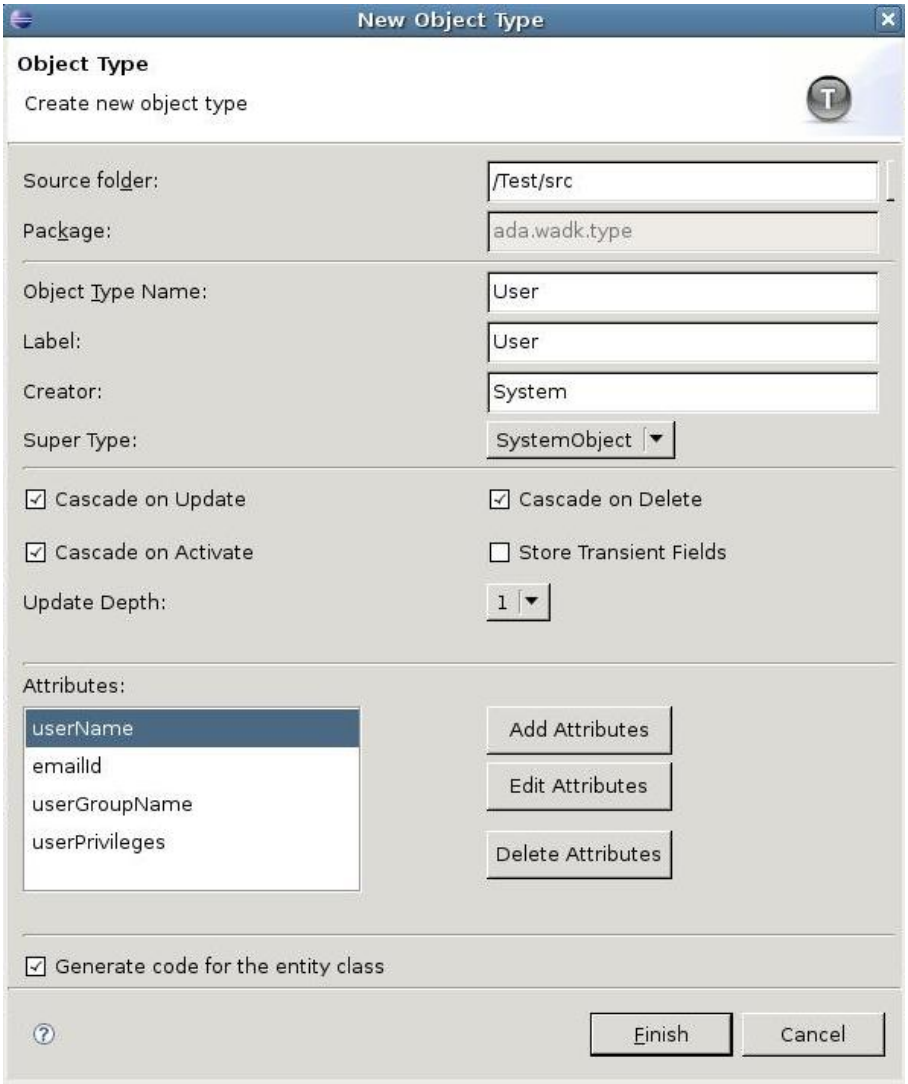

Figure 3: Object Type Creation Wizard - creating new object type

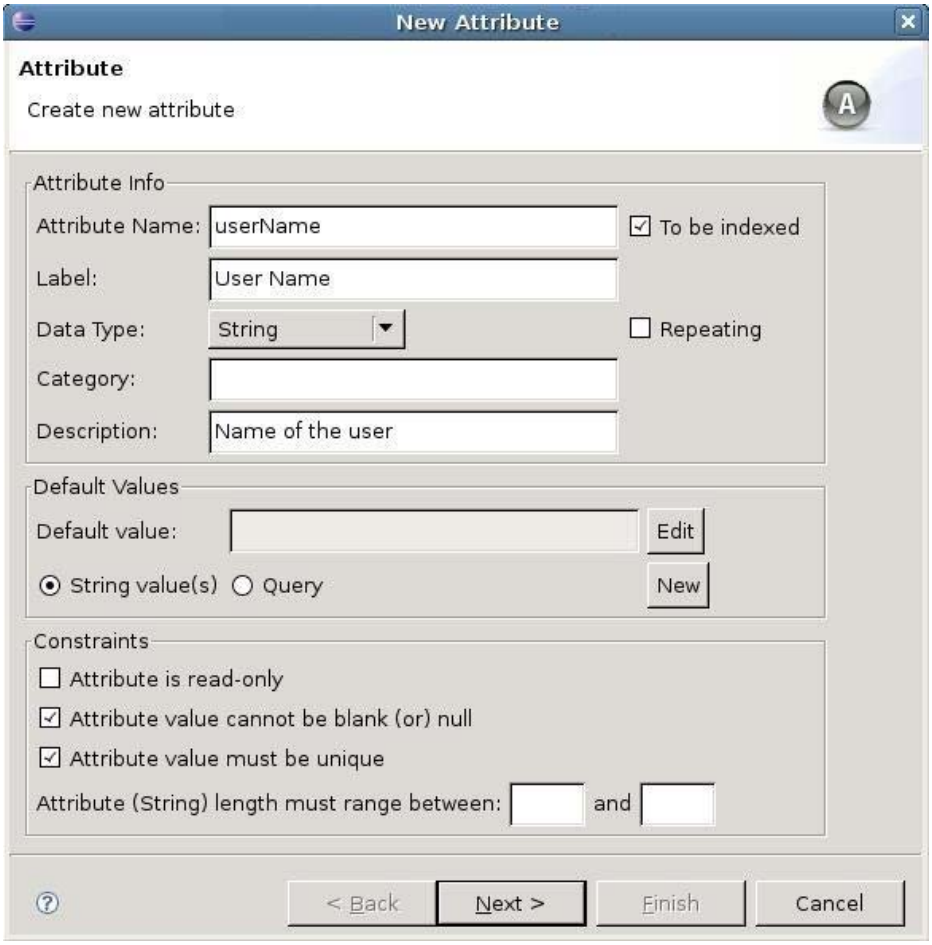

Figure 4: Object Type Creation Wizard - creating new attribute

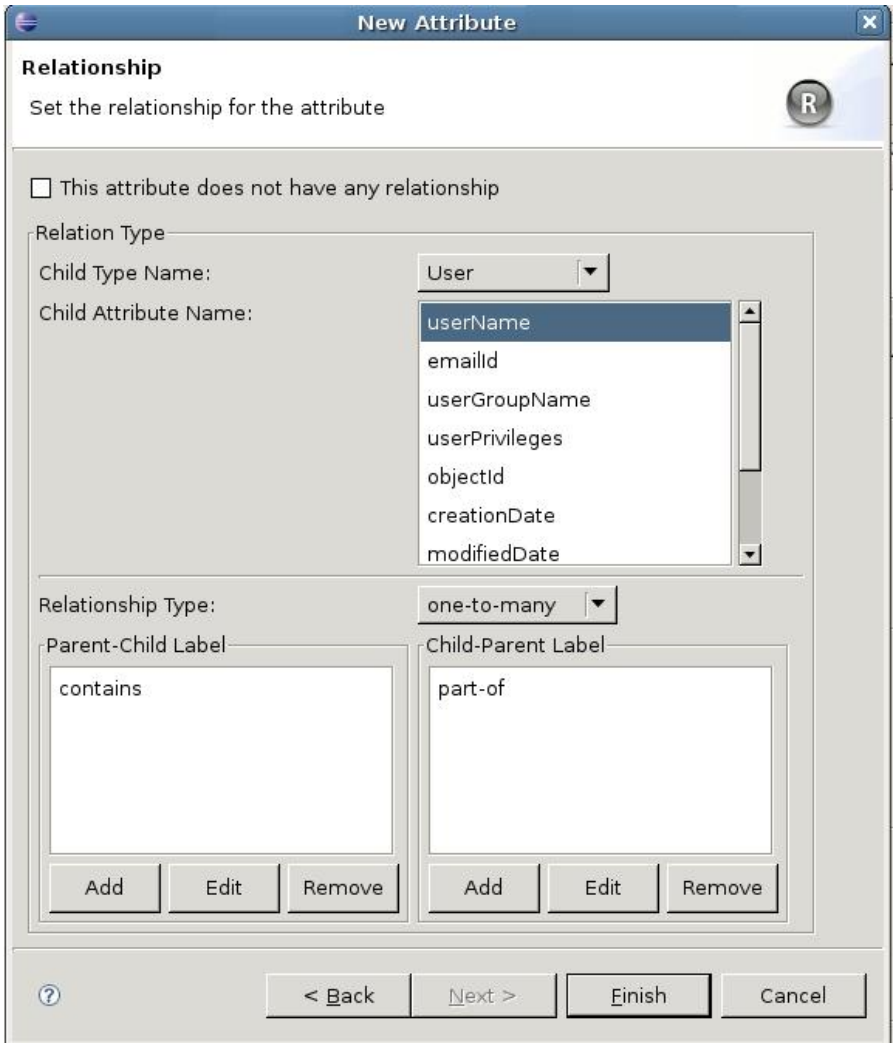

Figure 5: Object Type Creation Wizard - associating relationships with attribute

The following code snippet shows how to write, read and query the objects.

```
//writing a object to OO database
User user = new User();
user.setUserName("User-1");
user.setUserGroupName("default group");
user.setUserPrivileges("default acl");
user.setEmailId("user-1@mail.com");
Result result = user.save();
if(result.Success)
     System.out.println("User object is saved successfully");
else
```

```
     System.out.println(result.ErrMessage);

//read the object from OO database
User user = new User();
user.setUserName("User-1");
user = (User) user.firstMatch();

//query the object from OO database
Query query = Db4oAdapter.getObjectContainer().query();
query.constrain(User.class);
query.descend("userName").constrain("User-1");
ObjectSet objSet = query.execute();
if(objSet.hasNext())
     User user = (User) objSet.next();

//delete the object from OO database
User user = new User();
user.setUserName("User-1");
user.delete();
```

B – Object Database Performance Comparison

The object database used in the proposed object data model is db4o (Database for objects), an open source object database. Db4o displays significant performance advantages, especially when it comes to storing complex object structures [4]. The following graphs represents the comparison of response time to create, update, retrieve and delete objects using db4o as object database and with Hibernate/PostgreSQL as the backend.

Db4o shows a better performance than mapping the PostgreSQL with Hibernate in writing and querying the objects. The graph below shows the response time in performing various operations using the proposed data model.

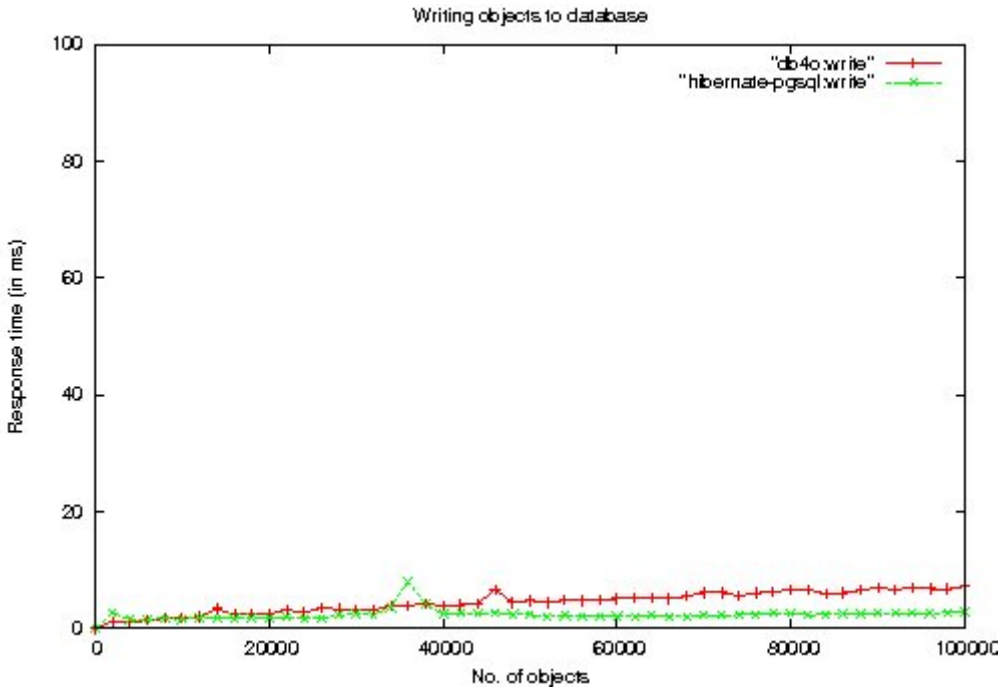

Figure 6: Response time in writing the objects to database

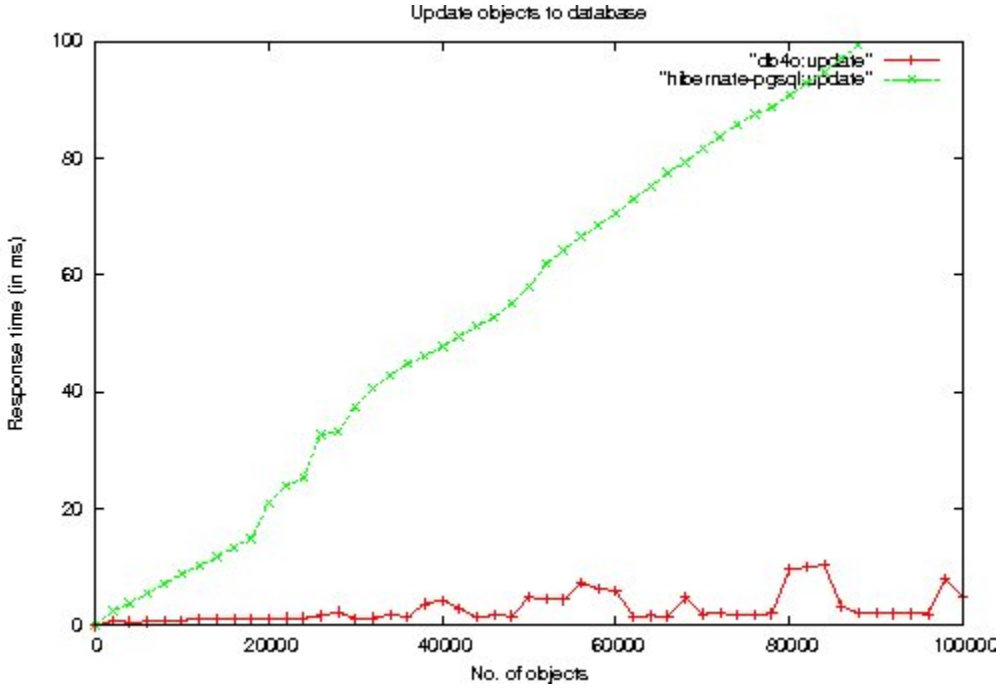

Figure 7: Response time in update of the objects to database

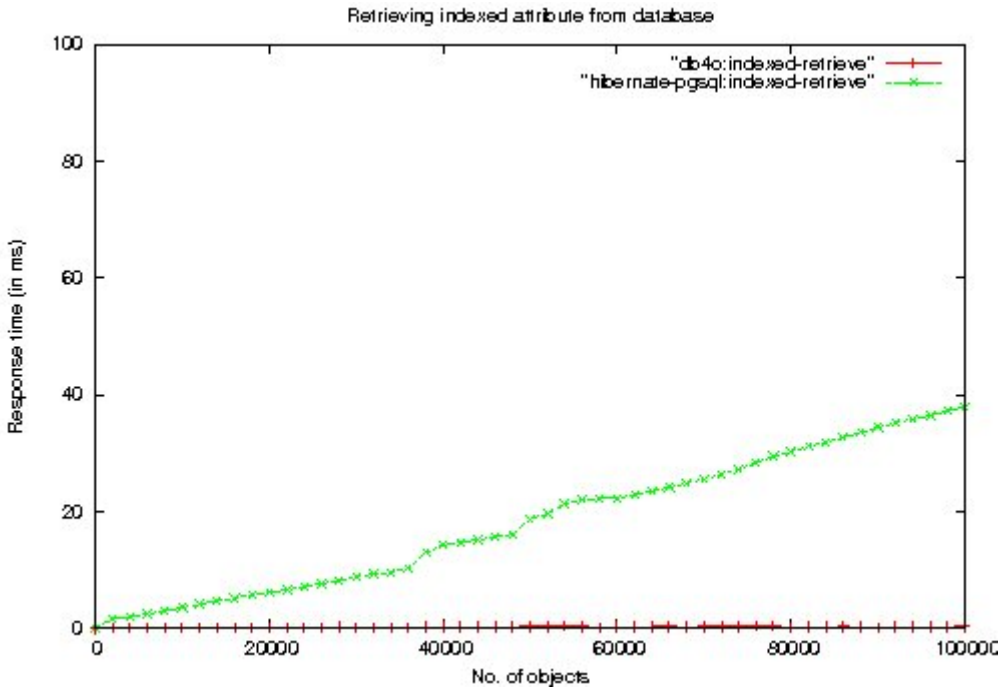

Figure 8: Response time in retrieving the indexed attribute value from database

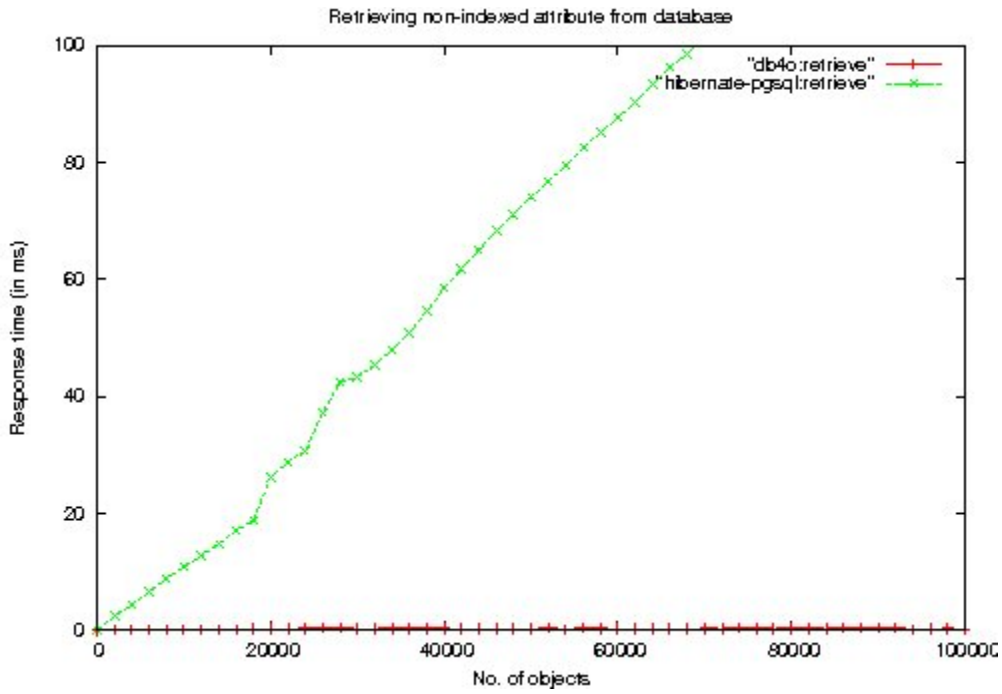

Figure 9: Response time in retrieving the non-indexed attribute value from database

7.0 References

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

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

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

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