1. Introduction

This project report details the development of an object persistence framework to work with different storage mechanisms. It offers object persistence services to the user application while shielding the user from details of the storage mechanism used. It is also extendable, allowing the user to customize it to suit a particular domain, or to support various other storage types.

1.1 Background

With current technologies leaning toward object-oriented concepts, there exists a need for these technologies to interact with older legacy systems as a large amount of data are still stored in these systems. Legacy systems typically use relational databases, which are based on the relational model and store data in the form of tables and records with fixed data types.

Although this model is sufficient for data storage, it is inadequate for storing objects, which have a user-extensible type system, encapsulation, inheritance, dynamic binding of methods, complex and composite objects, all of which are partly or entirely unsupported by legacy relational databases. An application should be able to store an object into a relational database and later retrieve it in its original form. Thus, the object is said to be persistent, i.e. it lasts beyond the execution lifetime of the application.

Therefore, to interoperate with legacy systems, some sort of translation needs to be performed between the object model and the relational model, and vice-versa. Object-relational mapping is the process of transforming between object and relational modeling approaches and between the systems that support these approaches.

This mapping is done by the persistence framework, which is a set of components that can be used by the user application to save and restore objects. The framework is responsible for converting the object into a format suitable for storage in a persistence mechanism during saving, and converting the data back into an object during retrieval. A persistence mechanism is simply any form of permanent, non-volatile storage that can
store and restore data back to its original form. This includes relational databases, object databases, filesystems, e-mail servers, web servers, and others.

Changing completely to object-oriented systems is not always a feasible solution due to time and costs involved. As such, a persistence framework acts as an intermediate layer between these two systems.

1.2 Motivation

Currently, there exists various products that are able to offer persistence services for objects in various programming languages. However, most of them are not transparent – they require some form of user intervention in the form of pre- or post-processing of the class definitions, or declaring the class schema to the framework during run-time. This means that not all objects can be persisted, only those that have been specially processed. Moreover, most of these frameworks are not made to be easily extendable by the user, support relatively few persistence mechanisms, or are tied to a proprietary mechanism.

Also, due to the relative youth of object-oriented databases, it is often desirable to store objects in a relational database, which more often than not, have been tested extensively in real world environments and successfully refined over the years to be a stable and reliable mechanism for data storage.

1.3 Problem Statement

The problems addressed in this project are:

- **Mapping the object model to the relational model and vice versa.**
  This is solved by mapping classes to tables, objects to records, and attributes to columns in a relational database. Inheritance trees are flattened out, and relationships are stored as foreign keys.

- **Implementing object persistence while maintaining transparency.**
  This is solved by utilizing object reflection, allowing the persistence framework to access an object’s attributes without it being declared explicitly.
• Designing a reusable persistence framework.
The framework is based on an open design with a layered architecture, allowing the user to extend the framework’s functionality and support additional persistence mechanisms.

1.4 Objectives

It is the goal of this project to develop a persistence framework with the following primary features:

• **Transparency** – the framework should shield the user from the details of the persistence mechanism, and the process of saving and restoring should be as transparent as possible.
• **Ease of use** – the framework should be easy to use, requiring the user to learn only a few functions. The persistence process should be as automated as possible.
• **Supports various mechanisms** – the framework should support various different types of persistence mechanisms.
• **Reusable** – the framework should be extendable by the user to increase functionality or support for persistence mechanisms.
• **Queries** – the framework should support querying of objects by the user based on criteria on object attributes. All objects satisfying the criteria are then returned to the user.

1.5 Results

The persistence framework was found to be successful in offering object persistence services, achieving all of the objectives outlined in Section 1.4. The primary contribution of this project is a transparent and reusable persistence framework.

1.6 Chapter Organization

The rest of this report is organized into the following chapters:
Chapter 2 introduces the concept of object persistence, object-to-relational mappings, and frameworks. It also reviews some of the current and past work done in this area.

Chapter 3 describes the architecture of this project, in terms of the layered model and the framework's role in the user application.

Chapter 4 presents the use case model and analysis model of the framework. The use case model captures the requirements of the system while the analysis model describes the high-level design and object interactions.

Chapter 5 is the detailed design model. It contains a detailed description of the class diagrams, attributes, methods, and algorithms. The reusability of the framework is also covered.

Chapter 6 discusses the implementation and any issues associated with it.

Chapter 7 presents the results of actual testing done on the finished persistence framework as well as the test cases.

The notation used in modeling follows the Unified Modeling Language [25] and the reuse model described in [24].