

# Towards a Layered Architectural Design of a Persistence Framework

Sai Peck Lee and Tong Ming Lim  
Department of Software Engineering  
Faculty of Computer Science & Information  
Technology  
University of Malaya  
50603 Kuala Lumpur, Malaysia

Ho-Jin Choi  
School of Engineering  
Information and Communications University  
119, Moonji-ro, Yuseong-gu, Daejeon  
305-714, Korea

*Abstract - Persistence middleware enabling objects to be persistent in persistence mechanisms have emerged in this rapidly evolving software market. However, problems prevail in terms of non-uniformity for programmers in making use of functions of the persistence service interface provided by these middleware in their programs for object persistence in a target persistent store. This stems from the issue of coupling between the domain objects and underlying details of persistence mechanisms in the source code of programs which suffers in terms of seamless mapping, or the need to introduce an additional view for programmers to specify their domain objects in order to simulate transparent persistence. This paper intends to discuss pertinent issues in persistence framework development focusing on the architectural design of Extolware, a persistence framework prototype developed within our research program, to achieve the objective of seamlessness and uniformity of persistence service interface besides providing a single object-oriented view to programmers.*

**Keywords: Persistence Framework, Persistence Middleware, Persistence Service, Seamlessness, Layered Architectural Design.**

## 1.0 Introduction

Software systems with sophisticated graphical user interface, multimedia management and control intensive requirements are increasingly becoming an integral part of modern advanced systems. In order to design and develop such software systems which are basically complex and semantically rich, database management systems based on object-oriented (OO) approach, which in the last decade was deemed more appropriate, have been proposed and developed. However, penetration of OO database management systems has been slow due to lack of support from large database vendors and of a sound mathematical foundation. Nevertheless, several OO database management systems have walked out of the research lab to the commercial market with products like UniSQL and Objectivity/DB. However, issues concerning the abovementioned problems have greatly hindered the acceptance of OO databases.

It is evident that a large number of existing applications and legacy databases have been implemented using relational and multidimensional database systems. For the purpose of enhancement in order to cater for current complex requirements, it is expensive, and in some way, risky for these applications and legacy systems to be reinvested into a new and not-so-widely accepted OO database technology. As a result, it is essential for researchers to closely examine relevant features of object-orientation, at the same time finding ways to leverage and exploit powerful supporting features of existing persistence mechanisms in order to lead to an alternative, acceptable and more consolidated persistence solution.

Within a span of less than 10 years, middleware solutions of object-to-table persistence framework have emerged and implemented with the purpose to provide persistence services, including materialization and dematerialization or mapping between objects and legacy databases such as relational and multidimensional databases, as well as a way to promote migration of conventional database technologies to OO database technology.

This paper intends to discuss pertinent issues in persistence framework development focusing on the architectural design of Extolware [7], a persistence framework prototype developed within our research program, to achieve the objective of uniformity of persistence service interface as well as a single OO view to developers, which would greatly help in ease of use and maintenance of application programs.

The background on persistence framework and a general discussion on the issues in persistence framework design and development will be given in Section 2. We will then proceed to discuss our motivation in Section 3. An overview of Extolware's architectural components is given in Section 4. Some discussions are given in Section 5. Finally, a conclusion is given in Section 6.

## 2.0 Background on Persistence Framework

A persistence framework provides an architecture serving as a template for software developers to adapt, extend and enhance within shortest time possible so that minimum programming and changes are required to allow objects to be saved, retrieved and manipulated in a target persistent store. Ambler [1] presents three most commonly used approaches to the management of persistent objects in practice.

A persistence framework must be robust and extensible with minimum coupling underlying a domain application in order to allow flexible enhancements and ease of adaptability. An extensible and robust framework that forms the persistence layer must allow ease of mounting with virtually all types of persistence mechanisms like relational management database system (RDBMS), flat file system, multidimensional management database system (MDBMS), even object-relational database management system (ORDBMS) and OO database management system (OODBMS). Both ORDBMS and OODBMS must be able to be mounted easily despite their differences in design and implementation. Besides that, it must also allow database systems from multiple vendors to be easily mounted, as well as both native and non-native database drivers used must be able to connect to their corresponding database systems. In order to make a persistence layer versatile and flexible, multiple database connection sessions must be permitted.

A well-designed persistence framework has a set of robust, consistent and extensible interfaces. All persistence service related functionalities and details are fully encapsulated. To trigger an operation, a message is used to activate the appropriate function in the persistence layer. Changing different databases, modifying domain class schemas and enhancing classes should be transparent to users at all time. Besides that, single and multiple-object retrievals must be supported. A multiple-object select operation produces a set of objects for purposes such as report printing, print preview and objects navigation. Some real-world applications such as multimedia editing or computer-aided engineering drawings take many hours for completing editing or drawing. These activities require long-duration management capability. Having such a characteristic is one of the requirements of a robust persistence layer.

Most features in the object-oriented model are recursive in nature. When an object loads some other related objects into the memory, the mechanism such as Lazy Object Relations is used [2] so that objects loaded into the working memory do not consume all the memory resources which may cause performance degradation.

An important feature is to allow ease of maintenance for domain classes as domain requirements change over time. This involves modifying domain classes, hence restructuring the data structure of the underlying persistence mechanism. Some degree of administrative facilities is necessary so that developers do not need to modify both domain classes and table structures manually. The design of the physical database model and mapping of objects heavily influence the performance of an object-relational access layer. Some tables favor read operation, some favor update and write operations, while others may favor a mix of these operations. Hence, the design of tables in a database could either improve or deteriorate the performance of the access layer [2].

Object identifier is the key to traverse in an OO database system. It must be unique and universal so that objects within a standalone server, client/server environment or distributed database environment can be easily and uniquely identified. Cursor is another important feature in a RDBMS or MDBMS environment. It allows the requested records to be grouped as a logical unit and then sent to an interested party. A persistence layer must support this feature in order not to overload the network by sending the requested objects in a highly uncontrolled fashion. Other than this, proxy is another useful feature which represents the identifiers of a group of interested objects to be sent to a requested party so that minimum overhead is imposed on the network and database server. Chaudhri [2] used the concept of smart-pointer to deliver the same service so that the framework does not perform poorly. As a required object is chosen, the actual object is then loaded into the working memory.

Some persistence frameworks support n-tier client/server architecture between clients and servers. This feature allows a persistence layer to be employed in a much complicated computing environment such as enterprise computing or multi-site distributed database environment that supports critical applications.

Finally, definition and query languages such as Structured Query Language (SQL) or Object Query Language (OQL) are additional features that should be supported so that both developers and end users are able to interact with the database systems through the persistence layer easily and effortlessly. Reporting tools today are still supporting objects that are structured into flat or un-nested record structure. This is not a 'must have' feature, but with this feature, many reporting tools are handy to users.

## 3.0 Problematic Issues and Motivation

OO development environments have become very popular among software developers these days because of their large repository of reusable classes and productive development environment that could lead to a shorter software development duration and higher application quality. In addition, it is easier to extend dynamic application logic to allow changes of domain requirements to be easily incorporated into an existing application system. Hence, OO Programming Languages (OOPL) and OO programming environments (OOPE) are becoming the chosen platform used by today's developers. Modelling languages such as Unified Modelling Language, which is a widely accepted OO

modelling language nowadays, provides a standard communication notation that enables large and complex OO application systems to be modelled quickly and consistently. Nevertheless, OOPL such as C++, Smalltalk and Java as well as rapid application development environments do not provide a suitable interface that allows objects in an application to be saved into a persistence mechanism seamlessly. Even though these languages and tools provide libraries (both native and non-native) and Application Programming Interfaces (API) (standard and proprietary) such as ODBC and JDBC to save data and retrieve data from a RDBMS or MDBMS, these middleware require the application kernel to construct query statements within the application to perform these operations. As application requirements evolve, these query statements must follow suit. The maintenance overhead is very high and productivity is low.

In order to take advantage of OO technologies and existing widely-accepted database systems, most applications are developed using a variety of OOPL and OOPE. Embedded SQL statements are the primary approach used to save and retrieve data from the persistence mechanism. This approach is appropriate for small applications that do not change domain requirements dynamically. However, for large, dynamic and complex domain applications, embedded SQL statements are not appropriate simply because domain classes containing application domain logic, SQL statements in the application kernel and the persistent store are too tightly coupled together. Hence, maintaining these applications from the programmer's perspective and keeping them up-to-date in order to adapt to changes in domain requirements, at the same time allowing a flexible selection of different types of persistence mechanisms, always incur a very high project cost. Furthermore, the stability of the application system is also being jeopardised.

Though efforts, either through a consortium or individually, had been undertaken to define a standard OO definition and query language for OO databases to facilitate object persistence and manipulation, these efforts did not successfully convince the database community or generate their interest to migrate to OO databases, taking into the fact of the powerful features and potential issues that OO databases can handle. Many database vendors realize that migrating from a widely accepted database management system to an OODBMS usually takes a long period of time. Furthermore, issues such as technology maturity, lack of support from major database vendors, lack of standard, no sound mathematical foundation and lack of rigorous research activities, are among a few factors that slow down the acceptance of OODBMS. As a result, persistence middleware and software products that enable objects to be persistent in persistence mechanisms have emerged in this rapidly evolving and competitive information technology market.

Software products such as object wrappers (Object-to-RDBMS or Object-to-MDBMS) middleware and ORDBMS have become the intermediate solutions for developers to manage objects in the memory and persistent stores. These products allow business applications developed using in an OOPL or OOPE to have their investment protected in terms of development time, money and effort. In addition, as OODBMS technology matures, these business applications could easily be migrated to an OO database platform. Hence, a robust, flexible and adaptable persistence framework is not easy to design. Factors that must be considered are ease of migration between different database systems, framework robustness, efficiency of mapping algorithms and standard compliances, as well as fundamental architectural design principles such as seamlessness and persistence transparency which must be reflected in its persistence service interface.

## 4.0 Architectural Components of Extolware

In this section, we present the architectural design of Extolware, a persistence framework prototype [6] developed within our research program, to achieve the objective of seamlessness and uniformity of persistence service interface besides providing a single object-oriented view to programmers.

Extolware [7] utilises a layered architectural design pattern in its architecture. Each layer in the framework is responsible for a unique task. The layer above provides services to the layer below. Layered pattern allows each layer to be modified and enhanced independently without interfering services provided by the layers above or below it. This is true as long as the interfaces defined by each layer are unchanged. Fig. 1 gives a conceptual architectural view of Extolware with its architectural components such as EODL Parser, EOQL Parser, Object-to-Legacy Data Structures Mapping Engine (or Mapper), Persistent Object Management System (POMS), Active Meta Class Expert, EOQL-Persistent Object Mapper, Objects Navigator, Two-tier XML-based Cache Manager, Object-based Two-Phase Transaction Management System and Wrapper Utility.

Object Management Facility (OMF) is a graphical-based tool that consists of Extended Object Management Tool (EOMT) and Object Schema Management Tool (OSMT) [4], forming the top-most layer of the framework. OSMT is used to define, delete, update and manage classes of a domain. All class, type, interface, constant and method definitions are stored and managed by OSMT as EODL statements. Each set of EODL statements defines classes of a domain in a module. Each module is stored as a file. Ease of use and ease of maintenance are the ultimate objective of this tool. EOMT is also designed for the same reason; it is easy to use and virtually no learning effort is required when select, insert, delete and update operations need to be carried out. It displays results returned from executing a database operation. If it is a select operation, the objects returned will be shown in a tree hierarchy. Data Service On-Demand [3] is used to subsequently display objects in an inheritance hierarchy as well as in complex objects, collection and relationship attributes.

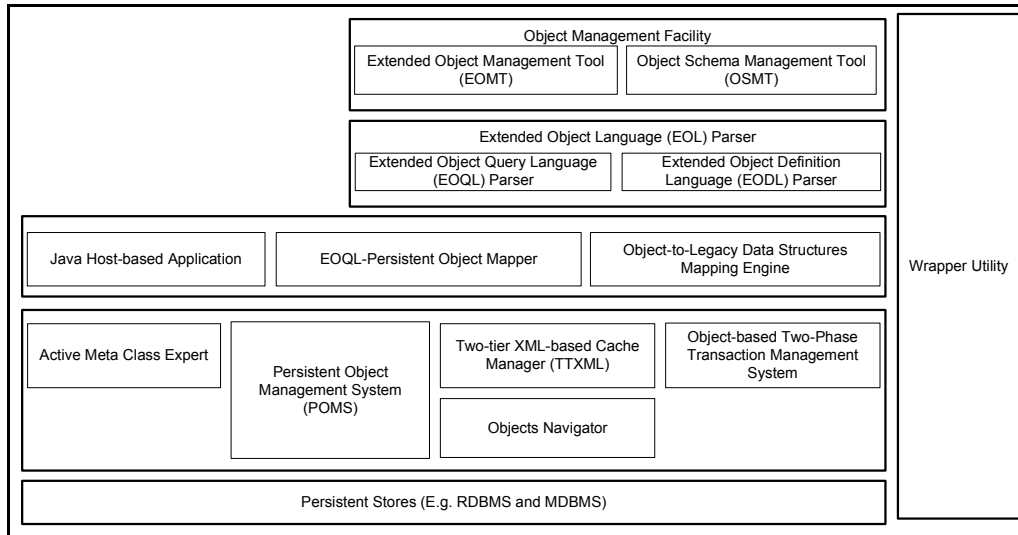


Fig. 1: Conceptual Architectural View of Extolware Persistence Framework [7]

EOL Parser is made of EODL Parser and EOQL Parser, forming the next layer of the framework. It provides services by parsing input from OMF's OSMT and EOMT components. OMF generates output as an EODL statements file or EOQL statements file, where EODL Parser and EOQL Parser respectively read the corresponding statements file as input. EODL Parser parses EODL statements to ensure that they are correct syntactically and semantically, likewise EOQL Parser parses the input file consisting of EOQL statements generated from EOMT.

The next layer of the framework architecture consists of Object-to-Legacy Data Structures Mapping Engine and EOQL-Persistent Object Mapper [5]. Object-to-Legacy Data Structures Mapping Engine provides services to EODL Parser. A successful validated set of EODL statements from EODL Parser is subsequently sent to Object-to-Legacy Data Structures Mapping Engine for the mapping sequences. Likewise, validated EOQL statements are sent to EOQL-Persistent Object Mapper and the information required such as class name, filtering details and class attributes to be displayed are picked up and subsequently bound to a persistent class so that POMS at the next layer can complete the requested database operations. POMS provides a set of interfaces forming an API, which are methods that can be called in any Java application by passing appropriate messages to manipulate persistent objects efficiently.

Active Meta Class Expert works hand-in-hand with Object-to-Legacy Data Structures Mapping Engine. Information regarding the domain will be created using classes at the level of Active Meta Class Expert. Active Meta Class Expert consists of a *ClassModule* class that holds module information about the domain. This includes information such as name of current domain, classes in the domain and other related information. *Node*, *AttributeNode*, *Key*, *Composition*, *Relationship*, *Complex* and *Inheritance* classes are linked to *ClassModule* using Composite design pattern [3]. Since these classes reside within the same layer as Active Meta Class Expert, information is always available. As the name implies, Active Meta Class Expert is an expert as far as providing all necessary details regarding the domain that the framework currently deals with.

POMS is made up of a group of classes providing core object management services. It provides a set of methods such as *read()* and *write()* to developers for persistently manipulating persistent objects with ease. All classes of the selected domain inherit the *PersistentObject* class if these classes are to be used to create persistent objects. Though this incurs some unavoidable coupling of methods with domain classes, it leads to a neat and easy application solution, which directly helps in easing future maintenance effort. As persistent classes are generated for a specific OO programming language, appropriate classes and methods are automatically created. EOQL-Persistent Object Mapper works hand-in-hand with POMS. Information maintained by EOQL-Persistent Object Mapper is passed to POMS so that database services requested by users can be executed.

Two-tier XML-based Cache Manager (TTXML) and Object-based Two-Phase Locking Protocol (OB-2PLP) within the transaction management system are two important components of the framework. In a multiple-user and multi-transactional environment, OB-2PLP is used to ensure data atomicity and consistency. On the other hand, TTXML is designed so that extremely large objects that cannot be fitted into the main memory can be navigated smoothly through the use of a secondary memory or local hard disk in XML document format. POMS and Object Navigator work hand-in-hand with the TTXML component. Object Navigator works with POMS and TTXML by providing navigational capability. It has a complete set of methods such as *first()*, *last()*, *prev()* and *next()*. As a result, all results returned from a *select* database operation can be retrieved and navigated by using Data Service On-Demand algorithm.

The last important layer of the framework is the Wrapper Utility layer. Wrapper Utility contains many miscellaneous classes such as date and data type conversion classes that virtually service all other layers. This design is to centralize all the commonly used classes within a single layer. As a result, no duplication of similar functionalities and less maintenance overhead is incurred in the framework.

## 5.0 Evaluations and Discussions

In this section, mapping algorithms, caching techniques, locking mechanism and design patterns used in the framework are evaluated and discussed.

### 5.1 Mapping Algorithms

The mapping techniques for both relational and multidimensional database systems for simple classes and classes in an inheritance hierarchy use the same mapping algorithm. A simple class is mapped to a table in relational and multidimensional database systems. Such mapping scheme is easy to implement. This approach is widely adopted and implemented by many commercial products. In addition, the object-to-table mapping approach does not consume high computing resources and do not form layers of overhead between the persistent stores and applications. In short, low middle-tier overhead suggests that it is a good mapping method to be deployed.

A number of mapping techniques have been researched and studied for objects in an inheritance hierarchy. These techniques include Vertical mapping, Horizontal mapping and Filtered mapping. Each technique has its pros and cons. In this research, Vertical mapping technique is chosen and implemented. This technique is adapted for classes with a large number of attributes and classes that may have many changes over time as domain requirements change. In most of the business environments, these two characteristics are an important consideration and the technique adopted must be able to accommodate such a need. Vertical mapping uses a class-to-table approach in conjunction with the Data Service On-Demand approach, hence, each reading and writing always involve one table only.

As it is, the selection of data from multiple tables needs expensive join operations for Vertical mapping. Hence, the chosen mapping technique may work very well for a multidimensional database system and not for a relational database system. A relational database system needs some considerable complex join operations to select objects from the tables. As for a multidimensional database system, it has the option to use multi-valued fields to hold its persistent pointers of the immediate parent class. No read operation is required for accessing the immediate parent. If the layer above this parent is interested, a subsequent read operation is carried out for this parent in order to access the class above it. Hence, the mapping technique for a multidimensional database system can be improved from the current implementation which uses join operation to access data between child and parent classes like a relational database does. The advantage of Vertical mapping over Horizontal and Filtered mapping is ease of table schema management with respect to the changes of a domain's requirements. As such, if the mapping schemes can be dynamically changed depending on the business domain and classes in the domain, then the framework will always have a higher performance over cost.

For a class with complex attributes, the relational and multidimensional databases utilise the same approach to perform the mapping processes. A persistent object identifier to point to a related class is used in this research. In this case, another read operation is required to fetch the data from the persistent store when a complex attribute is accessed. Though this is easy to implement, it requires the join operation to perform the selection process from the database system. One possible enhancement that may improve the performance of the read operation of a complex attribute is to embed the data in a multi-valued field of the multidimensional database system. A relational database system, however, does not have an alternative mapping solution toward this shortcoming.

Both relationship and collection properties of a class are designed and implemented very differently for both relational and multidimensional database systems. First of all, the characteristics of these properties are managed in the heart of the mapping engine. Secondly, management of object traversal is managed by a few special classes in the Object Navigator layer so that Data Service On-Demand feature can be implemented successfully. The object navigational requirement uses XML document structure as the secondary virtual disk so that any computing hardware that has very low memory resources can still browse and navigate a very large group of objects that cannot be fitted into the main memory at any one instance.

In the relational database system, an intermediate table is used to hold persistent object identifiers between the primary class and its associated classes. The insertion, deletion and updating operations need to access the primary table, associated table and intermediate table. Besides multiple read and write operations, join operations among these tables are expensive. The design and implementation of this approach is code intensive and complicated. The method adopted for multidimensional databases is simpler, easier and less code intensive. A multi-valued field is used to hold persistent object identifiers of all related records in the persistent store. As the primary object is formed, subsequent accesses to these associated objects are simply reading of records using the object identifiers stored within the multi-valued field. In this approach, no join operation is required. The implementation has shown that multidimensional database systems are

more economical and easier to implement as compared with relational database systems. Relational database systems are observed to suffer from performance and cost due to expensive join operations.

The implementation of relationship and collection of objects in this research has shown that it is complex and difficult to implement a high performance-mapping scheme due to great differences between the non-object-oriented model and object-oriented model. In addition, the collection and relationship characteristics are not supported by the legacy database systems making the mapping engine suffers an additional one layer of overhead between the application and database systems. The implementation of the proposed mapping scheme for relationships management in a relational database system relies on the join operations between the primary table, intermediate table and associated table. On the other hand, a multidimensional database uses a multi-valued field which gives better performance and lower mapping cost. The mapping schemes for the management of collection of objects and relationships between objects do not really have many other alternatives which may bring higher performance with lower cost. As such, two possible improvements may be the use of better design patterns and tighter code in the mapping engine.

In conclusion, both relational and multidimensional database systems are data-driven models but the multidimensional database system has extended its data model by incorporating features like multi-valued field and non-first-normal form. As such, the multidimensional database system requires lesser code and less complicated processes to implement the abovementioned object-oriented features. As for the mapping schemes proposed in this research, there is no perfect scheme that allows a mapping to be designed and implemented with the least cost and resources with highest efficiency and performance. The appropriate mapping scheme is best chosen based on the dynamic changes of the business requirements, the complexity of the tables and relationships between tables, as well as among data stored in most of the tables.

## 5.2 Caching techniques

A number of caching techniques have been proposed. Among these techniques, Memory-based Caching is the simplest and easiest technique to implement. It assumes that the client computer has sufficient memory resources and all objects are stored in the main memory. Memory-based Object Page Caching fetches all the data from the persistent stores and filter those that fulfil the *where* conditions at the local client. The objects are stored in the main memory and those that cannot be fitted in the memory are stored in the secondary cache. Lastly, Memory-based Object Page Pooling Caching (MBPP) technique improves on the cache algorithm by adding memory calculation routines and swapping expired objects with the new objects whenever the main memory starts to run out without loading all objects into the memory. MBPP technique was chosen and implemented for the following reasons:

- It uses an intelligent memory calculation routine to constantly monitor the available resources to decide how many objects should be kept in the memory and how many should be kept in the second-tier cache.
- The *where* condition is used to filter those objects that do not fulfill the condition. As a result, only those that fulfill the condition are managed in the cache.
- Not all the objects are in the main memory. Only the interested objects are kept in the main memory. The rest of the objects are kept in the secondary cache. So, the available memory resources can be used for applications and services.

Two-tier XML-based Caching technique uses the MBPP algorithm as its cache algorithm. The implementation of the MBPP technique is a successful one. Nevertheless, the MBPP cache depends on the memory calculation routine provided by Java language. As a result, at times some memory resources are consumed by the operating system services and Java Virtual Machine which is not within the control of the cache routines. In spite of that, the design allows smooth navigational capability and large object management even within a memory resource constraint environment.

## 5.3 Locking Mechanisms

The proposed Object-Based Two-Phased Locking Protocol enhances on the conventional Two-Phased Locking Protocol (2PLP) and applies the enhanced protocol on the persistent objects. The proposed locking protocol minimises the number of locking modes to four. They are Shared (S) lock, Exclusive (X) lock, Implicit Shared (IS) lock and Implicit Exclusive (IX) lock. In addition, a Time Out Option technique is introduced. The Time Out Option is designed to resolve deadlock among objects that request the same piece of resources from multiple transactions. In order to allow multiple transactions to exclusively run concurrently, multiple threads are used. A monitoring mechanism is created to maintain and monitor these transactions by constantly checking on the requested duration provided by the Time Out Option. Once the duration expires, a message is sent to the requesting transaction to either roll back or commit the object. The owner of the object will then be notified if the object has been modified so that necessary actions can be taken. The management of the logs for roll-back purposes is kept simple and easy. Images of the objects are maintained in an XML file so that the main memory is reserved for applications. Some observations are given below.

- The proposed locking mechanism is efficient and simple to implement.
- The difficulties arise from the implementation of the multiple threads and maintenance of the object images as some objects are quite large and recursive in nature.

- The Time Out Option is effective for a small number of concurrent transactions. If the number of concurrent transactions is large, multiple threads consume a considerable amount of memory resources. This would lead to substantial burden to the computing system. As a result, the locking mechanism would suffer.
- If too many requesting transactions request an object at the same time, the long waiting queue may sometimes cause an indefinite waiting time for the owner of the object.

#### 5.4 Architectural and Design Patterns in the Framework

The adoption of different design patterns in various parts of the framework still has space for improvement and enhancement. A number of observations made from the use of pattern languages in various parts of the framework show that:

- The use of Chain of Responsibility in a number of places of the framework can be improved. The materialization of persistent objects can be improved whenever the search space for the correct persistent objects to be materialised is very deep. Command design pattern is an alternative in this scenario.
- Generalization design pattern is used in POMS to allow any classes that need persistency to inherit the *PersistentObject* class. Doing so, such persistent class will have the capability to carry out many persistent object functionalities such as save, delete and select. However, the use of inheritance creates tightly coupled relationships between classes. An improvement that can be considered is to create a reference to the *PersistentObject* class instead of inheriting from it.

In conclusion, the implementation of the framework allows good evaluation on the design and techniques adopted in various parts of the framework. This allows one to make further improvement to the proposed design and to find out how much is accomplished against the goals defined in the research.

### 6.0 Concluding Remark

Extolware is designed using Layered architectural design pattern. Each layer provides a defined set of services through its consistent interfaces. Layered design pattern reduces coupling between layers and strengthens cohesion property between layers. Classes in each layer send and receive messages among classes in order to deliver services for the layer. The intricacies of the underlying cache management and locking mechanisms are completely hidden from the developers' view. Besides inheriting from the framework's *PersistentObject* that involves some unavoidable coupling in exchange for a neat and easy application solution, cohesion is the key principle used to ensure localization of domain specific details and successive encapsulation of persistence services details into progressively increased abstraction levels. This results in uniformity and reduced coupling, hence easing future maintenance effort, and subsequently helping to achieve the objective of fundamental architectural design principles for good software engineering practice.

Ease of use and simplicity in the persistence services has been achieved through the use of good design patterns in the design and implementation of Extolware. The prototyped framework is still subject to rigorous testing. Nevertheless, the development of the framework has provided plenty of good lessons, experience and feedback which can be used as the foundation for future research work.

### 7.0 References

- [1] Scott W. Ambler. "Mapping Objects To Relational Databases". Ronin International, 1999.
- [2] Akmal B. Chaudhri and Mary Loomis. "Object Databases in Practice". Object-Oriented Data Integration: Running Several Generations of Database Technology in Parallel, (Wolfgang Keller, Christian Mitterbauer and Klaus Wagner). Prentice Hall, 3-20, 1998.
- [3] Craig Larman. "Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process". Prentice-Hall, 2002.
- [4] Sai Peck Lee and Tong Ming Lim. "Object Schema Management Facilities in an Object Wrapper". The 2002 International Conference on Software Engineering Research and Practice (SERP'02). Las Vegas, USA, June 24–27, 2002.
- [5] Tong Ming Lim and Sai Peck Lee. "Object-to-Multidimensional Database Mapping Algorithms". The 6th International Conference of Asian Digital Libraries (ICADL2003). Lecture Note Series in Computer Science, LNCS 2911. T.M.T. Sembok et al (Eds.), Springer-Verlag Berlin Heidelberg 2003, Kuala Lumpur, Dec. 8–11: 556–562, 2003.
- [6] Tong Ming Lim and Sai Peck Lee. "A Persistence Framework for Object to Relational and Multidimensional DBMSs". Malaysian Journal of Computer Science, 17(1): 74–82, June 2004.
- [7] Tong Ming Lim, Sai Peck Lee. "Extolware Persistence Framework: A Prototype". Selected Readings in Software Engineering. UM Press, 103–136, 2005.