Malaysian Journal of Computer Science, Vol. 17 No. 1, June 2004, pp. 74-82

A PERSISTENCE FRAMEWORK FOR OBJECT-TO-RELATIONAL AND MULTIDIMENSIONAL DBMSS

Tong Ming Lim
School of Business and Information Technology
Monash University Sunway Malaysia
No 2 Jalan Kolej, Bandar Sunway, 46150, Petaling Jaya
Selangor Darul Ehsan, Malaysia
email: lim.tong.ming@infotech.monash.edu.my

Sai Peck Lee
Faculty of Computer Science & Information Technology
University of Malaya
50603 Kuala Lumpur, Malaysia
email: saipeck@um.edu.my

ABSTRACT

A persistence framework can be defined as a set of Java classes providing functionalities that map objects to a relational database management system (RDBMS) and/or a multidimensional database management system (MDBMS) so that these objects could be saved, retrieved, deleted and queried. Today, many business applications are developed using some kind of object-oriented programming language and objects in these applications are manipulated through SQL statements that are embedded in the application software. When these applications are small and have very little changes, embedding SQL statements in these applications is fine; but for applications which are dynamic and large, such applications are very hard to maintain and manage. Hence, a robust and extendable persistence framework layer that contains classes for mapping objects to a relational DBMS and/or a multidimensional DBMS allows a developer to concentrate on the development of the actual application. The design of the framework uses a set of good object-oriented design patterns that allow developers to extend functionalities and features easily. This allows saving, updating and querying of objects in the persistence store. In short, this paper focuses on the design, development and mapping of objects to the R/MDBMS persistence framework.

Keywords: ***Relational data model, multidimensional data model, persistence framework, mapping***

1.0 INTRODUCTION

Many business applications developed today use relational database management systems (RDBMSs) or some variation of RDBMS such as multidimensional database management systems (MDBMSs or post-relational or extended relational data models) to maintain and manipulate business data in these applications. RDBMSs and MDBMSs are widely accepted in many medium-to-large organisations because they allow data to be managed and manipulated effectively and efficiently. Nevertheless, if a system is programmed using some object-oriented programming languages (OOPL) such as Java or C++ and is modelled using some object-oriented modelling techniques such as UML and considering that RDBMS and MDBMS data models do not support object-orientated (OO) characteristics such as permitting a class to inherit from other classes and attributes to be defined as complex and collection type, developers must consequently embed SQL statements in their applications in order to map classes to tables so that classes can be managed persistently in a persistence storage.

2.0 CHARACTERISTICS AND WEAKNESSES OF RDBMS AND MDBMS

A relational data model emphasises data stored in a persistence store to be normalized so that each tuple has exactly one value for each attribute, every non-key value must fully depend on the key value, and every non-key value must be non-transitively dependent on the primary key. A RBDMS is simple to understand and data are stored in a two-dimensional table that has only columns and rows. Each row is a record (or tuple) and each column is a field (or attribute). Every field contains simple values such as integer, real, character and date. A RDBMS provides a manipulation language called structured query language (SQL) that has a strong mathematical foundation using relational algebra which allows advanced research in areas such as query optimisation. The relationships between tables in a RDBMS are constructed through attributes in both tables. As for a many-to-many relationship between two tables, a join table is used that contains the primary key from both tables in order to relate tuples from both tables. Attribute values of both tables are duplicated in a third table in order to the exhibit many-to-many relationship of these two tables. Hence, multiple table accesses are required and additional storage spaces are needed. The relational model captures the majority of the system's requirements and semantics of the real world but there are a number of semantics that a relational model does not capture. A relational data model does not support tables inheriting attributes from other tables. Moreover, an attribute in a table cannot be defined as a complex data type or a collection of data objects. In addition, a relational data model does not provide manipulative facility in a table, and as a result, operations to a table must be defined separately. As a result, maintaining a large business application is very time consuming and costly. Thus, many multimedia applications, CAD/CAM applications, time-series applications and applications that need rich, real-world semantics to be captured are difficult to manage and model using a relational data model.

A multidimensional database management system (MDBMS) is an extension of a RDBMS. A MDBMS stores, retrieves, updates and queries a record (or an item) using an extended version of SQL. A multidimensional DBMS provides developers the option to embed a table in an attribute. A MDBMS does not need to comply strictly to a relational model's requirements, that is, a MDBMS does not need to be a 1NF data model. Even though a multidimensional DBMS supports complex objects at the attribute level, each attribute can hold only one table or a collection of values. This feature allows an application to manage a limited version of a complex data type in an attribute. A multidimensional DBMS maintains one-to-one and one-to-many relationships for any two tables within a table. This is an improvement compared to a RDBMS. Hence, it only needs a single read and write operation. As for a many-to-many relationship, the MDBMS will need a join table to maintain this relationship. Nevertheless, it needs a tuple instance in a table to maintain a many-to-many relationship through a variable length multi-value field. A multidimensional DBMS has a few advantages over a traditional relational DBMS, but a MDBMS does not support tables inheriting from other tables and complex attribute types with multiple layers. It does not allow manipulative features to be defined within a table.

3.0 PROVIDING OODBMS FEATURES THROUGH OBJECT-RELATIONAL DBMS AND PERSISTENCE FRAMEWORK

An object data model must support objects and identity, complex data types, class hierarchies with inheritance, composition, association or relationship and operations in a class. These features allow the data model to capture the semantics of complex business applications, constructing CAD/CAM, time-series and multimedia systems. Nevertheless, a lot of business applications used by medium and large business organizations are not migrating to the object-oriented database management system (OODBMS) because it is not widely accepted by the industry, lack of standards, products are not mature and lack of support from large database software vendors. In order to provide features of an OODBMS, data must be made persistent in a mature and widely accepted database management system such as RDBMS or MDBMS. As a result, some database vendors provide an object-relational DBMS (ORDBMS) or object-extended MDBMS that provides features of an OODBMS, while others provide a robust and extensible persistence class framework that allows developers to manipulate data in a RDBMS or MDBMS through this framework. As a result, developers can develop applications using C++ and Java using classes, association, composition, inheritance and methods to save objects persistently in the persistent store through one of these approaches.

A number of ORDBMS products are reviewed in this research. ORDBMS is RDBMS with added object-oriented extensions. Some of the major relational database players that enter into the ORDBMS world are Oracle 8.x from Oracle, Universal Server (Illustra) from Informix, UniSQL/X from UniSQL, Universal Database from IBM and PostgreSQL [20]. Query languages used in these systems are extensions from SQL2 with some proprietary features. Support for SQL3 is still in its infant stage. The market share for ORDBMS will be larger than ODBMS [20] and the selection of which database vendors choose to pick very much depends on a number of criteria. These include support for object-oriented programming languages, simplicity of development, extensibility, complex data relationships, scalability and product maturity. The research also experiments with a few free object-relational database systems such as GigaBASE [21], PREDATOR [22], PostgreSQL [23] and LogicSQL [24]. These object-relational database systems are developed by either individuals, small groups of database enthusiasts or universities. The level of features supported by these products varies and their implementations vary depending on the maturity of the ORDBMS. However, most of these ORDBMSs have not supported all the core and optional features of the object-oriented data model proposed by ODMG [26].

Persistence frameworks such as Hibernate from JBoss [16], jRelationalFramework from SourceForge.net [17], ObjectDriver from INFOBJECTS [18] and VisualBSF from Objectmatter [19] are powerful and popular object/relational persistence frameworks that address the needs of the software developers. Hibernate supports association, inheritance, polymorphism, composition and collection with its extended query language. Most of the major SQL database systems such as mySQL, Oracle, PostgreSQL and Sybase database systems are supported by

Hibernate, jRelationalFramework, VisualBSF and ObjectDriver. jRelationalFramework is one of these frameworks that provides customisable 'hooks' for pre- and post- database operations so that customised processes can be embedded into the framework. ObjectDriver creates relational schema automatically but Hibernate uses XML documents to map objects to table schema and jRelationalFramework maps each persistent class by inheriting *PersistentObject* class and marking persistent attributes using the markModifiedPersistentState() method provided by *PersistentObject*. VisualBSF is one of these products that uses extensive graphical interfaces. Table schema for business classes is automatically generated. The mapped tables can be further managed by the developers in order to fine-tune the mapped mechanisms. In short, these persistence framework vendors research and manufacture their products using their unique approaches. Secondly, they only support relational database systems. Third, mappings are not completely automated. Some of them require manual mappings. Lastly, features supported by them are not complete and consistent. Hence, this research project intends to consolidate and propose a persistence framework architecture with a set of features, with the intention of producing a generic persistence framework using design patterns that suits both relational and non-relational database management systems.

3.1 Research Objective

This research project aims at designing and implementing a robust and extendable layer of objects-to-R/MDBMS persistence framework that consists of a set of Java classes which allows developers to map objects to a relational or multidimensional DBMS (R/MDBMS). These Java classes use open database connectivity (ODBC) API, Java database connectivity (JDBC) and other thin JDBC libraries to connect to both the RDBMS and MDBMS. In the designing of the classes, many reusable design patterns are used. These patterns are Adapter, Singleton, Façade, Proxy and Iterator. These reusable and well-defined patterns help producing a robust and reusable persistence framework.

The primary research objective of the project intends to consolidate various persistence framework architectures and mapping algorithms by providing the following set of features:

i. To utilise at least two database systems that are entirely different such as relational database and multidimensional database in the framework in order to demonstrate that the framework is versatile to handle both relational and non-relational database systems.
ii. To architect a generic and extendable persistence framework using design patterns so that customised codes can be embedded at the 'hooks' provided by the framework for calculation and validating purposes.
iii. To examine various mapping techniques and propose a mapping technique that auto-generates mapped table schemas for persistent classes through the use of active meta data maintained by framework during run-time.
iv. Provide a Data Service On-demand feature by designing and implementing an XML-based two-tier caching algorithm for large objects navigational purposes.
v. To extend an object query language and integrate it to the application programming interfaces provided by the persistence framework.
vi. Design and implement an enhanced object-based two-phase locking protocol with an innovative Time-Out option mechanism to resolve dead-lock problems.

3.2 An Overview on Objects to R/MDBMS/XML Persistence Framework

The persistence framework proposed in this paper as shown in Fig. 1 emphasises on ease of class extension, ease of class modification, usage of class setters and getters, ease of database connection setting to various databases and dynamic translation of object manipulation statements to internal structured query statements. In addition, class *ClassModuleFile* allows loading and unloading of classes' schemas. Layered design structure is used to construct the entire persistence framework.

The topmost layer is called **persistence framework layer** (PFW) that is responsible for the class schema to table schema maintenance task. It works together with class *SQLManageTable* to create, alter and delete RDBMS and MDBMS tables as class schema changes. Future development would be to enhance the functionalities of this layer to incorporate database migration and updating options. This will allow existing data to be migrated to a new database when the class schema changes. The PFW layer consists of four primary classes. They are *PFWmapMaster*, *PFWmapRDBMSlayer*, *PFWmapMDBMSlayer* and *PFWmapXMLlayer* classes.

In the **database layer** (DL), there are four sub layers in which each layer plays different roles. The first sub layer is called **internal database naming** (IDN) in which classes in this layer allow developers to name the type of database used and define the name of a RDBMS or MDBMS product that is intended to be connected to. Two interfaces are defined in the first sub layer that allow developers to use consistent method names to activate appropriate methods when connecting, disconnecting and setting the type of API driver to be used. The second sub layer is called **database transactional setting** (DTS). This layer allows developers to set database parameters such as set auto commits on and set transactional processing feature off. The third layer is called **database parameter setting** (DPS). This layer allows developers to set the type of middleware or driver such as JDBC or ODBC driver to be used to connect to the underneath database engine. The fourth sub layer is called **database broker** (DB). This layer performs various actual connection activities such as connecting to a relational database engine and disconnecting from a database engine. In this layer, all database broker classes are implemented from a *ConnectionBroker* interface.

The third layer is called **object SQL layer** (OSQL). This layer has four classes that are responsible for validating the object SQL statements and mapping object SQL statements to SQL92 statements.

The fourth layer is called **SQL statement processing** (SP) layer. It contains five primary classes that manage tables in a database management system, process SQL statements produced by the object SQL layer and perform the actual database operations such as delete and insert.

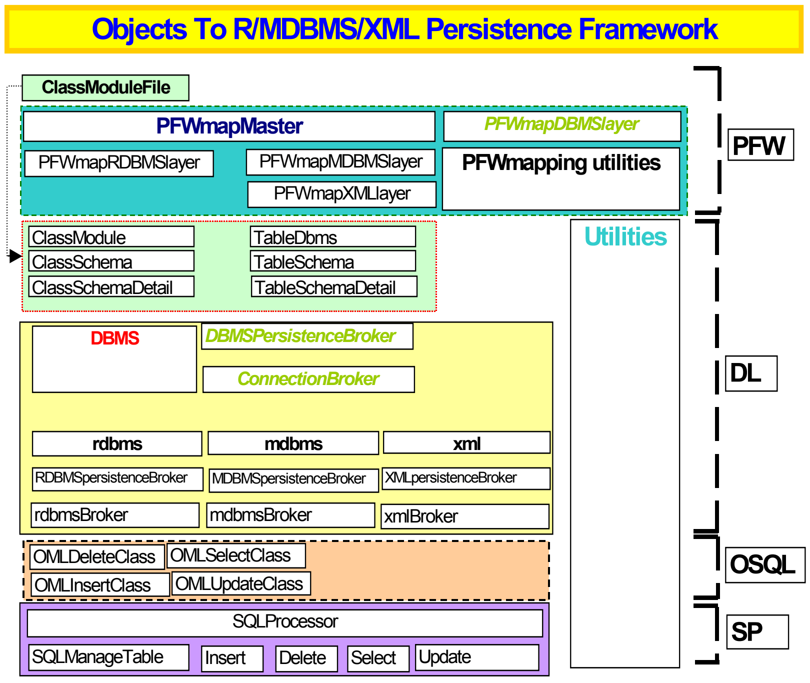

Fig. 1: Objects to Relational, Multidimensional and XML Persistence Framework

In the framework, Iterator design pattern [25] is used to make accessing instances of a collection easier and not exposing the underlying representation. The following sample Java code illustrates that use of Iterator design pattern allows additional checking and sequential access of items in a collection. This design pattern also has the strength of type validation of object.

```
public class course
{
  String name;
  int credithours;
}

public class student
{
  List courselist = new List();
  public void addcourse(course acourse) {}
  public student.Itr iterator()
  { return new Itr(); }
  public class Itr
  {
    private Iterator I = courselist.iterator();
    public boolean hasNext()
    { return i.hasNext(); }
    public course next()
    { return (course)i.next(); }
  }
}
```

3.2.1 Persistence Framework Layer

Fig. 2 shows a complete diagram of how each class in the persistence framework layer communicates to other classes. In this layer, interface *PFWmapDBMSlayer* is designed using the Adapter design pattern. The intent of the adapter design pattern [25] is to provide the interface a client expects, using the services of a class with a different interface. Methods defined in interface *PFWmapDBMSlayer* are to be 'materialised' in classes that inherit from it. The Facade design pattern is also applied here for class *PFWmapMaster* where it provides a unified interface to a set of interfaces in a subsystem. The *PFWmapMaster* class makes other classes such as class *PFWmapRDBMSlayer* and *PFWmapMDBMSlayer* easier to communicate with and use. The use of the Facade design pattern [25] is to provide an interface that makes a subsystem easy to use. Class *PFWmapMaster* and class *classModuleFile* are designed using the Singleton design pattern because they have only one instance and provide a global point of access to it. The single global access point provided by the Singleton design pattern always maintains one instance throughout the life of the application available to all classes in the framework.

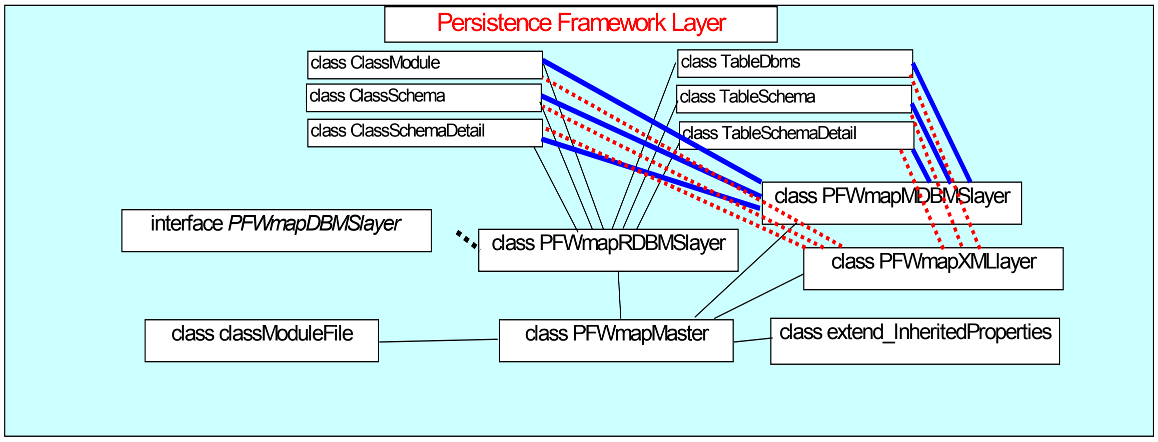

Fig. 2: Persistence Framework layer

Class *PFWmapMaster* refers to *classModuleFile* to obtain a reference to the active class schema definition. The *PFWmapMaster* class contains three primary methods; *PFWmapRDBMSlayer*, *PFWmapMDBMSlayer* and *PFWmapXMLlayer*, that carry out mapping processes for different types of databases. In the mapping process, *ClassModule*, *ClassSchema* and *ClassSchemaDetail* are referred and instances of *TableDbms*, *TableSchema* and *TableSchemaDetail* are created. A set of SQL statements is created as an output of this process. These SQL statements are used to create, update or delete tables in the targeted database engines.

3.2.2 Database Layer

The database layer as shown in Fig. 3 provides a few services. Each sub layer carries out specific functionalities. Many classes in this layer are designed using good design patterns from [4]. The Adapter design pattern is applied in this layer for interfaces *DBMSPersistenceBroker* and *ConnectionBroker*. The Adapter design pattern allows interfaces to be converted to the client's interface. The Proxy design pattern is used to design the class *proxyObj*. This design pattern provides a surrogate for another object to control access to it. It serves as a placeholder for records that are regularly manipulated. This design allows frequently read/write records to be fetched from the persistence store and stored into the memory for fast access and reduces network load. Class *DBMS* is designed using the Singleton pattern because it has only one instance throughout the life span of the entire application.

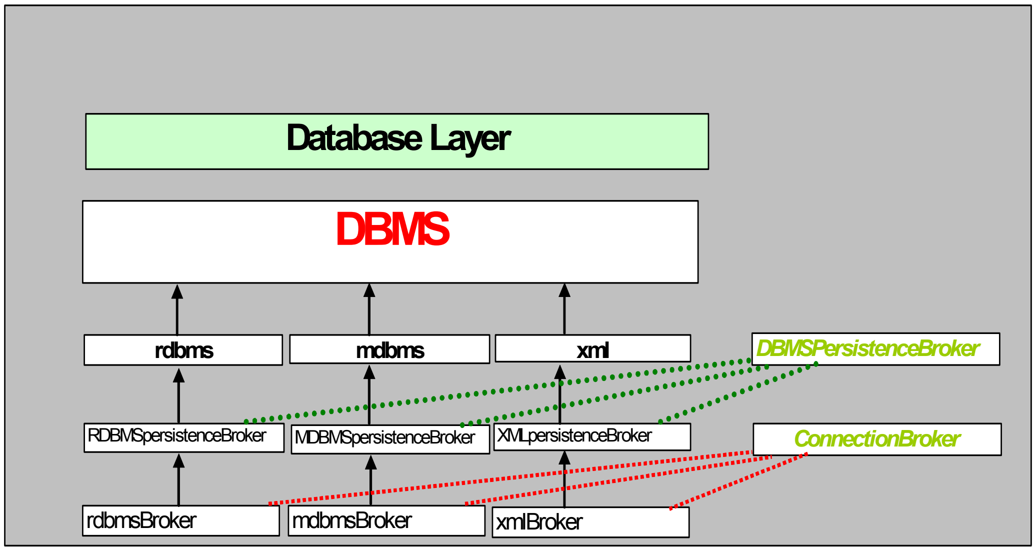

Fig. 3: Database layer

The internal database naming (IDN) sub layer contains classes that allow developers to name the type of database used and define the name of the RDBMS or MDBMS product intended to be connected to. The database transactional setting (DTS) layer allows developers to define whether to turn on or turn off auto commits, set transactional processing to auto rollback and whether records are in read-only mode. The database parameter setting (DPS) layer allows developers to define the type of middleware to be used. Some database engines permit alternative drivers that deliver better performance in the place of the original driver. This layer allows developers to have a few middleware choices. The last layer (DB) performs various actual connection activities such as connecting to a relational database engine and disconnecting from a database engine.

3.2.3 Object SQL Layer

The Object SQL Layer as shown in Fig. 4 has four classes that are responsible for validating the object SQL statements and mapping object SQL statements to SQL92 statements. These four classes are *OMLDeleteClass*, *OMLInsertClass*, *OMLSelectClass* and *OMLUpdateClass*. These classes retrieve active class definitions from the memory by referencing to *ClassModule* class. A *ClassModule* holds a collection of *ClassSchema* instances and a *ClassSchema* object holds a collection of *ClassSchemaDetail* instances. The structure used in this design is called composite design. Such a design pattern forms a tree structure to represent part-whole hierarchies [4]. It allows clients to treat individual objects and compositions of objects uniformly. For each instance of *ClassSchema*, it holds a collection of *TableSchema* and each *ClassSchemaDetail* object holds a collection of *TableSchemaDetail* instances. This allows each class definition stored in *ClassSchema* to refer to the appropriate table schema. For each attribute in the class definition, it refers to the appropriate table and table field information. The output of this layer produces a set of SQL statements that will be used by the SQL Statement Processing layer.

3.2.4 SQL Statement Processing Layer

In the SQL statement processing layer as shown in Fig. 5, the Singleton design pattern is used to design class *SQLProcessor* that has only one instance throughout the entire life span of the application.

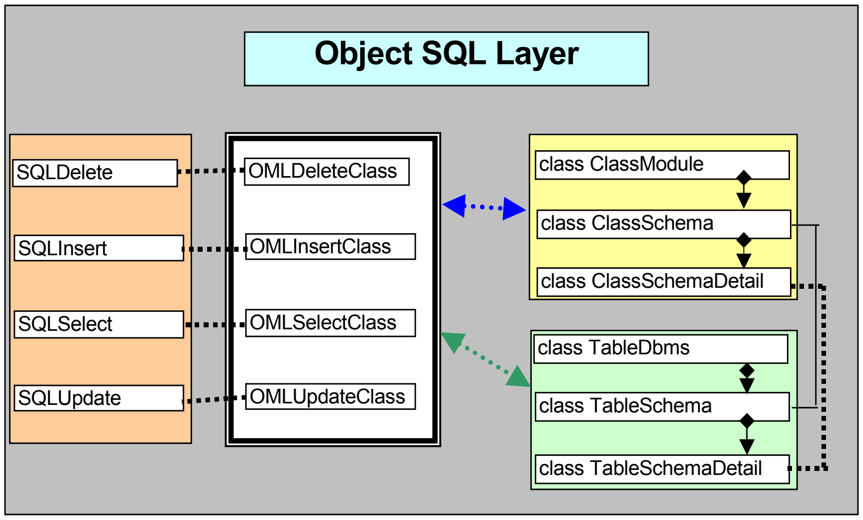

Fig. 4: SQL Statement Processing layer

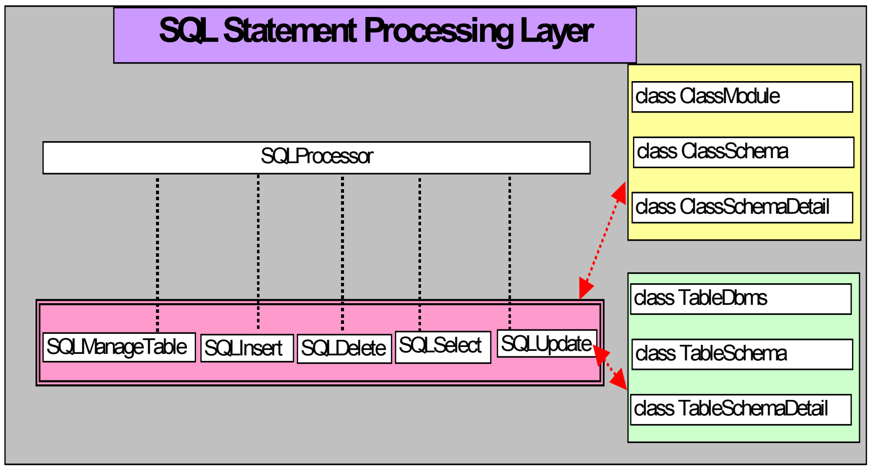

Fig. 5: SQL Statement Processing Layer

The SQL statement processing (SP) layer contains five primary classes that manage tables in a database management system. It processes SQL statements that are produced by the object SQL layer and performs the actual database operations such as delete and insert. These five classes; *SQLManageTable*, *SQLInsert*, *SQLDelete* and *SQLUpdate*, inherit from *SQLProcessor*. *SQLProcessor* defines SQL statements that will be processed by these classes.

4.0 CONCLUSION

Many aspects of such an implementation still require further enhancement and refinement. The list of tasks that need to be addressed in this research includes incorporation of features such as association and composition; performance issues such as instances traversal in a composition; implementation of mapping using multidimensional DBMS; and incorporation of XML capabilities so that data of a particular database could be exchanged with other applications.

REFERENCES

[1] Sai Peck Lee and Tong Ming Lim, “Objects to Multidimensional Database Wrapping Mechanism”. *Annual AoM/IaoM International Conference on Computer Science*, August 6-8, 1999, Westgate Hotel, San Diego, California.

[2] Sai Peck Lee and Tong Ming Lim, “Classes and Objects Management Multidimensional DBMS Data Model”, *IASTED International Conference Software Engineering* (SE ’97) San Francisco, California, 2-6 Nov. 1997.

[3] Sai Peck Lee and Tong Ming Lim, “Objects Collection Management in Multidimensional DBMS Data Model”, *ISORC Kyoto International Conference Hall*, Kyoto Japan, April 20-22, 1998.

[4] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[5] Elisa Bertino and Lorenzo Martino, *Object-oriented Database Systems*. Addison-Wesley, 1998.

[6] Georg Lausen and Gottfried Vossen, *Models and Languages of Object-Oriented Databases*. Addison-Wesley, 1998.

[7] Daniel Goldinj, “Middle Tier/Object Server Architecture (MTOS) - Object/Relational Mapping using Generic Business Objects”, *White Paper*, 1998.

[8] Daniel Goldinj, “Middle Tier/Object Server Architecture (MTOS) - Kernel Process”, *White Paper*, 1998.

[9] Kyle Brown and Bruce G. Whitenack, “Crossing Chasms, A Pattern Language for Object-RDBMS Integration”, White Paper, Knowledge Systems Corp. 1995. A shortened version is contained in: John M. Vlissides, James O. Coplien, and Norman L. Kerth(Eds.): Pattern Languages of Program Design 2, Addison Wesley 1996.

[10] Wolfgang Keller, “Mapping Objects to Tables: a Pattern Language” in *Proceeding of the 1997 European Pattern Languages of Programming Conference*, Irrsee, Germary, Siemens Technical Report 120/SW1/FB 1997.

[11] INFORMIX software Inc., “The Architecture of UniVerse”, January 2001.

[12] INFORMIX software Inc., “Extended Relational Databases”, *White Paper*, January 2001.

[13] Scott W. Ambler, “The Design of a Robust Persistence Layer for Relational Databases”, *Ronin International*, November 28, 2000.

[14] Scott W. Ambler, “Mapping objects to relational database”, *White Paper, Ronin International*, October 21, 2000.

[15] Scott W. Ambler, “Mapping Objects to Relational Databases: What You Need to Know and Why”, *Ronin International*, July 2000.

[16] Hibernate, “Relational Persistence for Idiomatic Java”, JBoss http://hibernate.bluemars.net/.

[17] jRelationalFramework, “jRelationalFramework Version 2.0”, SOURCEFORGE.net http://jrf.sourceforge.net/.

[18] ObjectDRIVER, “ObjectDRIVER – Architecture”, InfoObjects http://www.infobjects.com/.

[19] VisualBSF, “Mapping Tool Guide”, Objectmatter http://www.objectmatter.com/.

[20] Steve McClure, “Object Database vs. Object-Relational Databases”, IDC August 1997.

[21] GigaBASE, “GigaBASE Object-Relational Database System”, GigaBASE http://www.garret.ru/.

[22] PREDATOR, "PREDATOR Enhanced Data Type Object-Relational DBMS", PREDATOR http://www.distlab.dk/.

[23] PostgreSQL, *PostgreSQL: An Open-Source Object Relational Database Management System*. Paragon Corporation http://www.praragoncorporation.com/.

[24] LogicSQL, "LogicSQL: An Object-Relational Database System", LogicSQL http://www.cs.ualberta.ca/.

[25] Steven John Metsker, *Design Patterns Java Workbook*. Addison Wesley.

[26] Cattell, *The Object Databases Standard: ODMG 3.0*. Morgan Kauffman 2001.

BIOGRAPHY

Tong Ming Lim is a PhD candidate at the Faculty of Computer Science and Information Technology, University of Malaya. He is working on the object-oriented wrapper in his final year. He worked in TAR College, as a GM and IT Director in two commercial software houses for about 10 years. He is currently lecturing in Monash University, Malaysia, on computer science courses such as database management system, artificial intelligence and computer graphic programming. He is an ACM member and IEEE member since 1993.

Sai Peck Lee is currently an associate professor at the Faculty of Computer Science and Information Technology, University of Malaya. She obtained her Master of Computer Science from University of Malaya in August 1990, her Diplôme d'Études Approfondies (D.E.A.) in Computer Science from Université Pierre et Marie Curie (Paris VI) in July 1991 and her Ph.D. degree in Computer Science from Université Panthéon-Sorbonne (Paris I) in July 1994. Her current research interests include Software Engineering, Object-Oriented (OO) Methodology, Software Reuse and Application Framework, Organisational Memory System, Information Systems and Database Engineering, OO Analysis and Design for E-Commerce Applications and Auction Protocols. She has published an academic book and more than 70 papers in various local and international conferences and journals. She is a member of IEEE Computer Society, a founding member of Informing Science Institute, a member of the editorial board of a research bulletin, and has served as the executive editor of a journal for 2 years, as well as an active member in the reviewer committees and programme committees of several local and international conferences.