PersistF: A Transparent Persistence Framework with Architecture Applying Design Patterns

Samir Jusic and Lee Sai Peck
Faculty of Computer Science and Information Technology
University Malaya, Kuala Lumpur, Malaysia

samir@alcassoft.com saipeck@um.edu.my

Abstract

Persistence is the term used in computer science to describe a capability to store data structures in non-volatile storage such as a file system or a relational database (Wikipedia, 2003). There is a growing need to simplify the interactions among separate systems and to build complex software systems that are made out of components with well-defined services. At the base-level of such complex systems lies a persistence framework – a set of classes tailored to save, retrieve, update and delete objects to and from their persistence stores, such as databases and flat files. This paper presents the design, and implementation of the transparent persistence framework called PersistF.

Design patterns (Gamma, Helm, Johnson, & Vlissides, 1995) have been widely accepted in the software engineering community as the recommended approach to developing software. Part of the research effort of this work included the application of well-known design patterns in order to build the framework. We present how consistent application of design patterns allowed us to build a fully working persistence framework.

In order to support development of modern complex applications, some recent application development environments for different programming languages are built in with some kind of persistence framework. To use these frameworks in target applications, the developer often must go through a steep learning curve as each framework is specific in some way, but namely in respect of configuring the framework's runtime engine with enough information about the domain application. This configuration is often time consuming and error prone. Most of the existing frameworks require complicated configuration steps that are imposed onto the developer. Our aim with this work was to present a framework that will not require such complicated configuration steps and would provide its service to the target application with virtually no configuration of the framework's handling of domain classes.

Keywords: transparent persistence framework, design patterns, Java, relational to object model mapping

Introduction

As computing power increases, the number and variations of software applications built seem to have no end in sight. For a very large percentage of object-oriented applications, there is a constant need to store and retrieve application data. Without a doubt, the most popular storage mechanism is a relational database. However, the applications have to deal with the problem that arises due to the fact that objects (in the object-oriented world) are conceptually different in their structure and semantics from relational databases. In order to save objects in relational databases, there must be a mapping between the two domains. This mapping is often not natural and is described as the impedance mismatch (Ambler, 2002). The fundamental problem between the object-oriented world and the relational databases as the storage mechanism used is often referred to as the impedance mismatch. The object-oriented paradigm is based on proven software engineering principles, whereas the relational paradigm is based on proven mathematical principles. Because the underlying paradigms are different, the two technologies do not work together seamlessly (Ambler, 2002, 2000). To address this problem, a persistence framework is introduced to seamlessly provide the necessary mapping between the two worlds or paradigms.

There are a number of persistence frameworks available in the industry and academia to bridge the gap between object-oriented applications and relational database management systems (RDBMs). However, these frameworks are usually targeted to handle storage into relational databases and have little or no support for other persistence types, such as object-oriented databases and XML files. In addition, working with a persistence framework is rarely simple as it requires fairly complicated configuration to be done by the application developer before being able to use it.

The persistence framework described in this paper has the following goals:

- Provide transparent persistence (little or no intrusion into the application code).
- Support multiple persistence mechanism types, including relational databases, object-oriented databases and XML files.
- Be simple to use with little configuration required.

This paper describes the architecture and design of a transparent persistence framework called PersistF (**Persist**ence **F**ramework). PersistF addresses the inherent complexity of using a persistence framework by minimizing the required configuration that needs to be done by the application developer. In addition, if at all needed, it has a very minimal intrusion of its own code into the target application code, therefore being transparent to the developer. Finally, the system supports multiple persistence mechanism types, including relational databases, object-oriented databases and XML files for storage. In this paper, we focus on the application of software design patterns in the architecture of PersistF.

The remainder of this paper is organized as follows. Section 2 presents a brief overview of the major persistence frameworks currently available. Section 3 describes the features of PersistF. Section 4 describes the architecture and design of the framework as well as the application of design patterns in its implementation. In Section 5 some experiences in using the framework to build a web-based application are presented. Lastly, Section 6 summarizes our work and provides an outline for future work.

Related Work

The issue of object storage is a highly debated one. There are quite a number of research projects and commercial tools available in the market and on the Internet. A brief overview of the most popular tools is presented below. The list, however, is not an exhaustive one.

JDO – Java Data Objects

The Java Data Objects (JDO) (Russell, 2003) API is a Java model's standard interface-based abstraction of persistence. JDO technology can be used to directly store Java domain model's instances into the persistent store (database). Developed under the Java Community Process (Java

Community Process, 2004), JDO has great momentum in the development community. It provides transparent persistence through the use of post-compilation of Java class files to add additional interfaces to compiled Java classes (byte-code manipulation).

Hibernate

Hibernate (King & Bauer, 2003) is a powerful, high performance object-relational persistence and query service for Java. Hibernate rejects the use of build-time code generation / bytecode processing. Instead, reflection and runtime bytecode generation are used and SQL generation occurs at system startup time. Hibernate has surfaced as one of the main persistence engines for Java. Its support has been lately boosted further by the acceptance of Hibernate into JBoss (JBoss, 2004). Widespread usage among Java community is already present.

Java Persistence API - JSR 220

The Java Persistence API (SUN Microsystems, 2006) originated as part of the work of the JSR 220 Expert Group to simplify EJB CMP entity beans. The Java Persistence API is not based on any single existing persistence framework but incorporates--and improves upon--ideas contributed by many popular frameworks, including Hibernate, TopLink, JDO, and others.

- The Java Persistence API is a POJO (Plain Old Java Objects) persistence API for object-relational mapping.
- It contains a full object-relational mapping specification supporting the use of Java language metadata annotations and/or XML descriptors to define the mapping between Java objects and a relational database.
- It supports a rich, SQL-like query language.
- It also supports the use of pluggable persistence providers.

Castor

Castor (Castor, 2004) is an open source data-binding framework for Java. Castor aims to provide transparent persistence service for relational databases and XML files. It is a collection of advanced mapping frameworks, which allows Java objects to be transformed to and from XML elements and relational database content.

Summary Remarks

The above-mentioned frameworks and APIs have a common trait: they all require the end user (developer) to configure the framework’s behavior by providing enough information about the target application and the way in which persistence aspects of it are to be handled in the context of a given framework. This decreases the overall productivity for the developer and complicates the development process by forcing the developer to learn the lingo of the target framework. Their approaches to accomplishing persistence differs, as well as the features that the frameworks support, but they all require additional steps in the form of configuration of the framework to get any actual persistence work done. Hibernate is currently the most popular open source framework found in the industry. However, it appears that JSR 220 specification may soon become the leader (partly due to the fact that Hibernate creator, Gavin King, has actively joined the JSR-220 specification team).

PersistF: A Persistence Framework Overview

The detailed design and functionality of PersistF is discussed in the following sections. First, an overview of all the supported functionality is provided followed by the discussion on the framework's design choices. The functionality provided by the framework is grouped into: (1) provisions for transparent persistence, (2) support for user-defined classes and relationships between them, and (3) support for different persistence mechanisms.

Transparency

With transparency, the framework takes care of all the persistence-related work on behalf of the user. The management of unique identifiers, the retrieval of objects from the data store, materialization of objects from the specific data store is done automatically. There is a minimal amount of persistence-related code that is propagated at the application level. The domain level (i.e. definition of business classes) is untouched by the framework code footprint.

Support for User-defined Classes and Configuration

The framework does not require any prior knowledge about the domain in which it is supposed to be used. With the use of Java's reflection mechanisms, the framework dynamically inspects and creates the necessary mappings between the object and target persistence mechanism. This approach not only eliminates the need for users to dedicate time and resources to the development of tedious persistence-related code, but also enables them to fully focus on the domain development.

Where PersistF differs from more popular frameworks, such as Hibernate (King & Bauer, 2003), is in its innovative configuration-free setup. PersistF dynamically discovers the properties of classes and is able to provide the mapping into the persistence store without requiring explicit (and complicated) configuration often found in other frameworks. Our current work requires the developer to register domain classes which are to be handled by the framework. There is no need to provide mapping information in the form of configuration files for each field and where it is supposed to be stored.

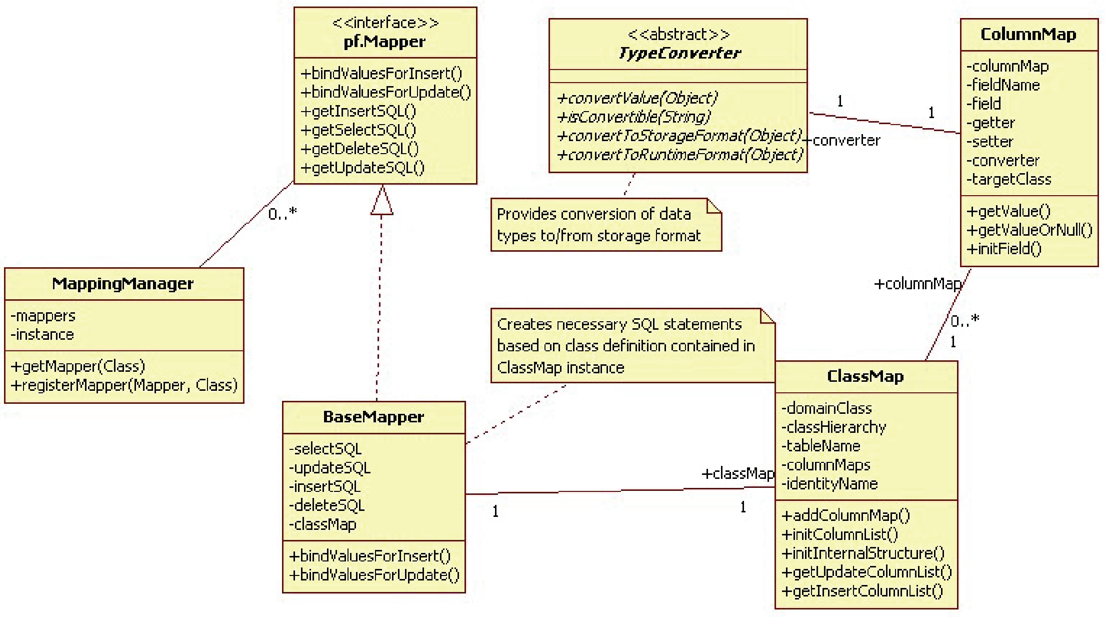


Figure 1: Framework Mapping Strategies

Figure 1 depicts the most important classes which collaborate to perform class mapping functionality. Mapping of user-defined (and all other) classes is done through the use of ClassMap instances which basically capture the class structure – name and its related fields. Each field has a ColumnMap instance that captures the way in which a field can be mapped to a persistence storage. Mapper class actually performs the logic behind doing the various persistence tasks such as extracting values and building the necessary SQL statements based on the structure of the ClassMap. There is a dedicated ClassMap for each user-defined class. Whenever a typical application starts it will attempt to initialize PersistF. Initialization of PersistF will load all available metadata mapping with the help of a singleton MappingManager instance. MappingManager stores all of the individual Mapper classes. Other parts of the framework will use the MappingManager to obtain the required Mapper and ClassMap classes. Operations of loading, saving and deleting objects require the MappingManager to obtain the ClassMap and Mapper instances according to the target Java class and forward the task to them.

Multiple Persistence Mechanism Types

PersistF is built around the concept of working with multiple persistence mechanism types. As a starting point, the framework supports relational databases and XML-based documents as persistence stores. Support for additional storage mechanisms is achieved by extending the framework at well-defined customization points. Customization takes the form of implementing interfaces which will be automatically called by the framework once they are plugged into it. Our aim is to have support for object-oriented databases in the near future.

Architectural Design of PersistF

The design of PersistF is based upon layers (Buschmann, Meunier, Rohnert, Sommerlad, & Stal, 1996). The major framework layers are depicted in Figure 2.

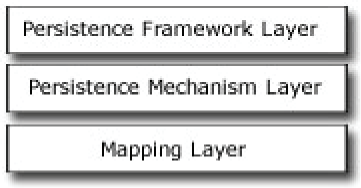


Figure 2: Architectural Layers of PersistF

Each of the three layers contains additional classes that perform functionalities pertaining to that level.

Persistence Framework Layer serves as the framework's API for the user's end applications. It provides the classes that should be directly used from or communicated to target applications such as Session and PersistenceServer.

Persistence Mechanism Layer provides the infrastructure to connect to different storage mechanisms – for example, relational databases and flat files.

Mapping Layer provides the necessary mapping mechanism between the in-memory native object representation of the data and the target storage mechanism representation. Classes in this layer are tailored for different storage mechanisms to provide the specific implementation to them.

Persistence Framework Layer

Persistence Framework Layer is the primary contact point to the end-user's application. This layer contains the Session, Transaction, and PersistenceServer classes as the primary set of classes which the application must use to get persistence work done. Classes in this layer are exposed to external applications which make use of them to accomplish persistence tasks.

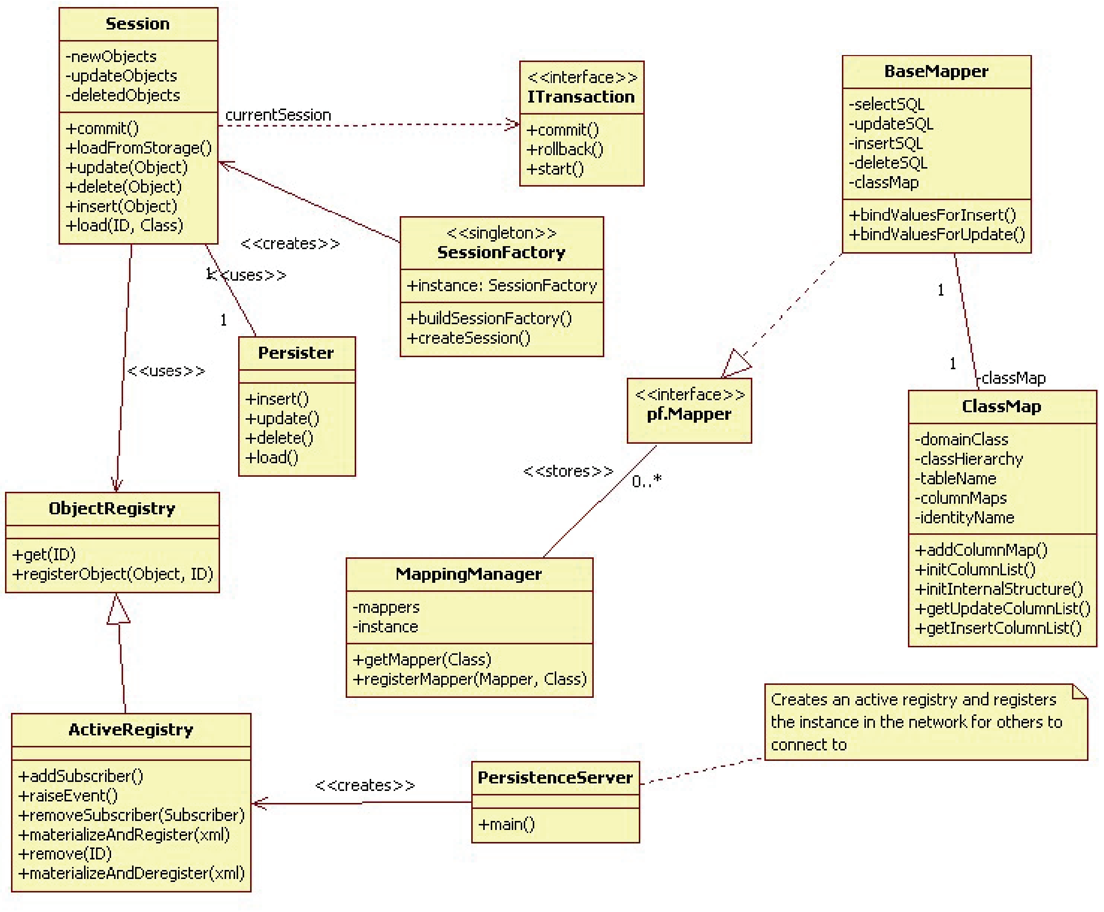


Figure 3: Persistence Framework Layer Classes of PersistF

The framework exports the Session component which has the central role of performing object lookup, creation and storing from a given data store. ITransaction and Session classes allow client applications to create units of work in the context of persistence. Sessions are obtained from the SessionFactory, and Transactions are created on demand from a given Session instance. Figure 3 illustrates the core classes found in the persistence framework layer as well as the connections to the most important classes beyond that (namely the Persister which is used to perform persistent store-related detailed operation as well as MappingManager and Mapper which are responsible with maintaining the mapping information of domain classes to their storage).

Persistence Mechanism Layer

Persistence Mechanism Layer provides the functionality related to connecting to a given persistence mechanism. This layer supports connection to the relational databases and flat-file (XML) files.

Future extensions in this layer would allow the addition of new persistence storage types (LDAP, e-mails) through subclassing of PersistenceMechanism class. Naturally, to support completely new persistence types, there is a need to not only subclass classes in this layer but also in other affected layers, as appropriate.

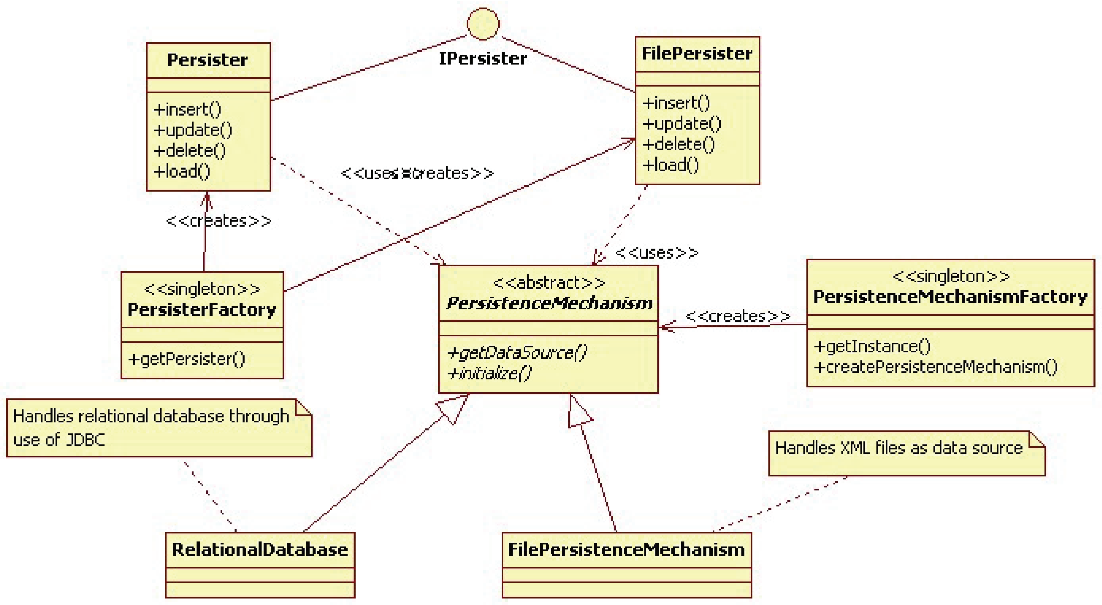


Figure 4: Persistence Mechanism Layer Classes of PersistF

The sample implementation of the framework focused on the relational databases. The necessary customization, hook and concrete implementation classes were built with the idea of a relational database as the target storage. Figure 4 illustrates persistence mechanism layer of PersistF.

The layer is designed to support extensions to the types of storage through the implementation of interfaces that abstract the actual storage type. By implementing interfaces in different layer's packages, it is possible to provide implementation classes that will be able to store data into new and yet unsupported data stores. This provides great flexibility to the end-user as they would not be tied to a particular type of storage.

Each concrete implementation of IPersister interface interacts with a concrete sub-class of the PersistenceMechanism in order to perform the actual storage and retrieval operations on the target persistence store. In order to extend the framework with support for a new persistence mechanism type, an appropriate IPersister must be created and a sub-class of PersistenceMechanism that deals with the actual storage.

Mapping Layer

Mapping layer contains the classes that perform the mapping between Java class definitions and their respective storage equivalents. The mapping classes are ClassMap, ColumnMap, Mapping-Manager and Mapper. Mapping layer offers the service of analyzing the structure of Java objects passed to it and extracting the information found in them. It provides the ability to map the contents of Java objects to corresponding persistence storage record sets and vice versa. Mapping layer is built on top of Java's runtime reflection support that allows for inspection of object structure and content at runtime.

Mapping layer is a stateless layer in that it does not maintain state information. This means that calling this layer's service to map an object to its storage equivalent does not require that the layer to contain prior information on that object - only its structure is maintained within the layer, which applies to all objects of a given class.

Design Patterns Used in PersistF

PersistF is built through deliberate applications of software design patterns. Wealth of information is available in (Fowler, 2003; Gamma et al., 1995; Larman, 1997), which cover the definition of design patterns and their concrete application in problem solving. Our work has been greatly improved by the application of these design principles. This section will elaborate on the ways in which design patterns were applied in the construction process of PersistF.

Façade

Façade pattern allows for a definition of an application service's entry point. It seemed highly appropriate to have such a mechanism in-built into the framework so that external applications would have a fairly simple entry point to access and use PersistF. Our main façade class is the Session class that provides the necessary methods to load, insert, update and delete objects. Session class in turn "knows" how to interact with a number of PersistF classes to get the work accomplished. Figure 5 illustrates this.

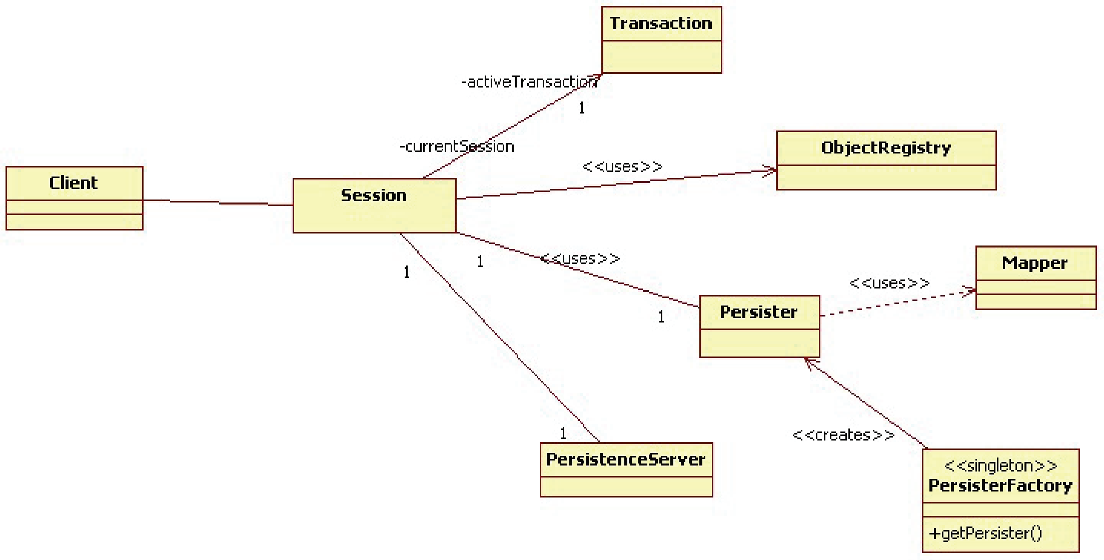


Figure 5: Session Pattern in PersistF

Factory

Factory pattern is a very common design pattern with several variations. PersistF uses the Factory design pattern in several places to hide the ways in which certain implementation classes are provided to the framework's runtime execution environment.

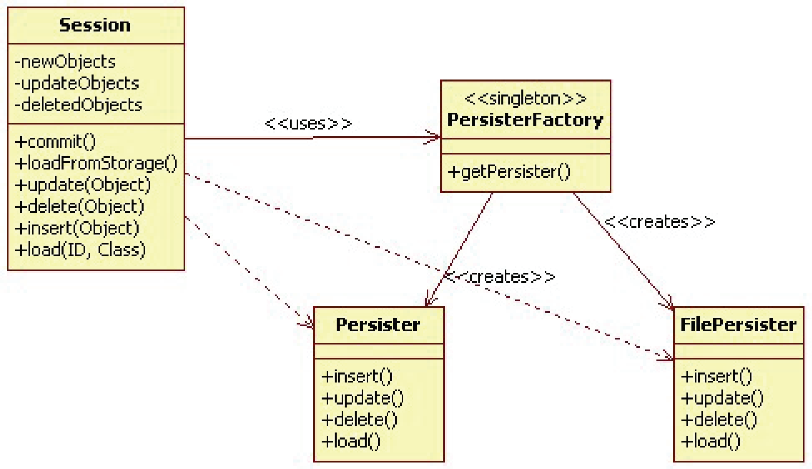


Figure 6: Factory Pattern in PersistF

Figure 6 illustrates the creation of Persister classes is accomplished through a PersisterFactory class that provides the appropriate Persister class, depending on the framework settings at runtime. Session class need not know the decision making process that goes into the creation of the appropriate Persister class – the details of it are hidden in the factory itself. Factory creation guarantees that the Session will obtain the appropriate Persister class that is able to work with the underlying persistence mechanism.

Strategy

Strategy pattern is applied to the way in which the class inheritance hierarchy mapping is handled in PersistF. A different hierarchy mapping strategy is employed depending on the chosen way in which the application deals with the hierarchy for a target domain class. PersistF is able to use a different hierarchy mapping algorithm for each domain class, having support for mapping one class to one table, one concrete class to one table and one table for entire class hierarchy mapping strategies. These strategies are further extended in the context of handling hierarchy mapping for file-based persistence mechanism. Actual implementation of the target mapping algorithm is created with the use of a Factory design pattern that creates different Strategy implementations. This is illustrated in Figure 7.

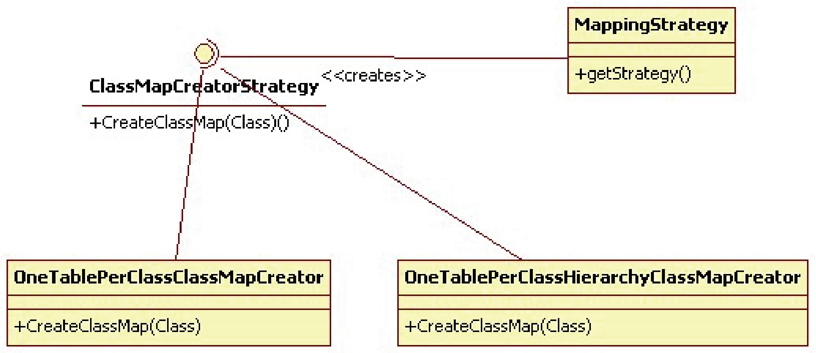


Figure 7: Strategy Pattern in PersistF

Unit of Work

PersistF uses Unit of Work (Fowler, 2003) pattern to deal with demarcation of persistence work. Transaction demarcations (start and end of a transaction) are handled through the application of the Unit of Work pattern. It provides for an easy way to deal with having a complete picture of all affected objects for a given persistence context through the ability to keep track of added, changed or deleted persistent objects. Figure 8 illustrates the use of Unit of Work pattern in the context of Session and Transaction classes.

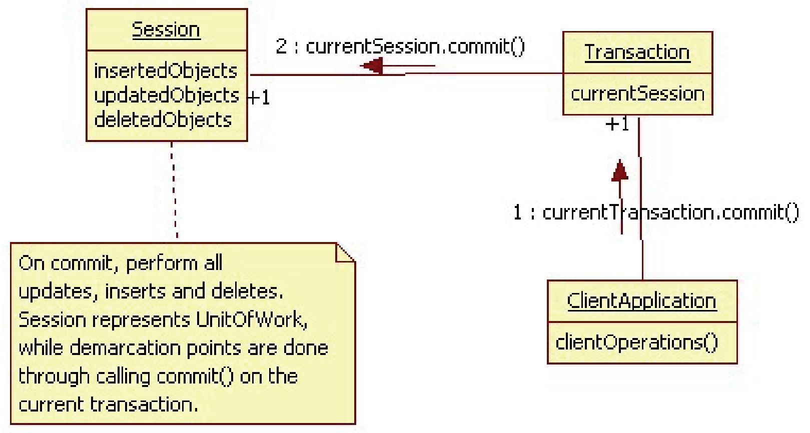
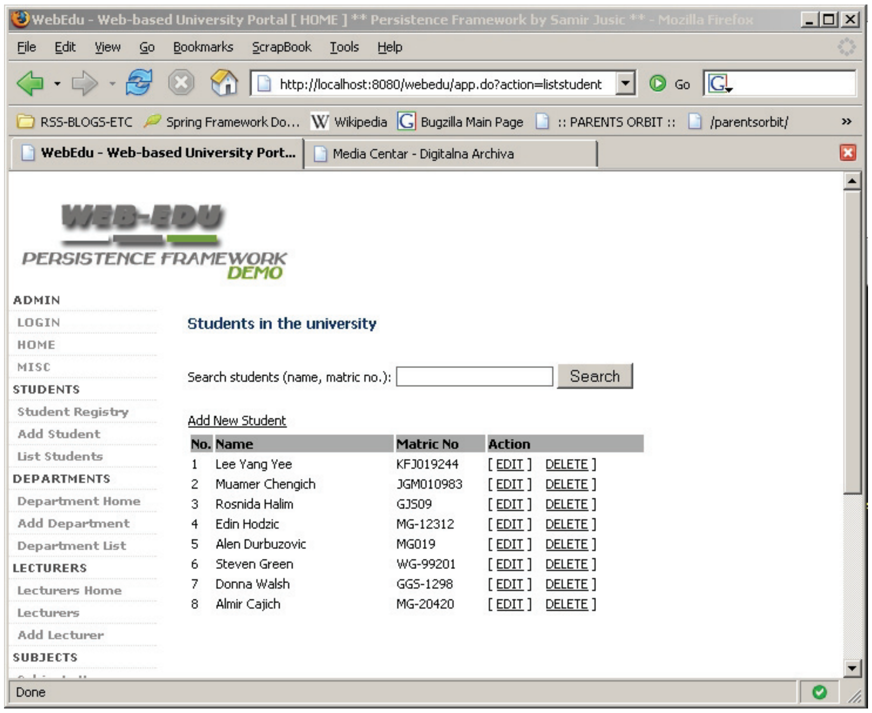

Session
insertedObjects
updatedObjects
deletedObjects
+1
2 : currentSession.commit()
Transaction
currentSession
+1
1 : currentTransaction.commit()
ClientApplication
clientOperations()
On commit, perform all updates, inserts and deletes. Session represents UnitOfWork, while demarcation points are done through calling commit() on the current transaction.

Figure 8: Unit of Work Pattern in PersistF

Using the Framework

Testing the framework has been used within a web application environment in which the framework's functionality was put to the test. The goal of the exercise was to test-drive the framework in a typical development environment. In this case, we choose a web-based application as our development test bed. The application was a general student registry, based on Java JSP technologies and running on Apache Tomcat Servlet container, using MySQL open source database and XML as the different persistence stores.

We concluded that the speed of development was very good as we were able to quickly add support for domain classes and provide the basic CRUD functionalities on them without the need for much configuration on the part of the framework. In addition, switching to a different storage mechanism had no impact on the application, except the obvious minor configuration of the framework. Figures 9 and 10 show screenshots of the sample application we built to test drive PersistF's features.


WebEdu - Web-based University Portal [HOME] ** Persistence Framework by Samir Jusic ** - Mozilla Firefox
File Edit View Go Bookmarks ScrapBook Tools Help
http://localhost:8080/webedu/app.do?action=liststudent
RSS-BLOGS-ETC Spring Framework Do... Wikipedia Bugzilla Main Page :: PARENTS ORBIT :: /parentsorbit/
WebEdu - Web-based University Port... Media Centar - Digitalna Archiva
WEB-EDU PERSISTENCE FRAMEWORK DEMO
ADMIN LOGIN HOME MISC STUDENTS Student Registry Add Student List Students DEPARTMENTS Department Home Add Department Department List LECTURERS Lecturers Home Lecturers Add Lecturer SUBJECTS
Search
Done

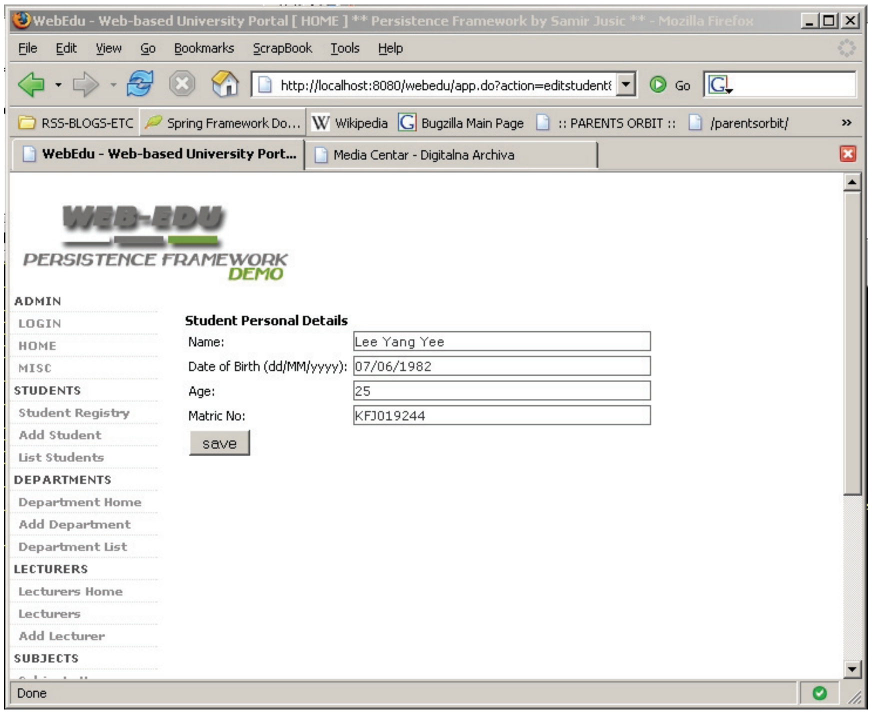
Students in the university

Search students (name, matric no.):

Add New Student

No.	Name	Matric No	Action
1	Lee Yang Yee	KFJ019244	[EDIT] DELETE]
2	Muamer Chengich	JGM010983	[EDIT] DELETE]
3	Rosnida Halim	GJ509	[EDIT] DELETE]
4	Edin Hodzic	MG-12312	[EDIT] DELETE]
5	Alen Durbuzovic	MG019	[EDIT] DELETE]
6	Steven Green	WG-99201	[EDIT] DELETE]
7	Donna Walsh	GGS-1298	[EDIT] DELETE]
8	Almir Cajich	MG-20420	[EDIT] DELETE]

Figure 9: PersistF Demo Application – Showing list of records


WebEdu - Web-based University Portal [HOME] ** Persistence Framework by Samir Jusic ** - Mozilla Firefox
File Edit View Go Bookmarks ScrapBook Tools Help
http://localhost:8080/webedu/app.do?action=editstudent
RSS-BLOGS-ETC Spring Framework Do... Wikipedia Bugzilla Main Page :: PARENTS ORBIT :: /parentsorbit/
WebEdu - Web-based University Port... Media Centar - Digitalna Archiva
WEB-EDU PERSISTENCE FRAMEWORK DEMO
ADMIN LOGIN HOME MISC STUDENTS Student Registry Add Student List Students DEPARTMENTS Department Home Add Department Department List LECTURERS Lecturers Home Lecturers Add Lecturer SUBJECTS
save
Done

Student Personal Details

Name:	Lee Yang Yee
Date of Birth (dd/MM/yyyy):	07/06/1982
Age:	25
Matric No:	KFJ019244

Figure 10: PersistF Demo Application – Editing Record

Conclusion

Our work produced a transparent persistence framework, called PersistF, that is able to handle multiple persistence storage mechanisms – a feature often not found in the most popular Java-based persistence framework in the market. In addition, PersistF requires minimal configuration by the developer. The goal was to minimize complicated configuration files which are often a source of frustration during development. Instead, PersistF dynamically works with domain classes and generates the required storage constructs for interacting with the persistence store on the fly. Our initial implementation focused on providing the support for multiple persistence mechanisms as well as achieving the ease of use.

Future work includes a more robust handling of persistence transactions and tackling caching issues. In addition, framework will add support for object-oriented databases.

References

Ambler, W.S. (2002). The object-relational impedance mismatch. Retrieved February 20, 2007 from http://www.agiledata.org/essays/impedanceMismatch.html

Ambler, W.S. (2000). Mapping objects to relational databases: O/R mapping in detail. Retrieved February 20, 2007 from http://www.agiledata.org/essays/mappingObjects.html

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. & Stal, M. (1996). *Pattern-oriented software architecture: a system of patterns.* John Wiley and Sons.

Castor - open source data binding framework. (2004). Retrieved from http://www.castor.org/

Fowler, M. (2003). *Patterns of enterprise application architecture.* Pearson Education Inc.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns – Elements of reusable object-oriented software*. Addison Wesley.

Java Community Process. (2004). http://www.jcp.org/

JBoss. (2004). Open source Java application server. Retrieved from http://www.jboss.org/index.html

Jordan, M., & Atkinson, M. (1998). Orthogonal Persistence for Java - A Mid-term Report. *Proceedings of the Third International Workshop on Persistence and Java (PJW3)*.

King, G. & Bauer, C. (2006). Hibernate – relational persistence for idiomatic Java. Retrieved February 20, 2007 from http://www.hibernate.org/hib_docs/v3/reference/en/html/.

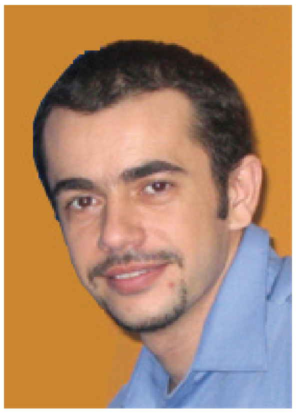
Larman, C. (1997). *Applying UML and patterns*. Prentice Hall.

Russell, C. (2003). JSR-000012 Java™ data objects (JDO) specification (maintenance release), version 2.1. Retrieved from http://jcp.org/aboutJava/communityprocess/final/jsr012/index2.html.

SUN Microsystems, EJB 3.0 Expert Group. (2006). JSR 220 - enterprise Java beans, version 3.0, Java persistence API.

Wikipedia. (2003). Persistence. Retrieved February 20, 2007 from http://en.wikipedia.org/wiki/Persistence.

Biographies



Samir Jusic (*samir@alcassoft.com*) is currently a Chief Technology Officer for Alcassoft – a software development company based in Kuala Lumpur, Malaysia. His interests are with software architecture and ways to improve the software development lifecycle through agile methodologies. He holds a bachelor's degree in Information Systems Engineering from Multimedia University, Malaysia and a Master of Software Engineering from University Malaya, Malaysia.

Sai Peck Lee is currently an associate professor at Faculty of Computer Science & Information Technology, University of Malaya. She obtained her Master of Computer Science from University of Malaya, her Diplôme d'Études Approfondies (D. E. A.) in Computer Science from University of Pierre et Marie Curie (Paris VI) and her Ph.D. degree in Computer Science from University of Panthéon-Sorbonne (Paris I). Her current research interests include Software Engineering, Object-Oriented (OO) Methodology, Software Reuse and Framework-based Development, 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 had also served as the executive editor of a journal for 2 years, and has been in the programme and reviewer committees of several local and international conferences.