Fall 2005

Object Relational Mapping for Enterprise Application Architecture

Musafare Machisa
Regis University

Object Relational Mapping for Enterprise Application Architecture

A thesis submitted

by

Musafare Machisa

to

Regis University

In partial fulfillment of the requirement for the degree of

Master of Science

In Computer Information Technology

Acknowledgements

I would like to thank my advisor, Rob Sjodin for devoting his time answering questions, meeting with me and offering guidance on the project. I would also like to thank my wife and kids for their understanding and patience during the time that I worked day, night and weekends to complete this project.

Abstract

This project investigates the patterns/techniques used and issues that are encountered in developing object-oriented (OO) to relational data mappings for enterprise application architectures. The research will identify the different techniques of object relational data mapping as documented by Martin Fowler in "Patterns of Enterprise Application Architecture" (text for MSCS 630). A prototype implementation will be developed using the techniques found in this research. The goal of the project is to analyze and understand Martin Fowler's patterns and other object relational mapping patterns used in industry and to recommend a set of "best practices".

Table of Contents

Acknowledgements ---------- 3

Abstract ---------- 5

Chapter 1 - Introduction ---------- 8
- Background ---------- 8
- Goals and objectives ---------- 9
- Issues Encountered ---------- 10
- Scope and Limitations ---------- 11
- Definition of terms ---------- 11
- Summary ---------- 12

Chapter 2 - Review of Literature and Research ---------- 14
- Research Methods ---------- 16
- Literature ---------- 16
- Questionnaires ---------- 16
- Random Sampling ---------- 16
- Knowns and Unknowns ---------- 17
- Contribution that the project will make ---------- 18

Chapter 3 - Explanation of the Patterns ---------- 19
- Data Source Patterns ---------- 19
- Behavioral Patterns ---------- 25
- Structural Patterns ---------- 27
- Summary ---------- 38

Chapter 4 - Methodology ---------- 40
- Phases ---------- 40
- Work Breakdown Structure ---------- 41
- Resource Requirements ---------- 42
- Systems Development Life Cycle Model ---------- 42
- Prototype Design ---------- 43
- Prototype Development ---------- 44
- Prototype Testing ---------- 46
- Summary ---------- 47

Chapter 5 - Project History ---------- 48
- Project Initiation ---------- 48
- Project Management ---------- 48
- Project Events and Milestones ---------- 49
- What Went Right ---------- 50
- What Went Wrong ---------- 51
- Summary ---------- 52

Chapter 6 - Lessons Learned ---------- 53
- Lessons Learned ---------- 53
- Next Steps ---------- 54
- Conclusion & Recommendations ---------- 55
- Summary ---------- 56

Bibliography and References ---------- 63

List of Figures

Figure 1 – Application Layers 18
Figure 2 – Table Data Gateway 19
Figure 3 – Row Data Gateway 21
Figure 4 – Active Record 22
Figure 5 – Data Mapper 24
Figure 6 – Unit of Work 25
Figure 7 – Identity Map Sequence Diagram 27
Figure 8 – Foreign Key Mapping 30
Figure 9 – Person Inheritance Structure 32
Figure 10 – Single Table Inheritance 33
Figure 11 – Player Inheritance Structure 36
Figure 12 - Class Table Inheritance 37
Figure 13 – Project Gantt Chart 48
Figure 14 – Rapid Prototype Methodology 57
Figure 15– Bank Account Domain Class Model 58
Figure 16 - Bank Account Active Record Model 58
Figure 17– Bank Account Data Model 59
Figure 18– Create bank account 60
Figure 19 - Update bank account 60
Figure 20– Test withdrawal 61
Figure 21 - Test deposit 61
Figure 22 – Test Find account and Lazy load 62

Chapter 1 - Introduction

Background

Object relational mapping is a fundamental concept of linking in-memory objects to relational database tables. Several patterns of object relational mapping are documented and widely implemented in industry. In most cases the same patterns are known by different names depending on the author. Object relational mapping addresses a number of issues caused by subtle differences between objects and database relations. A study of Martin Fowler's book, "Patterns of Enterprise Application Architecture" was carried out to understand his patterns. Research was also conducted looking at other literature and it was found out that there is a plethora of resources and discussions on this most important subject.

The foremost problem identified by all authors is that of "impedance mismatch" [Barry & Associates] between in-memory objects and relational database tables. There is a general consensus that object programming languages could be better served by object-oriented databases rather than relational databases because of mapping issues that arise but this is always hindered by politics and the fact that relational databases are well understood and widely established in industry. It is also argued that object-oriented databases do not have the same appeal as relational databases especially to non-expert users. The concept of objects is very abstract and making use of object databases would require most general users to have knowledge about objects. Data warehousing and business intelligence may be another reason why object databases have not widely penetrated the market.

According to Leavitt Communications "…OO databases don't scale up to high transaction volumes…" Warehouse systems rely heavily on relational databases and many less complicated tools are available in the market for querying data warehouses on relational databases. Leavitt Communications also cites the absence of compelling reasons from management's point, to move from relational databases to object databases.

The size of the domain model (number of domain objects that need mapping) and the hierarchy of objects through inheritance are some of the factors that determine the complexity of object relational mapping. Many vendors have come up with tools for object relational mapping to undercut the time that is spent by developers on solving non-business problems. According to Martin Fowler and Scott Ambler, coding an object relational layer is easy with a simple domain and simple hierarchies. As these two get more complex, more programmer time is wasted troubleshooting bugs, performance and tuning problems that are related to object relational mapping. That is when a mapping tool becomes handy.

Goals and objectives

The goals of this project are to conduct a research of object relational mapping architectures and patterns, document the core patterns and implement a prototype application using the patterns learned. The prototype will be implemented using the Java programming language.

Issues Encountered

The main issue encountered during the project concerns the problems that are associated with the research methods other than the study of literature. Sampling three companies for interviewing was among the list of research methods in the proposal but time to get the target people into an interview was a problem due to their pressing schedules. The author also used questionnaires in order to try to fill the void left by interviews but that did not work either. No responses were received from the three respondents.

Another issue was that time seemed to run out before the prototype was started. There was a bit of panic on the part of the author during the prototype development when it became clear that the programming was time-consuming and the project paper was not making progress. A review of the project scope helped to solve the problem. Also, as a result of time constraints, the goal of implementing the Unit of Work pattern was not accomplished.

Scope and Limitations

The project's scope will be limited to the following considerations:

- Conduct research and understand object relational mapping patterns
- Document the patterns.
- Identify problems, advantages and disadvantages of object relational mapping.
- Design and implement a prototype employing the core patterns required in an object to relational mapping and also give reasons for the choice of the patterns used in the prototype.
- Present the project/thesis paper to Regis University for graduation

Definition of terms

Class

Is an abstract type, a blueprint of common aspects of objects of the same type [Java Sun Tutorial]

Domain model

A picture or diagram of all the "nouns" [Martin Fowler, 26] of the business model

Object

An object is a software bundle of related variables and methods. Software objects are often used to model real-world objects you find in everyday life [Java Sun tutorial].

Relational Database

Organization of data into tables (relations)

UML

Unified Modeling Language

SDLC

Systems development life cycle

Identity Map

An in-memory object also known as a hash map. It contains key-value pairs of data or objects. It works like a phone directory but the keys have to be unique.

Metadata

Data that describes other data (data about data)

Primary Key

Unique identifier of a database table record

Foreign Key

An identifier that relates a database table record to another record in a different table

Persistence

Moving/saving of data to disk or database storage

Summary

The purpose of this project is to understand object relational mapping patterns. This chapter discussed the background, goals, objectives, scope, issues and terminology associated with this project. While databases and objects have to be mapped, all literature that was read has emphasized the enormity of the problems that mapping can cause. These problems should be good news to software vendors who need to sell object relational mapping tools that facilitate the development process, allowing developers to concentrate on solving business problems.

Chapter 2 - Review of Literature and Research

The research was done primarily with the intent of understanding patterns of enterprise application architecture as documented by Martin Fowler et al. Fowler's book was used as a baseline of the research that would also include literature from the internet and discussions with experts. It became clear during the research that there is an abundance of knowledge and references on the subject of object relational mapping. The patterns and their problems are well understood and documented. Fowler's book was found to be hard to read and understand in one pass and reading literature on the internet helped the author to appreciate and understand Fowler's patterns.

The following resources were used as literature for this project.

- Patterns of Enterprise Application Architecture – Martin Fowler
- http://www.hibernate.org/ (Hibernate)
- http://www.Service-architecture.com/object-relational-mapping/articles/index.html
- http://www.Agiledata.org/essays/mappingObjects.html
- http://java.sun.com/docs

The Hibernate website dwells on the Hibernate object relational mapping tool. Its main focus is marketing of the tool. It also describes the object relational mapping concepts and what the tool can accomplish. As a marketing site, it provides product download and documentation links.

The Service-architecture and Agiledata web sites were found to be very helpful because of their simplified approach in explaining object relational patterns and design considerations. The Service-architecture site promotes "transparent persistence" of data where database calls are made through the programming language instead of database call interfaces like JDBC. This enables the database objects to be treated in the same manner as in-memory objects. Besides transparent persistence being easier for programmers it also reduces "…code and, through caching, improves performance over using embedded SQL …" [Barry & Associates, Inc.] thus making it is faster. The catch is that transparent persistence, while it is easier for programmers to understand, is time consuming and the best approach is to use a tool that can generate this code instead of writing it in-house.

The java.sun website was referred to for verification of definitions of terms and application programming interfaces (APIs).

Research Methods

The following research methods were used:

Literature

This included reading Patterns of Enterprise Application Architecture by Martin Fowler and internet resources cited earlier in this chapter.

Questionnaires

Three questionnaires were used to supplement materials read and also to get an understanding of what object relational patterns were implemented by the chosen individuals/companies.

Random Sampling

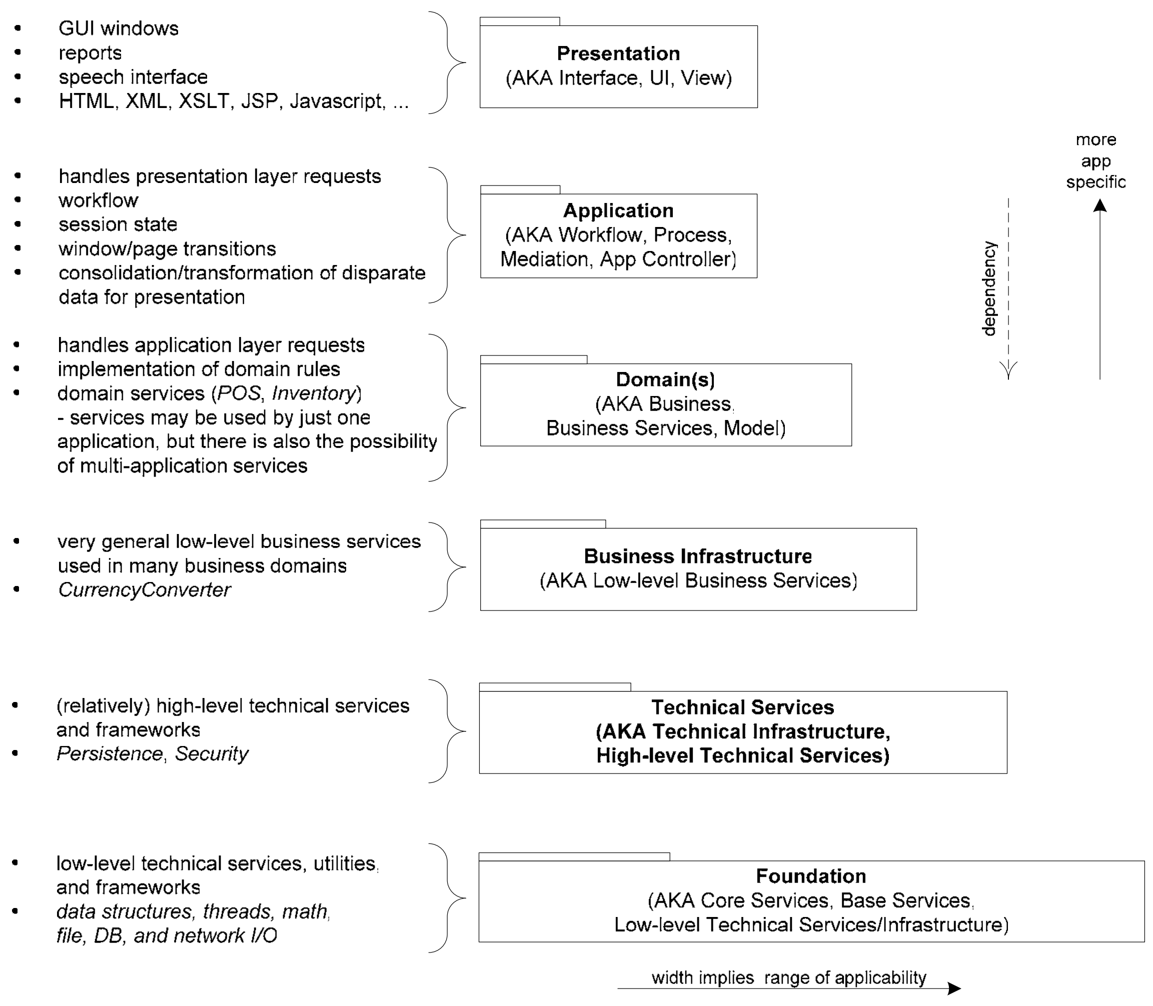
Random sampling was used for the choice of respondents of the questionnaires above. Although this was a very small list of respondents, the sample was going to ensure that responses were going to cover experiences from Java, dot Net and C++ architects or developers. “In random sampling, all items have some chance of selection that can be calculated. Random sampling technique ensures that bias […] is not introduced regarding who is included in the survey” [Australian Bureau of Statistics, http://www.abs.gov.au/].

The interviewing method was replaced with questionnaires due to tight schedules or lack of time on the part of the interviewees. The most effective of these research methods was literature reading mainly because it was within the author’s control.

Knowns and Unknowns

It is widely known that most of today’s applications, whether they are written in object-oriented or procedural language, save data to permanent storage in the form of relational tables. These applications communicate with databases through standard interfaces like Java Database Connectivity (JDBC) and Open Database Connectivity (ODBC) using the standard Structured Query Language (SQL) and other vendor-specific SQL.

From the author’s experience, it is also known that most application outages are caused by or blamed on databases because data unavailability means system unavailability or uselessness. Another known aspect is the recommended separation of applications into layers that are cohesive, performing one and only one function. The layers should also be loosely coupled, independent of other layers. In this regard, object oriented design recommends a service layer to handle data persistence and retrieval from the database. Craig Larman [Applying UML and Patterns page 450] refers to this layer as the Technical Services layer because it deals with the technologies of databases and data persistence.



GUI windows
reports
speech interface
HTML, XML, XSLT, JSP, Javascript, ...
Presentation
(AKA Interface, UI, View)
handles presentation layer requests
workflow
session state
window/page transitions
consolidation/transformation of disparate data for presentation
Application
(AKA Workflow, Process, Mediation, App Controller)
handles application layer requests
implementation of domain rules
domain services (POS, Inventory)
- services may be used by just one application, but there is also the possibility of multi-application services
Domain(s)
(AKA Business, Business Services, Model)
very general low-level business services used in many business domains
CurrencyConverter
Business Infrastructure
(AKA Low-level Business Services)
(relatively) high-level technical services and frameworks
Persistence, Security
Technical Services
(AKA Technical Infrastructure, High-level Technical Services)
low-level technical services, utilities, and frameworks
data structures, threads, math, file, DB, and network I/O
Foundation
(AKA Core Services, Base Services, Low-level Technical Services/Infrastructure)
more app specific
dependency
width implies range of applicability

Figure 1 – Application Layers

Contribution that the project will make

The project will achieve the following objectives:

1. List mapping patterns, issues and techniques for object relational mappings.
2. Document cost/buy recommendations - factors that should be considered by a development team when deciding to build or buy an existing object relational mapping solution

Chapter 3 - Explanation of the Patterns

Martin Fowler's book discusses several patterns but this project will focus on three groups of patterns namely data source architectural patterns, object relational behavioral patterns and object relational structural patterns. Other authors' views will also be cited where it is appropriate.

Data Source Patterns

Data source patterns are either Gateways or Mappers [Martin Fowler] that are invariably organized into software layers of the application architecture.

1. Table Data Gateway (TDG)

In this pattern each database table has a single object that handles all rows of the table (data) and all access SQL code for select, insert, update and delete for that particular table. In simpler applications, TDG can be used to encapsulate access to more than one table. Data Access Objects are synonymous with TDG in that they offer "…create/read/update/delete (CRUD)…" [Deepak Alur] functionality on data.

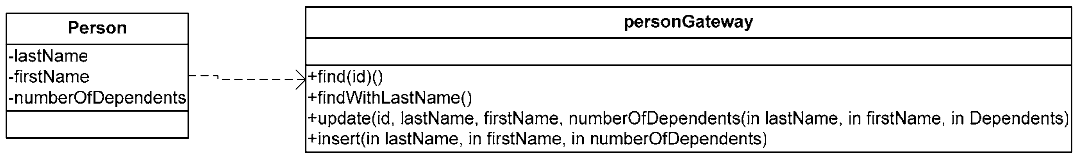

Person
-lastName
-firstName
-numberOfDependents
personGateway
+find(id)()
+findWithLastName()
+update(id, lastName, firstName, numberOfDependents(in lastName, in firstName, in Dependents)
+insert(in lastName, in firstName, in numberOfDependents)

Figure 2 – Table Data Gateway

[Martin Fowler page 144]

A person table would have the following PersonGateway

```
Class PersonGateway
        Public IDataReader FindAll() {
                String sql = “select * from person”;
                Return new OleDbCommand(sql, DB.connection).ExecuteReader();
        }
        Public IDataReader FindWithLastName(String lastName) {
                String sql = “select * from person where lastname = ?”;
                IDbCommand comm. = new OleDbCommand(sql, DB.connection);
                comm.Parameters.Add(new OleDbParameter(“lastname”, lastname));
                return comm.ExecuteReader();
        }
        … [Martin Fowler page 147]
```

2. Row Data Gateway (RDG)

A row data gateway object offers access to a single row of a table. This means that there is one instance for each table row pulled from the database. This pattern works in conjunction with find operations outside of the RDG objects. The find operations generate in-memory objects that are exact matches of each table row and each column is type-mapped to the programming language’s data type. According to Martin Fowler this pattern has a disadvantage of creating an extra layer of finder methods. However, he does not mention that the finder methods may as well be created as static methods within the gateway class. Effort must be put into ensuring that no more than one RDG operate on the same tables as this may result in one RDG operations being undone by the other. This pattern works well with small, less complex mapping, especially where there is no domain model or when it’s used as a “data holder” [Martin Fowler] for a domain object because RDG typically does not include domain logic in the objects but getter and setter methods to get or set values to attributes of a row.

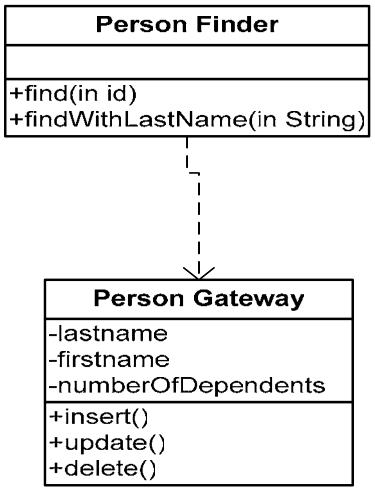


Figure 3 – Row Data Gateway

[Martin Fowler, page152]

3. Active Record (AR)

An Active Record is an object that encapsulates all three elements namely data, data access and domain logic. This is a heavily coupled pattern that works well as long as there is no complex logic and the number of tables involved is low (this is a relative comparison). Data fields map directly from the database to the objects data members and all tables are also mapped to Active Record objects just as in the domain model. Coupling in this pattern introduces scalability and change management issues since a change to the database will require a change to the Active Record. Another disadvantage of Active record is that as the database tables' relationships increase, the business logic also gets complex making it difficult to use active record because relationships, collections and inheritance do not map easily onto Active Record.

The getCourse method in the diagram is an example of a simple business logic that is embedded in the Student object. The logic in this method may include checking for courses that are completed and passed.

Example of an Active record looks like:

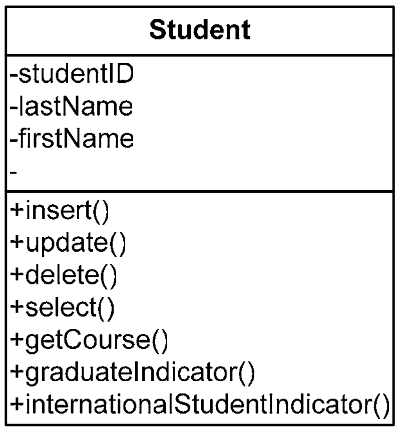


Figure 4 – Active Record

4. Data Mapper

A data mapper is a layer of separation between objects and database tables. It consists of finder methods that know how to get data from the database and abstracts the database's existence from the domain model. No SQL code is involved and the domain and database can evolve independent of each other. Independence of layers is a major goal in large, complex software projects as it eventually contributes to service availability and better customer service. The more available a system is, the more it meets customer demands. Businesses do not want every little change to one of the application components to cause an outage to the system.

Data mappers make use of identity maps to store data from a database and they use the primary keys from the database as the keys for the identity map. An identity map is an in-memory object of keys and values (key-value table) that can be created in object language programming. The keys ensure that each instance of a row is loaded into the map once. Martin Fowler [page 198] gives the following code as an example of how a "people" Identity Map is created and used.

```
private Map people = new HashMap()
public static void adperson(Person arg) {
        soleInstance.people.put(arg.getID(), arg);
}
public static Person getPerson(Long key) {
        return (Person) soleInstance.people.get(key);
}
public static Person getPerson(long key) {
        return getPerson(new Long(key));
}
```

One of the challenges that need to be overcome in the data mapper is the ability for it to be able to know which objects have been inserted, deleted or updated and be able to cleanly persist them to the database. Unit of work is a behavioral pattern that is used to mitigate problems associated with large transactions that affect more than one table. Finder methods that pull data into maps also need to be intelligent enough to know how much data is needed to be pulled into objects so as to minimize the size of in-memory graphs of objects. This is accomplished through Lazy Loading, another behavioral pattern that is also discussed in the book.

While this pattern has an advantage of being independent from the database, its major disadvantage is the added layer of finder methods.

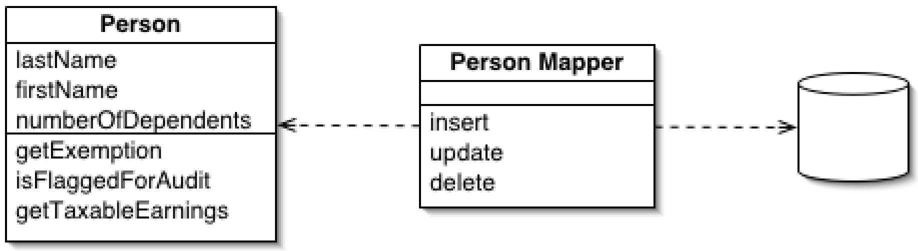


Figure 5 – Data Mapper

[Martin Fowler page 165]

Behavioral Patterns

These patterns are designed to support and reduce problems that are inherent in data source patterns. These problems include managing of transactions and managing of in-memory objects to avoid duplication and preserve data integrity.

1. Unit Of Work

This pattern allows tracking of changes and bundles them into smaller database transaction calls. The order of update, insert or delete as dictated by database table relationships, is also better managed by using the unit of work pattern. The advantages of unit of work include reduction of network traffic, elimination of expensive granular calls to the database and management of concurrency.

For objects to be part of a unit of work one of two types of registration has to be done with the unit of work. A caller may register objects to a unit of work or objects may register themselves through their methods to a unit of work. Omitting these registrations can cause problems but these are mitigated by using code generation tools.

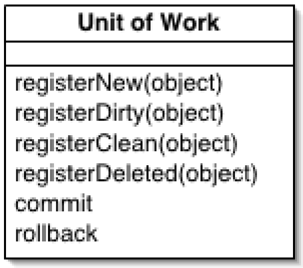


Figure 6 – Unit of Work

[Martin Fowler page 184]

2. Identity Map

This is the second behavioral pattern that “ensures that each object only gets loaded once by keeping every loaded object in a map” [Martin Fowler, page 195]. All data requests are first checked against the identity map before a request is made to the database. Identity maps can either be generic, for all the tables or explicit, one map per table. While generic maps have an advantage that you do not need to create a new map whenever a new table is added, their disadvantage is that you need database unique keys for the identity map.

It is recommended that identity maps be treated as session objects to minimize concurrency issues. These objects may also be registered with the unit of work so as to take advantage of all the good things that come with it. Identity map is also used for caching read-only and frequently accessed data. The Service-architecture web site describes caching under the realm of transparent persistence which means it makes use of identity maps and all persistence requests are handled in using the programming language.

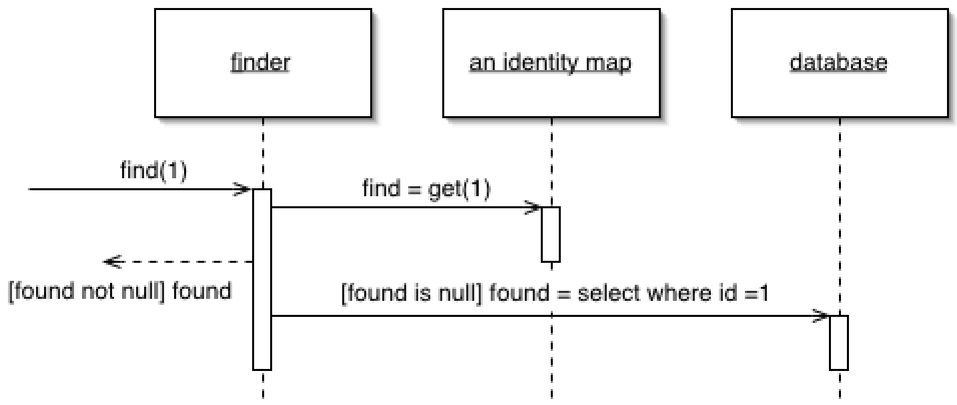


Figure 7 – Identity Map Sequence Diagram

The above diagram by Martin Fowler is a sequence diagram that shows requests for data going to the identity map (cache) first before attempting to retrieve from the database.

3. Lazy Load

This pattern is designed to improve performance and avoid loading of the entire object graph when the main object is loaded. Lazy Initialization is when you read a "high-overhead attribute such as a picture into memory" [Scott Ambler] when needed. The amount of data being pulled from the database determines whether to use Lazy Load or not. Its perfect implementation is when data is being pulled from different records or tables but not all of it is needed for the main object. Other names for lazy load include Lazy Read [Scott Ambler].

Structural Patterns

These patterns deal with the structural differences between in memory objects and database tables/rows. The patterns include Identity Field, Foreign Key Mapping and the three types of Inheritance Mapping namely Single Table, Class Table and Concrete Class Inheritance.

Identity Field

Database rows are identified by unique primary keys while in-memory objects are identified using internal memory addresses. A database primary key is not important to an object that is in-memory until it needs to persist the data to the database. The following are recommendations on how to use Identity Field:

- Use meaningless and immutable keys like sequence numbers to reduce problems that may arise when the meaning of a key needs to be updated. When something has a meaning, it is highly likely that the meaning may need to be changed in the future. An example of a meaningful key is last name and first name. It is not uncommon for people to change their names and each change will need to be propagated down all the relationships in the database. This may not be easily accomplished and sometimes it will involve first dropping the foreign keys and then putting them back after the update. However, a disadvantage of database-generated keys (also known as sequences) is that they require special proprietary handling when migrating (exporting or importing) or copying databases.
- Use of compound keys is also discouraged in favor of simple keys. Compound keys consist of more than one column to create uniqueness of a record. In most cases they tend to have business meaning and risk not being immutable.
- The use of table or database-unique keys may be determined by the application. While table unique keys are the most common, database-unique keys work well with generic Identity Maps. Other terms used for database-unique keys are OID (Object ID) factory and GUID (Global Unique Identifier). Key table [Martin Fowler] also similar to OID factory, can be

made to support table-unique keys by including a table name column in the key table which means that each table will have its separate row.

- The size of a key and the data type of a key may affect the efficiency of a key because most database systems build primary keys with underlying indexes for faster access. For this reason most developers prefer integer as opposed to character/string keys.

Foreign Key Mapping

As in the case of Identity Field, objects do not need to know database foreign keys as they have a way of referencing each other through memory addresses. Object relationships are implemented through "…references to objects or operations" [Scott Ambler]. The only time objects need to know foreign keys is when they need to send data to the database which checks foreign keys for referential integrity.

The main issue with Foreign Key Mapping in objects arises when collections are involved and there is need to persist an update, insert or delete to the database and it's not known what has changed in the collection. The simplest way [Martin Fowler] to handle it is to delete every thing and insert the collection again. Other ways of solving this issue involve comparing the collection with the database or keeping the original collection for comparison with the changed collection before writing to the database. The latter is more favorable because it does not involve another costly trip to the database.

An association table mapping is another form of foreign key mapping designed to address the many to many relationships in a database. These relationships are in the form of a join or associative table in the database.

Example of Foreign Key Mapping [Martin Fowler, page 238]

Consider the class diagram in Figure 8. A foreign key is created in the track entity to identify the album that it belongs to. Collections in objects result in multiple rows being created for saving to the database and in this case, each album will have more than one foreign key relationship into the track table.

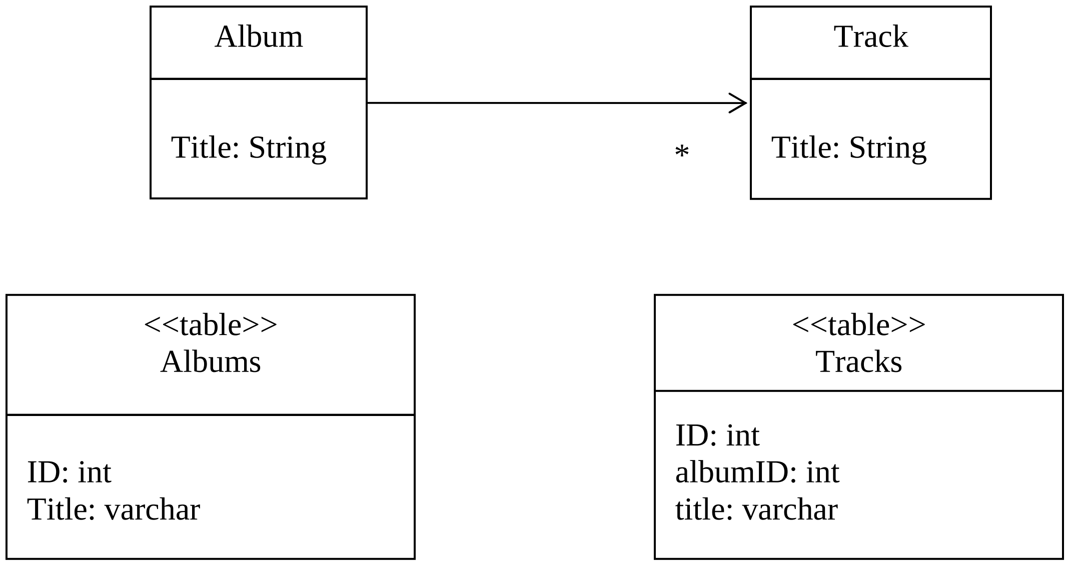


Figure 8 – Foreign Key Mapping

Single Table Inheritance

This pattern may be likened to de-normalization of data in database design language. This is a move from the normalized form of a database design which has less data redundancy due to the existence of relationships between tables. The cost of normalization is that it increases the need for using table joins when retrieving data that maps to a single inheritance structure and or collection in the program. Conversely de-normalization compresses data into fewer tables for performance.

When single table inheritance mapping is chosen, fields are promoted up the object/class inheritance tree. Inheritance mapping cannot be simply sustained in databases without incurring costs that are related to data retrieval through joins. It can be argued that databases do support inheritance but not in the way that objects support it or rather in the way that object programmers would like. This is supported through foreign key relationships but traversing these relationships is not as easy as it is in objects.

Issues associated with single table inheritance include:

- Not all fields are used all the time which may result in confusion
- Waste of space but it’s dependent on the database system
- The table has potential of being too large hence affecting performance

The following class inheritance is mapped into a single database table Person. The waste of space is very clear in the Person table as customers will not have a salary and or bonus and employees will not have preferences etc.

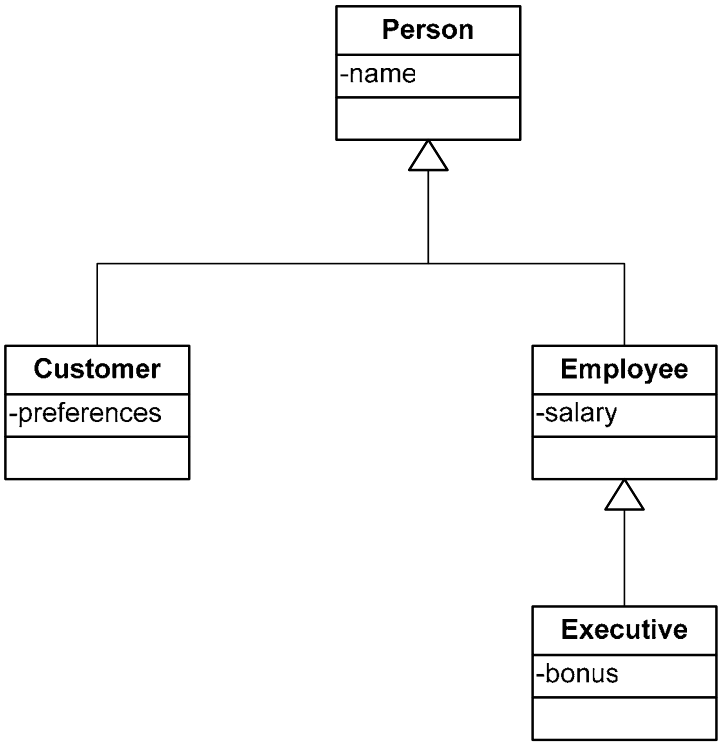


Figure 9 – Person Inheritance Structure

[Scott Ambler]

With single table inheritance, all the above classes are mapped to one table as follows:

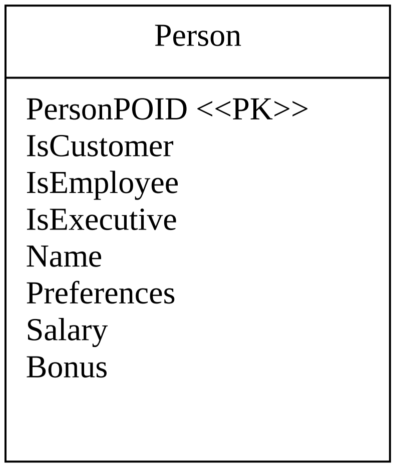


Figure 10 – Single Table Inheritance

[Scott Ambler]

Class Table Inheritance

Class table inheritance allows each class in the inheritance hierarchy to be directly mapped to one and only one table in the database. Ways to ensure that the hierarchy is maintained in the database include:

- Making use of one key across the tables for all related rows of an inheritance hierarchy.
- Use foreign key from the super class to the subclasses – each table will have its own primary key.

Class table inheritance has three main problems related to it.

i. This pattern is “highly normalized” [Martin Fowler] so one has to use SQL joins in order to bring all related data into memory as an inheritance graph. SQL joins always get inefficient as their number increases and most query optimizers do not do very well as soon as there are more than three joins.
ii. It is difficult to know which sub tables to read when you are looking for general information and the way around it is to use outer joins which are generally slow to execute.
iii. Changes to the domain model need to be mirrored to the database – the two are not independent of each other.

However, this pattern has advantages over Single Table Inheritance in that there is no waste of space through unused columns and it removes confusion by directly mapping to the domain model.

Concrete Table Inheritance

A concrete class is a class that is not abstract in nature. When given the name of a player one is almost certain to want to know what type of player is he/she and the answer might be cricketer – so a player is an abstract or non-concrete class.

In concrete table inheritance, each concrete class of an inheritance is mapped to one table. This pattern is also known as leaf table inheritance but Martin calls it concrete because some classes that are not leaf and are not abstract can be mapped to a table. Unlike class table inheritance, this pattern does not duplicate keys across tables but all other columns in the super class are duplicated in the subclass tables.

A major problem with concrete table inheritance is that of referential integrity where you cannot have a generalized class table (super class) associated with another class outside the inheritance structure. A player may participate in many charity events and a charity event may have many players participating. With concrete table inheritance there is no player table so each of the concrete tables, Footballer, Cricketer etc. have to be linked to the charity table, which is not possible in databases, or each will have its own association table like Cricketer Charity Function, creating clutter. Another issue with this pattern is that any field move from one class to another in the hierarchy tree, will require a database change. Duplication of the super class fields in each column means more places to make changes whenever a super class field changes.
Like class table inheritance, concrete table inheritance removes the large occurrence of unused columns. It also eliminates table joins when reading data and table access is only limited to the time when its class is accessed. The example of class inheritance [Martin Fowler, page 293] produces three tables using concrete table inheritance.

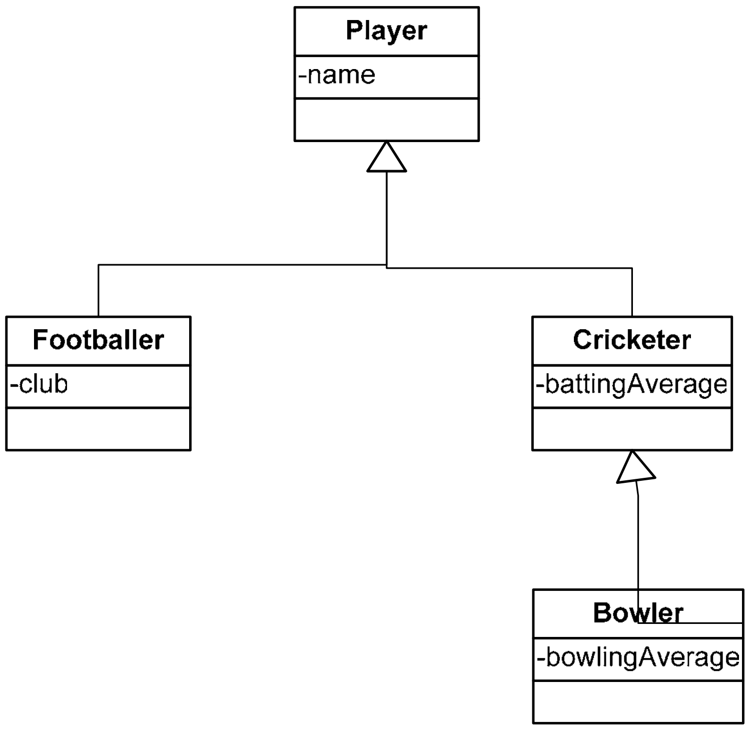


Figure 11 – Player Inheritance Structure

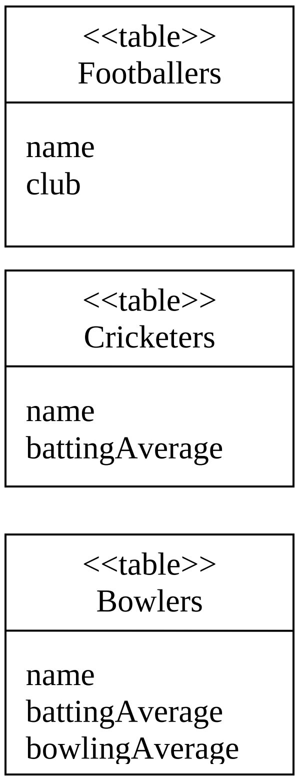


Figure 12 - Class Table Inheritance

Inheritance Mappers

This pattern consists of database mappers that handle inheritance hierarchies. Each concrete class and abstract class on the hierarchy has a mapper that knows how to find, insert, update and delete an object from the database. Inheritance mappers are needed to support all the inheritance patterns, Single Table, Class Table or Concrete Table Inheritance.

Summary

This chapter looked at the architectural patterns that are required to solve problems encountered in enterprise application development. Each of these patterns falls into at least one layer of the application architecture. Layering is a way of separating application tasks in order to improve performance, independence of components, change management, ease of application maintenance and security. The bottom line of all these improvements is high availability and customer satisfaction.

There is no rule of thumb on what patterns to use for an application. In most cases these patterns can co-exist and can be applied so that they are in support of each other within the same layer or across layers. Data source patterns are concerned with movement of data and they are complemented by behavioral patterns to ensure high performance and data integrity. Inheritance mapping patterns solve the problem of “impedance match” between object inheritance structures and database tables. These (inheritance patterns) can also be mixed within the same code – one does not have to use one and only one inheritance pattern.

Chapter 4 - Methodology

This project was carried out in three phases over a period of 24 weeks beginning in March 2005 and ending in August 2005. The project's approach was a waterfall model. The waterfall approach "… got its name from the way in which each phase cascades into the next (due to overlapping)." [E.M Bennatan page 64]. The first phase of the project needed to be completed before the second phase could start. There was still some overlap between the second and third phase because the writing of the paper started in phase II and ended in phase III.

Phases

Phase I Tasks

This phase included meetings with the project sponsor/advisor to define the scope, goals and deliverables of the project. A baseline of object relational mapping techniques and patterns was established by reading through Fowler's book and other resources on the internet. Next, questionnaires were sent out to a few hands-on experts of these techniques at work places to compare and contrast their object relational mapping patterns/techniques against the techniques established in the baseline.

Phase II Tasks

The main task of this phase was to design and develop the prototype using patterns identified in the literature. The author would also start and get most of the project paper written, documenting the findings of phase I of the project. The prototype design, class diagram and data model are found in Appendix B and C respectively.

Phase III Tasks

Tasks for this phase included completing the paper that was started in phase II and implementing a prototype of the object relational patterns from the design from phase II. This task would also culminate with the presentation of the project/thesis to Regis University.

Work Breakdown Structure

Task	**Sub-Task**	**From**	**To**
Phase I	Research	03/07/2005	04/29/2005
	Abstract	03/07/2005	03/11/2005
	Project proposal	03/13/2005	04/13/2005
	Extended outline	04/11/2005	04/29/2005
Phase II	Research Documentation	05/02/2005	06/24/2005
	Paper	05/01/2005	06/24/2005
	Prototype Design and Code	05/01/2005	06/24/2005
Phase III	Complete Paper	06/27/2005	08/26/2005
	Presentation	06/27/2005	08/26/2005
	Closure	08/26/2005	08/26/2005

Table 1 – Work Breakdown Structure

Resource Requirements

The following two people were involved with this project.

1. Student – Musafare Machisa

   The roles and responsibilities of the student were to conduct the research, document findings, design and code a prototype of the patterns learned. The student was also the project manager responsible for progress monitoring and setting up of review meetings with the project advisor from time to time.

2. Project Advisor – Robert Sjodin

   The project advisor was there to provide guidance and direction on the scope and expectations of the project. He also provided technical expertise in the design and implementation of the prototype.

Systems Development Life Cycle Model

Overview of the SDLC model/methodology

Rapid Prototyping is the SDLC methodology that was used in the development of the prototype. The goal of a rapid prototyping is to quickly develop and demonstrate the essential features of an application or product. This methodology/model is documented by Stephen R. Schach in his book “Object-Oriented and Classical Software Engineering” – Appendix A. The methodology was chosen because it was considered that this prototype was a proof of concept and unlike most prototypes, it will not evolve into a fully-fledged system or application. Specifically, it would not have phases like maintenance and retirement that are found in the model.

Prototype Design

The prototype is based on a bank account system. A bank account is a base class that represents either a checking or a savings account. This type of relationship is known as “IS-A” relationship because a checking account is a banking account as is a savings account. Another important feature of a bank account is that it is owned by at least one customer and a customer may also own several bank accounts. The class diagram in appendix B and data model in appendix C depict the mapping problems that encountered when doing object relational mapping.

1. The bank account class allows a collection of owners to be modeled inside the class. Collections are mapped into tables through one to many relationships and sometimes through association tables that resolve many-to-many relationship. Many authors of the object relational mapping subject discourage data model driven development because it constrains object structures and behaviors like inheritance and collections. Rather, they prefer domain-driven development as it gives developers the flexibility of designing objects without worrying about how the data will be laid out in the database. The final design approach for the prototype was data-driven in order to remove the need to use customer collections. This approach also allowed use of single table inheritance mapping that removed the checking and saving Account sub-classes.

Another reason for the approach was that it supports rapid prototyping making it quicker to write the code.

2. The bank account model was chosen for prototyping because it covers a number of issues that are discussed under the subject of object relational mapping. The bank account model contains hierarchies and other features like collections and many to many relationships that are the cause of the infamous “impedance mismatch” when trying to map objects and tables.

The importance of the two diagrams in the design is that models portray design patterns which in most cases make it possible to build systems that are scalable and easily maintainable. Models also provide good references when troubleshooting problems or planning redesign.

Prototype Development

A prototype was developed based on the data source layer of the enterprise application architecture. The importance of the data source layer is to provide a bridge between the domain objects and the database. Test programs were used to simulate the human or presentation interface to the data source layer and ultimately the database. The prototype would be able to accept data and persist it to a database and also read and display data from a database.

Patterns included in the prototype are:

1. Active record

   This pattern was chosen for the prototype because it is simple and convenient for small and less complex applications. This pattern has both data access and domain logic built in the object, a feature that does not provide for scalability in applications with large numbers of domains objects. The design started off as a Row Data Gateway but realized that that encapsulating the domain logic in the RDG would be appropriate for the size of this prototype as there would be no scalability issues involved in the future.

2. Single Table Inheritance

   This is represented by how the classes in a hierarchy are mapped to a single table in the database. The bank account class has two sub types namely the Checking and Savings account. Both these sub-classes exhibit one special attribute and one special operation/method. The pattern “collapsed” the base table, Savings and Checking account into one table with a type column that distinguishes the account types of savings/checking. The reasons for choosing to implement single table inheritance were:

   - The active record data source pattern requires both data access and domain logic to be in the gateway class and it does not easily support persistence of object hierarchies.
   - Trying to implement a different pattern would require taking the domain logic out of the active record classes and creating an extra layer of more code.

While single table inheritance allowed ease of programming, it also created a table that is quicker to retrieve data from without using expensive SQL joins that are a disadvantage of class table inheritance.

Prototype Testing

One test program was implemented for both the bank account and the bank transaction active records/gateways. Test areas included create, update and domain logic for deposit and withdrawal operations. The following test scenarios were created and successfully tested. Results of the testing are the appendix D.

1. Insert a bank account record in the bank account table in the database. Two types of accounts were created namely a checking account and a saving account. In each case the checking fee or the interest field would be empty and the type field would clarify what type of account it is.

2. The second test case was to update a bank account record, changing it from a savings to a checking or vice versa and also changing the owner of the account.

3. The third test involved creating a deposit and a withdrawal from an account. This case would update the bank account table and create records in the transaction table.

4. Lastly, a test was performed to find a bank account and related bank account transactions. This case also simulated the lazy load pattern by loading the bank account object into memory and then getting the transactions when needed.

Summary

This chapter covered the design, development and testing of the prototype application. The goal of the prototype was to demonstrate patterns that were studied during this project. The experiences from this phase contribute to the recommendations and conclusion in the coming chapters. Only two classes and tables were implemented in this prototype. The first design had proposed a highly hierarchical structure that included a customer table and other related tables like Customer address etc. Generating mappings for all these classes and tables was taking time so a decision was made to concentrate on the bank account and bank transaction classes (implemented as active records). The two classes still presented some challenges with respect to coding and implementation. More details of the prototype are covered later under the project history.

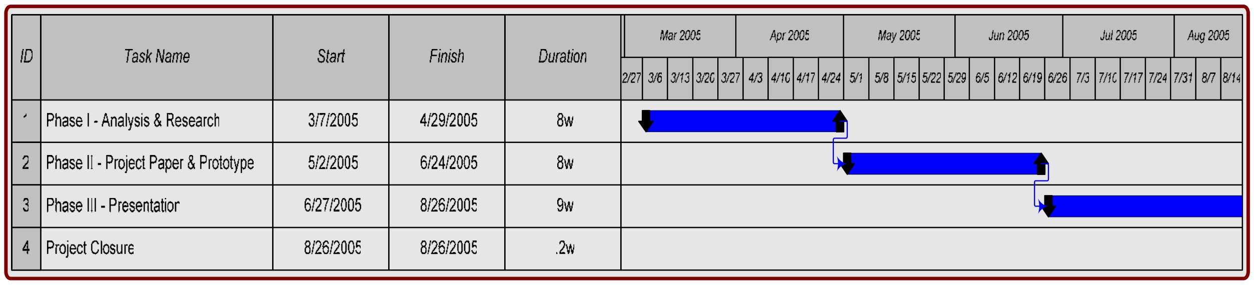
Chapter 5 - Project History

Project Initiation

The project idea was conceived by the project advisor while reading the Patterns of Enterprise Application Architecture book by Martin Fowler. The first meeting was held in early March to have an overview of the patterns and also to relate them to previous object oriented courses taken at Regis University for this program. The outcome of the meeting included the expectations and guidelines on how the research project was going to be conducted. Of particular interest was to learn, understand and prototype how these patterns are able to be implemented and maintain graphs of objects mapped to relational database tables.

Project Management

Microsoft Project was used to create a Gantt chart (plan) of the project. The whole project would span six months starting in March and ending in August 2005. A detailed table of the timeline is also provided under the project timeline topic.



ID	Task Name	Start	Finish	Duration
1	Phase I - Analysis & Research	3/7/2005	4/29/2005	8w
2	Phase II - Project Paper & Prototype	5/2/2005	6/24/2005	8w
3	Phase III - Presentatior	6/27/2005	8/26/2005	9w
4	Project Closure	8/26/2005	8/26/2005	.2w

Figure 13 – Project Gantt Chart

A project proposal was submitted and approved in April 2005 when the research had already started (March 2005).

Meetings were also regularly scheduled with the project advisor. The purpose of these meetings varied from status, review and technical guidance. The author would also make impromptu calls to the advisor whenever there were issues to be discussed.

Another tool used for managing the project was a project diary. This took the place of meeting minutes and status reports that would have been used in a large project. The diary allowed the author to track discussions, problems faced and any decisions that were made. For example, records of problems faced with the active record pattern while coding the prototype, the switch to row data gateway and back to active record again were all tracked in the diary.

Project Events and Milestones

Research

This was the preliminary task that had to be done in order to gain knowledge of the patterns, their advantages and disadvantages. In this phase the author was required to understand how each pattern works and to document the patterns implemented in the prototype.

Prototype Design and Development

The research milestone was required to be completed before moving to this task. The party/relationship model, a model of people and organizations was the first to be considered for prototyping. This model had many levels of hierarchy and it would ensure that a number of mapping patterns were included. The downside of this model was that it is very abstract and it would take time to explain and get it to work. A prototype was finally created and successfully tested using a bank account application/model. Periodic prototype scope reviews were done during this phase to ensure timely delivery of the prototype.

What Went Right

The most important thing that went right during the project is that the research and literature reading started early. Enough time was allocated for reading literature and comparing information from different sources. As it turned out, there was also need to re-read and refer to the literature once the paper and prototype were started.
The project also succeeded in making the author understand and appreciate the issues related to object relational mapping and the need for having these patterns so that time is not spent “re-inventing the wheel”. Although a few classes were coded, it was clear that this kind of work is time-consuming. It emphasized the need to use mapping tools for object relational mapping.

Another thing that worked well for this project is the periodic review of the scope of the project. Through this process, the author would meet with the advisor and make adjustments where it was necessary.

What Went Wrong

The following things went wrong during the project:

1. All the research methods did not work except literature. It was hoped there would be substantial information from experts through interviewing and questionnaires but both did not work well. This experience confirmed that time is always a constraint for interviews and questionnaires have a problem of not being responded to.
2. The prototype started too late in the project and that resulted in its scope being changed to small and simple. As a result there was panic during the prototype development when it became apparent that the programming was time-consuming and the project paper was not moving forward.
3. The Unit of Work was not implemented in the prototype due to two reasons:
   i. Primarily, time was running out for the project to be done. There was still a lot of work to be done on the paper.
   ii. The benefits of the unit of work were not going to be noticeable in this stand-alone application. Another way to ensure data integrity and reduction of database calls would have been to use the database's transaction unit capability. This involves starting a transaction and committing at the end when everything has succeeded. If an error occurs in between, the database would rollback the whole transaction.

Summary

This chapter looked at the project management, milestones, obstacles and successes of the project. The project management skills included work breakdown structure in order to come up with the project plan, project tracking through notes and reviews. Although there were a number of things that didn't go well, the objectives of the project were met. The research was completed and documented and a prototype application was developed using some patterns that were learned.

Chapter 6 - Lessons Learned

Lessons Learned

Carrying out this project exposed a number of experiences to the author whose main area of study was object oriented technologies. The author had the chance to act in different capacities at different stages of the project. These capacities included being project manager, researcher, designer and programmer.

The author learned that not all project management tasks are necessary for each and every project. This project was staffed with only one resource from the beginning and it was also carried out in a very short timeframe. There were times when the author was multi-tasking between programming and writing the paper. The author also lost some time during the early phases of the project. It seemed as though there was plenty of time but things started looking behind around the time the project paper course was starting. This was a lesson on the importance of keeping to the plan, managing oneself and never to allow time to slip.

Another lesson learned from carrying out this project is the experience of trying out different research methods in gathering input for this paper. It seems, to the author, that the most effective research method is reading literature because it is within the power and control of the researcher to make it work. The author thinks that it is advisable for research projects to spend more time on literature research while at the same time putting effort on other techniques like interviewing and questionnaires. Generally, people are not motivated to attend or answer research interviews or questionnaires.

Writing code for the prototype provided insight into the time required to write mapping layers between objects and relational tables. It confirmed the authors' recommendation for using commercial tools to do this type of programming work. Almost two weeks were spent trying to get the prototype with only two classes and a test program to work using the patterns. Most of the time was spent trying to interpret the authors' explanations and examples into working code.

Lastly, the project started with a much wider scope that was going to involve more patterns being studied and documented. The scope was later reduced to cover a couple of data source, structural and behavioral patterns so that the project could be completed in time. Project progress and scope reviews often reveal these kinds of problems when you are in the middle of the project. Reducing the scope and allocating additional resources to the project are ways that may be considered for solving problems related to wide scopes but there is no total solution. Sometimes both ways have to be employed.

Next Steps

This project will end at the end of the MSCIT program. However the study of this subject will go on as more opportunities get presented with new projects in the workplace. The author will find the lessons of this project very helpful in the day to day conversations with architects and developers.

Conclusion & Recommendations

From the reading that was done and the experience gained while creating a prototype of some of the patterns, it is clear that object relational mapping is a huge exercise in application development involving objects and database tables. The exercise is huge in the following aspects:

1. There is need from the developer's side to understand both object and database behavior. Most developers understand objects very well and have little knowledge about getting the best behavior and performance out of databases. As a result many hours are spent troubleshooting application performance problems due to poor mapping or poorly-formed access code.
2. There is also need for knowledge of the best practices and patterns that apply to different situations. "One size fits all" does not apply to all applications. Patterns help reduce time for development as they provide proven frameworks on how to accomplish tasks and sometimes a mix of patterns has to be employed for best results

The most effective way to solve object relational mapping problems is to make use of a relational mapping tool. Although there is no direct relationship between the number of domain objects and database tables, the general trend is that a large domain model results in more mappings. Therefore there is more value in using a tool when the application has a large domain model. Buying an object relational mapping solution should also depend on whether there is return on investment. A number of small projects may be used to add up towards return on investment for a mapping tool.

Summary

This chapter discussed the lessons learned during the project, most importantly the importance of project management and tracking. As identified in the Next Steps section, there will not be any further work on the project. The chapter ends with a conclusion and factors that influence a decision to buy or not buy a mapping tool. Object relational mapping issues will remain with us for as long as there is need to use non-object oriented databases. The patterns studied and documented in this paper will help speed up the mapping process and it is no secret that most mapping tools are built in adherence to these best practices.

Appendix A

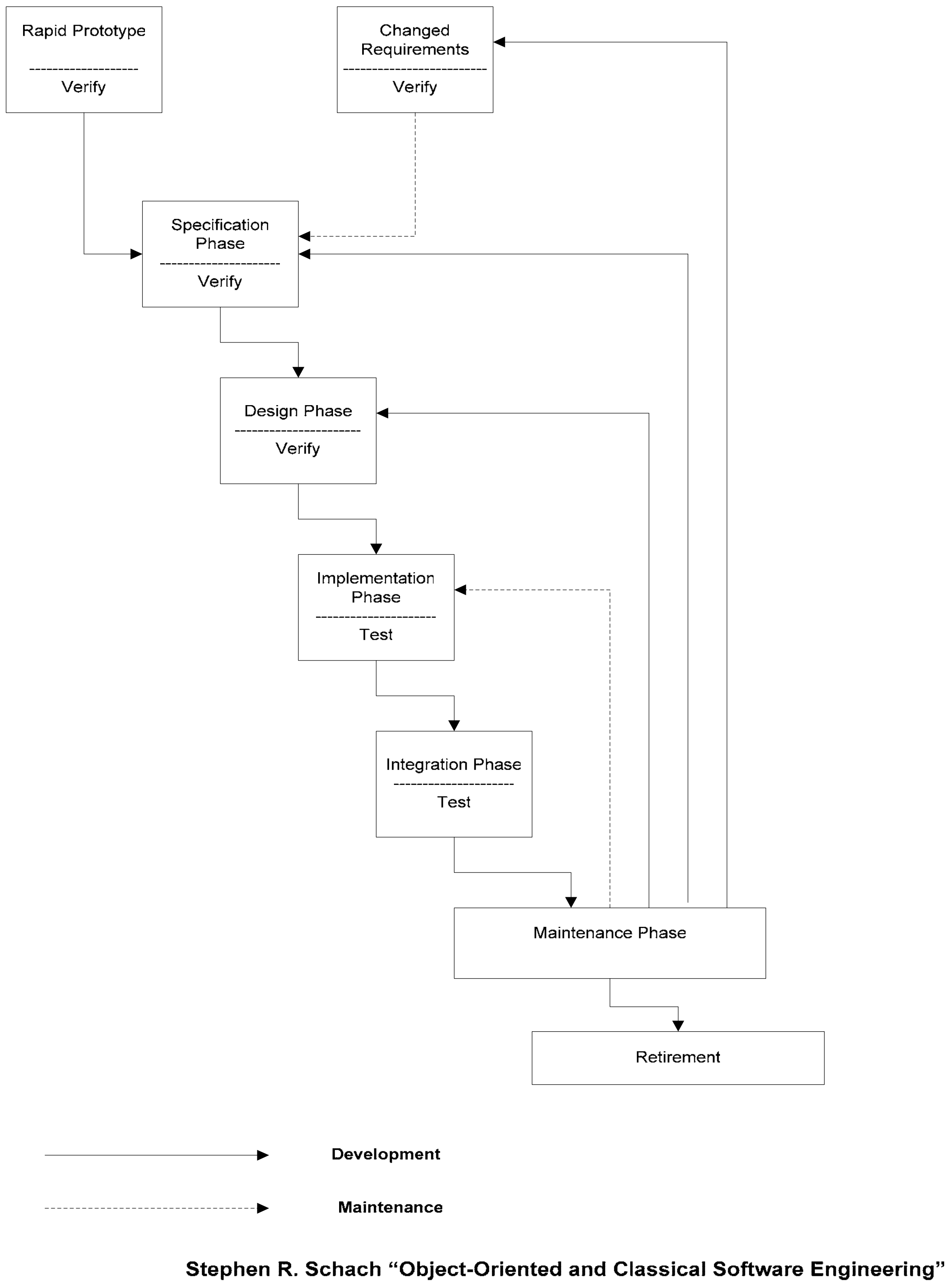


Figure 14 – Rapid Prototype Methodology

Appendix B

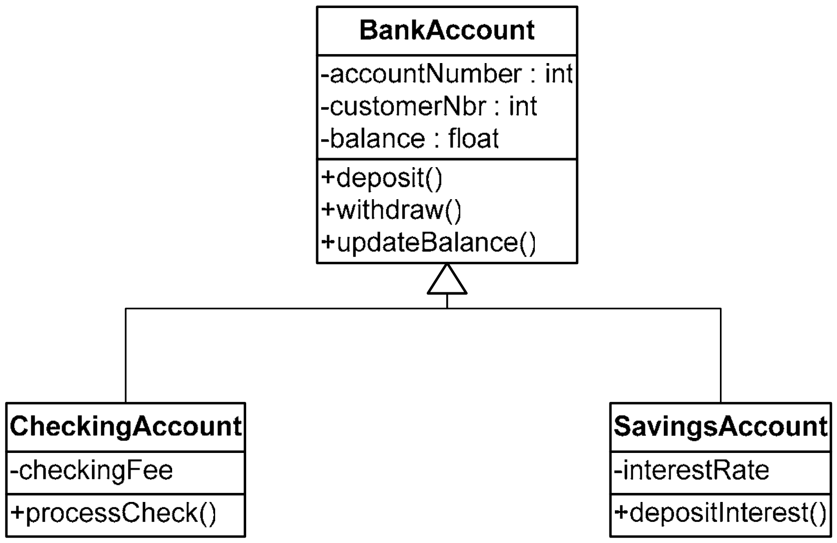


Figure 15– Bank Account Domain Class Model

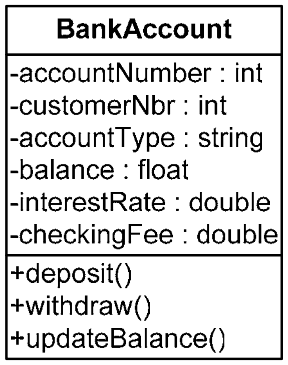


Figure 16 - Bank Account Active Record Model

Appendix C

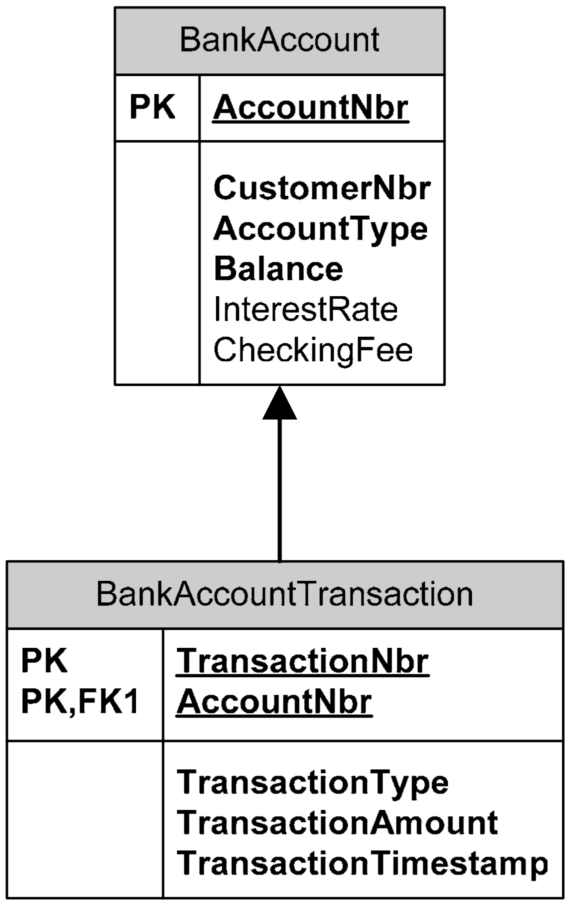
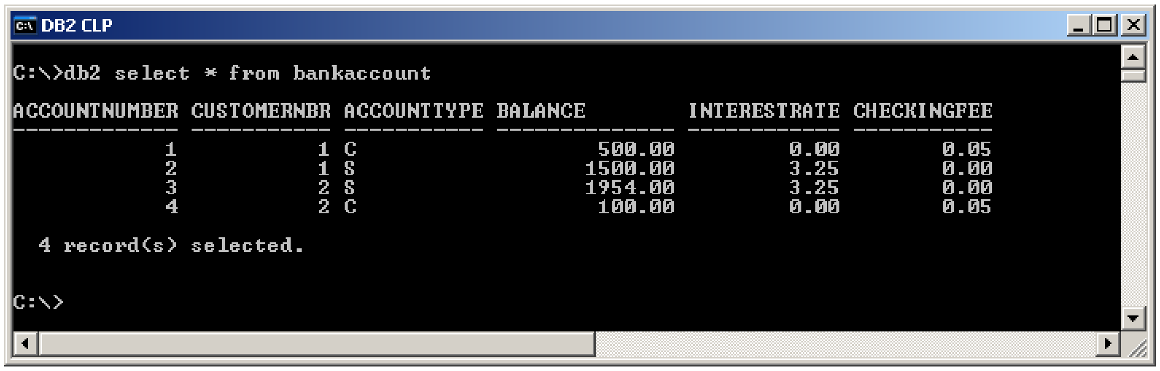


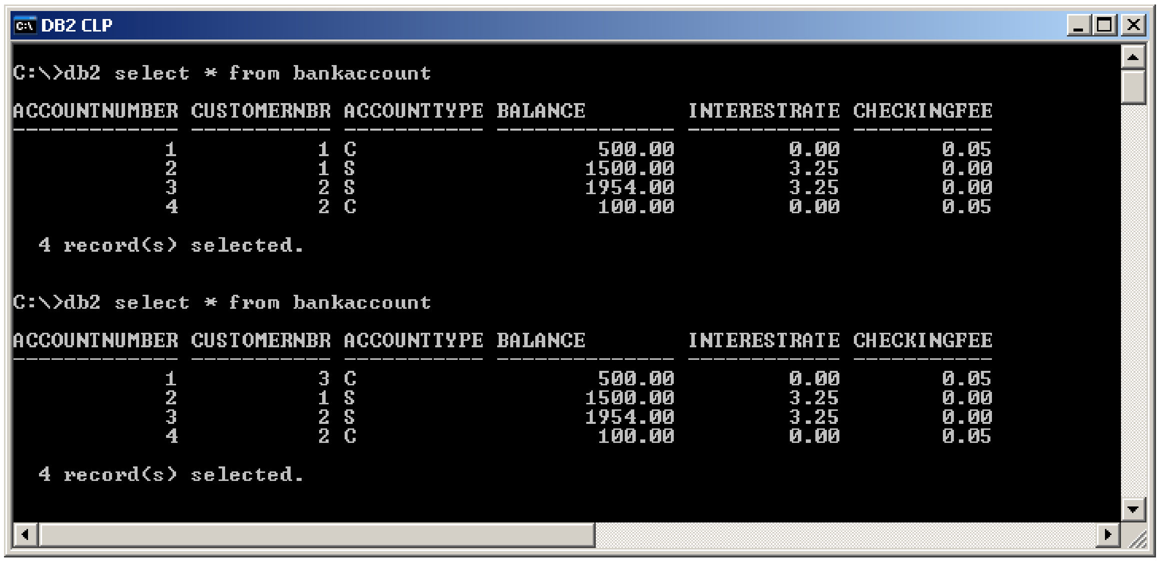
Figure 17– Bank Account Data Model

Appendix D - Prototyping Test Results



```
DB2 CLP
C:\>db2 select * from bankaccount

ACCOUNTNUMBER CUSTOMERNBR ACCOUNTTYPE BALANCE        INTERESTRATE CHECKINGFEE
------------- ----------- ----------- -------------- ------------ -----------
            1           1 C                   500.00         0.00        0.05
            2           1 S                  1500.00         3.25        0.00
            3           2 S                  1954.00         3.25        0.00
            4           2 C                   100.00         0.00        0.05

  4 record(s) selected.


C:\>
```

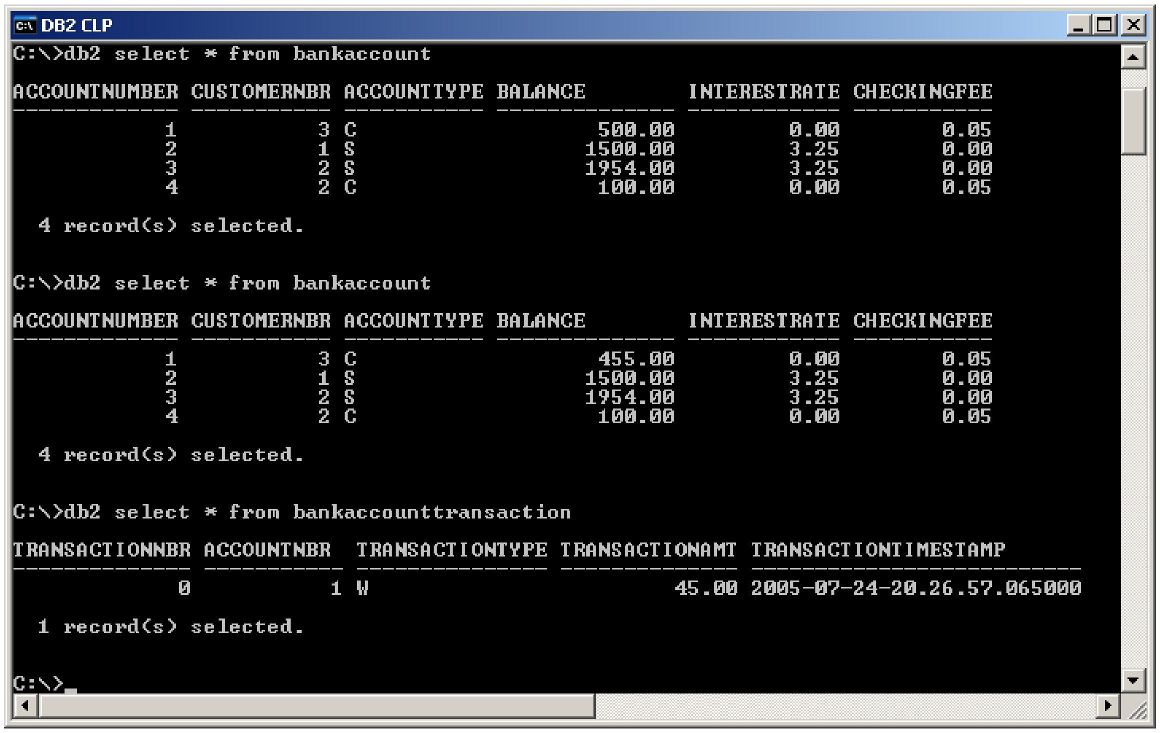
Figure 18– Create bank account



```
DB2 CLP
C:\>db2 select * from bankaccount

ACCOUNTNUMBER CUSTOMERNBR ACCOUNTTYPE BALANCE        INTERESTRATE CHECKINGFEE
------------- ----------- ----------- -------------- ------------ -----------
            1           1 C                   500.00         0.00        0.05
            2           1 S                  1500.00         3.25        0.00
            3           2 S                  1954.00         3.25        0.00
            4           2 C                   100.00         0.00        0.05

  4 record(s) selected.


C:\>db2 select * from bankaccount

ACCOUNTNUMBER CUSTOMERNBR ACCOUNTTYPE BALANCE        INTERESTRATE CHECKINGFEE
------------- ----------- ----------- -------------- ------------ -----------
            1           3 C                   500.00         0.00        0.05
            2           1 S                  1500.00         3.25        0.00
            3           2 S                  1954.00         3.25        0.00
            4           2 C                   100.00         0.00        0.05

  4 record(s) selected.
```

Figure 19 - Update bank account

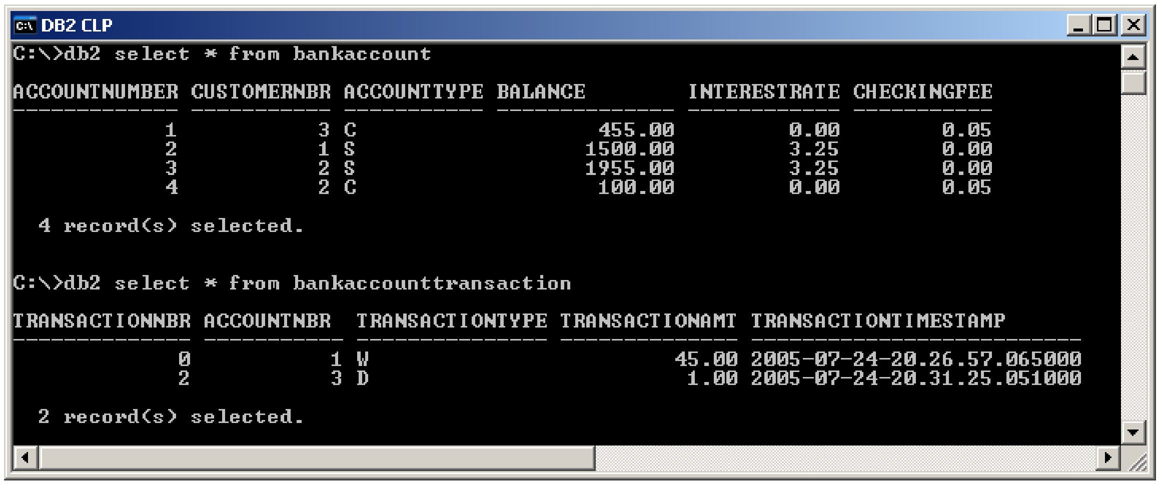
Shows account Number 1 changed from customer 1 to 3



```
C:\>db2 select * from bankaccount

ACCOUNTNUMBER CUSTOMERNBR ACCOUNTTYPE BALANCE        INTERESTRATE CHECKINGFEE
------------- ----------- ----------- -------------- ------------ -----------
            1           3 C                   500.00         0.00        0.05
            2           1 S                  1500.00         3.25        0.00
            3           2 S                  1954.00         3.25        0.00
            4           2 C                   100.00         0.00        0.05

  4 record(s) selected.

C:\>db2 select * from bankaccount

ACCOUNTNUMBER CUSTOMERNBR ACCOUNTTYPE BALANCE        INTERESTRATE CHECKINGFEE
------------- ----------- ----------- -------------- ------------ -----------
            1           3 C                   455.00         0.00        0.05
            2           1 S                  1500.00         3.25        0.00
            3           2 S                  1954.00         3.25        0.00
            4           2 C                   100.00         0.00        0.05

  4 record(s) selected.

C:\>db2 select * from bankaccounttransaction

TRANSACTIONNBR ACCOUNTNBR  TRANSACTIONTYPE TRANSACTIONAMT TRANSACTIONTIMESTAMP
-------------- ----------- --------------- -------------- --------------------------
             0           1 W                        45.00 2005-07-24-20.26.57.065000

  1 record(s) selected.

C:\>_
```

Figure 20– Test withdrawal

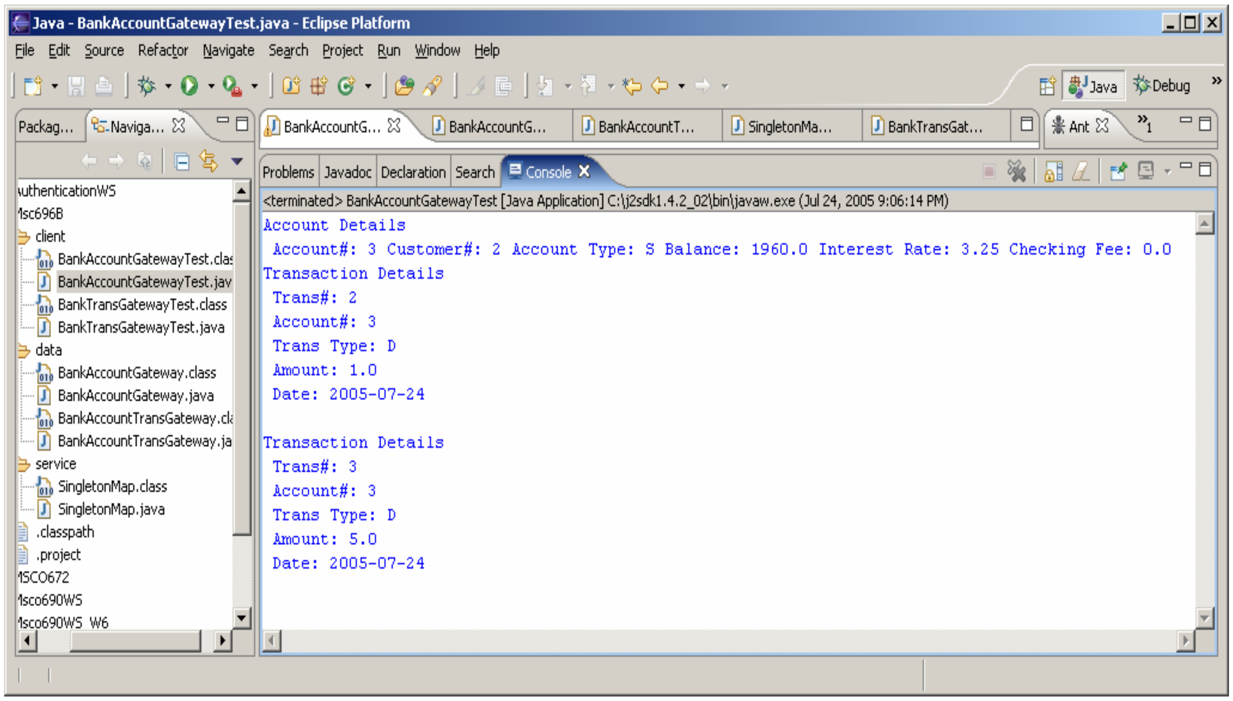
Withdraw $45 from account number 1



```
C:\>db2 select * from bankaccount

ACCOUNTNUMBER CUSTOMERNBR ACCOUNTTYPE BALANCE        INTERESTRATE CHECKINGFEE
------------- ----------- ----------- -------------- ------------ -----------
            1           3 C                   455.00         0.00        0.05
            2           1 S                  1500.00         3.25        0.00
            3           2 S                  1955.00         3.25        0.00
            4           2 C                   100.00         0.00        0.05

  4 record(s) selected.

C:\>db2 select * from bankaccounttransaction

TRANSACTIONNBR ACCOUNTNBR  TRANSACTIONTYPE TRANSACTIONAMT TRANSACTIONTIMESTAMP
-------------- ----------- --------------- -------------- --------------------------
             0           1 W                        45.00 2005-07-24-20.26.57.065000
             2           3 D                         1.00 2005-07-24-20.31.25.051000

  2 record(s) selected.
```

Figure 21 - Test deposit

Deposit $1 in account number 3



```
<terminated> BankAccountGatewayTest [Java Application] C:\j2sdk1.4.2_02\bin\javaw.exe (Jul 24, 2005 9:06:14 PM)
Account Details
 Account#: 3 Customer#: 2 Account Type: S Balance: 1960.0 Interest Rate: 3.25 Checking Fee: 0.0
Transaction Details
 Trans#: 2
 Account#: 3
 Trans Type: D
 Amount: 1.0
 Date: 2005-07-24

Transaction Details
 Trans#: 3
 Account#: 3
 Trans Type: D
 Amount: 5.0
 Date: 2005-07-24
```

Figure 22 – Test Find account and Lazy load

A find on account number 3 retrieves the base details about the account and also provide access to the transactions.

Bibliography and References

1. Martin Fowler (2003), Patterns of Enterprise Application Architecture, Addison Wesley
2. Barry & Associates, http://www.Service-architecture.com/object-relational-mapping/articles/index.html, Barry & Associates
3. Scott Ambler (2005), http://www.Agiledata.org/essays/mappingObjects.html, AgileData
4. Sun Microsystems, http://java.sun.com/docs
5. Hibernate, http://www.hibernate.org/
6. H.M. Deitel & P. J. Deitel (2003), Java™, How to Program, Prentice Hall
7. Craig Larman (2002), Applying UML And Patterns, An Introduction to Object-Oriented Analysis and Design and the Unified Process, PH PTR
8. Stephen R. Schach (2002), Object-Oriented and Classical Software Engineering, McGraw-Hill
9. Leavitt Communications, Inc. http://www.leavcom.com/db_08_00.htm
10. E. M. Bennatan (2000), On Time Within Budget, Wiley