New Patterns for Reducing Number of Access to Database in Layered Information Systems

GholamAli Nejad HajAli Irani
University of Bonab
Velayat Avenue, East Azerbaijan, Bonab
5551761167, Iran

Ali Akbarpour Bonab
University of Bonab
Velayat Avenue, East Azerbaijan, Bonab
5551761167, Iran

ABSTRACT
Nowadays, most information systems are developing based on layered architectures. Connecting to database is the most important part of layered architectures and there are many connections to database. So, the performance of information systems can be improved by reducing the number of such connections.

For this purpose, new patterns have proposed; also solution domain and structure of provided patterns have been explained by practical examples.

Finally, reusability and performance of provided patterns have been examined and the results approve the productivity of provided patterns in comparison with previous methods.

As a future work, a standard and reusable category of patterns will be reached by developing and categorizing other new patterns.

General Terms
Software Architecture, Layered Architectures

Keywords
Software Analysis and Design, Three Layer Software Architecture, Quality Attributes, XML.

1. INTRODUCTION
The usage of software systems is increasing in real-world applications; therefore, the size of storing and retrieving data is growing.

Data access performance optimization is the most important feature in system development. There are variety of optimization techniques based on the system size and its development.

Software architecture is an important issue in development of information systems. Optimizing the architecture affects the whole system optimization. One of the most applicable architectural patterns is the three-layer architecture.

This architecture is composed of the DAL (Data Access Layer), the BLL (Business Logic Layer) and the UI (User Interface) layers. The BLL consists of the majority of codes in three layer architecture. So, optimizing the BLL will affect the whole system optimization.

Due to characteristic of the BLL, the major part of Business Process (BP) is placed in the BLL. According to the anatomy of BP, any process for its operations requires storing and editing data. It is usually accomplished in the process body.

Any access to data in the BLL may cause access to database in the DAL. Any access to the database needs time. Furthermore, increasing the number of accesses to data in the BLL will increase the number of access to the database.

In this paper, to improve the productivity of the BLL, number of accesses to database has been reduced. To achieve this goal, the following steps have been recommended:

1. To investigate and categorize the problems of current approaches and architectures.
2. To develop and explain two new patterns for decreasing number of accesses to database.
3. To evaluate provided patterns and investigating their advantages.

2. RENIEW THE PROBLEMS OF EXISTING SOLUTIONS
In most object-oriented methodologies, each system is composed of a set of Use Cases, which is identified as Process or Core Asset in the other methodologies.

Based on UCP (Use Case Point) methods [1], system Use Cases, can be divided into main and non-main Use Cases. The main Use Cases have more access to data than the non-main. In order to optimize the number of accesses to data, optimizing the main Use Cases, will improve the whole system productivity. For example, the Borrow Use Cases of library system have evaluated. Alternative flows of these Use Cases are similar to Figure 1.

In the Use Case of Borrow, the sent data from the UI to BLL includes BNO (book's number) and MNO (Member ID). For BLL optimization, many architectures and methods have been presented.

For example, Martin Fowler in [2], investigates the institutional architectures of system each of which leads to optimize the system development. The architecture of Oracle Suite [3] writes all BLL codes into database functions for reducing the number of accesses to data.

Microsoft's architecture used in [4], writes all DAL-codes into database's Stored Procedures in order to reduce the number of access to data.

Table 1 classifies available methods to decrease number of accesses to data in database. All methods can be divided into two major categories. First category uses Stored Procedures or Functions of Database and second category doesn't use them.

Table 1: classification of current approaches

First category (by Stored Procedures)	To decrease number of access to database and improve system performance, all of BLL layer codes must be written in database such as [3], [4].
Second category (without Stored Procedures)	To decrease number of access to database and improve system performance by object oriented patterns, techniques and heuristics such as [2].

The first category of methods implies the BLL codes into database. Furthermore, less time is required to access data, but the code inside the database does not have common programming language strengths, because its structure is not fully formed.

The BLL codes include major part of project. So it is not easy to write BLL codes into database. For example, languages that used in the database such as PL-SQL [5], TSQL [6] and SQL*PLUS [7] do is not as powerful as Object-Oriented. Writing large amounts of code without object-oriented and other programming language techniques, neglects other capabilities such as extendibility, modifiability and reusability. Furthermore, the methods of database languages are not recommended for large software systems.

In the second group of approaches, using design patterns and object-oriented heuristics such as [2] which are trying to optimize and reduce the number of access to data in BLL and DAL.

None of the existing approaches are ideal for optimizing the number of accesses to the system. In some references, such as [9], one or two methods have been provided. But none of them have been investigated comprehensively.

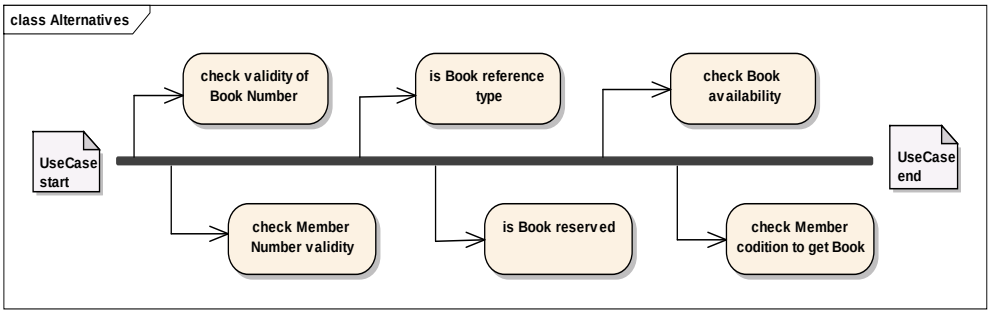

Fig 1: Alternative flows of Borrow Use Case

3. PRESENTED PATTERNS

In this section, new patterns are provided for reducing number of accesses to data in the BLL. Following steps are considered for each pattern:

1. Describing the problem domain

2. Describing the solution domain

3. Describing the structure of pattern

4. Supporting the provided pattern by practical example

3.1 Multi Command Pattern (MCP)

3.1.1 Describing the problem

In Use Case implementation, multiple requests or multiple data from database may be required. Usually, these requests apply from the BLL and using the DAL is unavoidable to access each data for each request. These requests may occur for storing and editing in database system.

3.1.2 Describing the solution

The purpose of this pattern is storing and editing the data in the DAL when these data do not interact with each other. This pattern is usable to request all commands from the DAL in one time, and the DAL returns all data and results to the BLL in one time.

For example, in library system, consider the Borrow Use Case; the Alternative Flow is presented in Figure 1. Different entities that are in relationship with Borrow Use Case have been showed. For Use Case operations, member, borrow, document and book entities must be modified.

It is essential to access DAL for each request in usual methods. Furthermore, it is required to connect to database for each request in usual methods. But, in this pattern, all operations can be performed just in one access to database.

In order to implement this pattern, the class named Multi Command Entity (MCE) in the DAL is required. Every entity (like Book Entity) is composed of commands such as select, insert or update. And it is possible for user to use these commands in the BLL. Thus, MCE class should support all methods of system entities.

3.1.3 Describing the structure of pattern

Using all methods of entity classes in this pattern are possible by adding additional methods without any change in methods of entities; the purpose of these methods is creating and returning the relevant SQL command without executing it.

For example, consider the Insert method from Book's methods. The responsibility of this method is getting data and creating SQL command and executing it. Naming these new methods are following this format: previous method name_MCP(…).

These new methods have same parameter as previous methods. In addition, the task of these new methods is same as previous ones, but without executing the command. Also BLL should be replaced like Table 2.

As a result, the MCP class in the DAL takes all SQL command by Add method and storing it in a list. Then by Run method, all commands should be sent to execute into database. In order to execute sent command by MCP class, the method named MCP_SP is required into database.

This method's duty is getting and executing all SQL commands and returning the result(s).

In order to execute the set of sent command by MCP class, MCP_SP method must be create into database. As regards that sent commands may have different state of CRUD; the result of executing output may have different state. For simplicity, XML format is used for sending data to database. The DTD of MCP_SP result will be like Table 3.

Table 2: replacing previous codes by applying MCP pattern

Previous method	New method

Book.Insert(BookInfo b);	MCP x=new MCP();
	x.add(Book.Insert_MCP(BookInfo B));
	x.Run();

Table 3: DTD of MCP_Sp result

```
<?xml version="1.0"?>
<!DOCTYPE DTDsend [
<!ELEMENT MCP_Sp (Response+)>
    <!ATTLIST MCP_Sp
          ResponseCount CDATA #REQUIRED >

<!ELEMENT Response (FieldNames,Records)>
    <!ATTLIST Response
          No      CDATA     #REQUIRED
          Type    CDATA     #REQUIRED
          Name    CDATA     #REQUIRED >

<!ELEMENT FieldNames (Name+)>
   <!ATTLIST  Name
                Value   CDATA       #REQUIRED >

<!ELEMENT Records (Row+)>
    <!ELEMENT Row (Col+)>
        <!ATTLIST Col
                    Value  CDATA      #REQUIRED> ]>
```

3.2 One Many Pattern (OMP)

3.2.1 Describing the problem

In some Use Cases, because of one-to-many relations between system entities, entity's data or recorded data may be required several times. In this case, the primary key will be used in storing other entities.

For example, in Accounting Systems, for storing data more than two references to entities may be needed, meaning that more than two times access to database is required. For example, entities like Figure 2, for inserting data of A, B, and C entities, more than three accesses is required to access data of entities in database. This problem can be even more sophisticated.

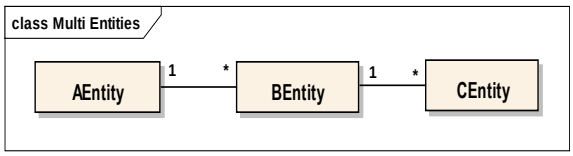

Fig 2: example for one many relationship

3.2.2 Describing solution

For problem like that, OMP pattern minimizes the number of accesses to one access to database. The problems like this problem are caused by using the primary key of entity A in other entities like B and C. For this reason, entity B should be started after recording data of entity A, and this is the reason that they could not send all data from the BLL in one access to database. Furthermore, for solving this problem and sending more than one inserting command for A, B and C entities in one access to database, the following instructions are required in provided OMP pattern:

1. Getting commands from OMP entity class in DAL.
2. Using OMP_PK key in entity B instead of primary key of entity A.
3. Sending commands to function named OMP_SP into database and inserting data of entity A by OMP_SP function.
4. Using primary key of entity A instead of OMP_PK key for entity B inserting command.
5. Replacing new codes in BLL according to Table 5.

This pattern output is similar to MCP pattern. Namely, the output of more insert commands by OMP can be returned by DTD of Table 3 to the BLL.

Table 5: replacing previous codes by applying OMP pattern

Previous	New

AInfo a = new AInfo();
a.Attr1=Value1;
a.Attr2=Value2;
PK=Aentity.Insert(a);
BInfo b=new BInfo();
b.FK=PK;
b.attr1=Value1;
Bentity.Insert(b);
	AInfo a=new AInfo();
	a.Attr1=Value1;
	a.Attr2=Value2;
	OMP Entity x=new OMP Entity();
	PK= x.Add(Bentity.Insert_MCP(A));
	BInfo b=new BInfo();
	b.FK=PK;
	b.attr1=Value1;
	b.attr2=Value2;
	x.Add(Bentity.Insert_MCP(b));
	x.Run()

For two or more than one many relations like Figure 2, all commands can be perform just by one access to DAL by OMP pattern and the codes must replace according to Table 5.

4. EVALUATION

In this section, the performance of one of provided patterns (OMP) have evaluated. The presented pattern decreases the access numbers to database to one. However, using this pattern adds new codes and depletes performance. Minimizing the access number is worthy in expense of depleting the performance.

The Borrow Use Case of library system have been examined by normal coding and with new provided pattern for diversity of data in both formats. Results are illustrated in Figure 3.

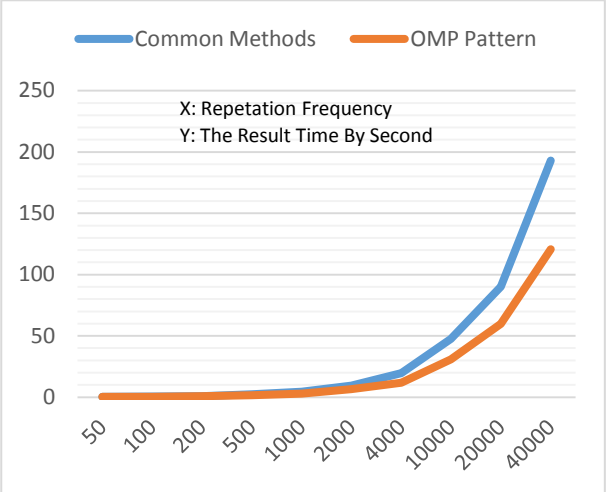

Fig 3: performance comparison between OMP pattern and common methods

Examination codes exist in [10].

5. CONCLUSION AND FUTURE WORK

In this paper, two patterns named MCP and OMP have been provided for reducing the number of access to database. Patterns have been evaluated and it is obvious that system productivity improved without any additional overhead. Therefore, these patterns can be used to improve system productivity in any system.

By providing new patterns or extending the existing ones, new Framework can be produce for provided patterns. By creating Framework for provided patterns, system productivity can be improve in any project.

6. REFERENCES

[1] S. Diev, Number 2, 2006.Software Estimation in the Maintenance Context, ACM SIGSOFT Software Engineering Notes, Volume 31.

[2] M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, R. Stafford, 2002. Patterns of Enterprise Application Architecture, Addison Wesley.

[3] A. Passi, V. Ajvaz, 2009 Oracle E-Business Suite Development & Extensibility Handbook, Oracle Press.

[4] Microsoft Patterns & Practices Team, 2009 Microsoft Application Architecture Guide, Second Edition, Patterns and Practices, Microsoft Press.

[5] M. Rosenblum, D. Delmolino, L. Cunningham, 2011 et al, Expert PL/SQL Practices: for Oracle Developers and DBAs, Apress.

[6] D. Comingore, D. Hinson, 2007. Professional SQL Server 2005 CLR Programming: with Stored Procedures, Functions, Triggers, Aggregates, and Types, Wiley Press.

[7] J. Gennick, 2005. Oracle SQL*Plus: The Definitive Guide, O'Reilly Press.

[8] B. Meyer, 1994. Object Oriented Software Construction, Second Edition, Prentice Hall International Series in Computer Science.

[9] J. Goodson, R.A. Steward, 2009. The Data Access Handbook Achieving Optimal Database Application Performance and Scalability, Pearson Education.

[10] Full source codes are available online at: http://www.4shared.com/zip/YYyqCRb4/OMP-Final_Source_Codes.html