MVSDAP: a new extensible, modifiable and secure data access pattern for layered Information Systems

GholamAli Nejad HajAli Irani[1], Vali Tawosi[2]

[1] Faculty of Engineering, University of Bonab
Bonab, 5551761167, Iran

[2] Electronic and Computer Faculty, Tarbiyat Modares University
Tehran, Iran

Abstract

Nowadays, in software architecture especially in agile methodologies dynamicity, extendibility and modifiability are important challenges. In three layered architecture, plenty of patterns are provided for data access layer. In this paper additional to secure access to data and dynamic validation controls, emphasis is on extendibility and modifiability of data access pattern. We tried newly provided pattern not to lose performance. In this new pattern, respecting to object oriented heuristics, a module along with a tool has been provided that is able to be attached to every project and perform all data access tasks. This pattern has maximum reusability so it can be used in different kinds of projects of any size regardless to the methodology used.

Keywords: *Layered Software Architecture, Quality Attributes, Object Oriented Design, Data Model.*

1. Introduction

Nowadays, extension and modification is innate portion of software systems and often projects in every stage of development and even after deployment and at runtime need to be extended or modified. Extendibility and modifiability of a system still is a great challenge for software architectures. Software production line is one of main activities in software engineering which received a huge attention in research [5]. Creation of a software product line that is able to produce new software based on stack holder's requirements, using reusable pre made platforms and a great ability of extendibility and modifiability, is the main concern in software product line [8]. Software product lines use a variety of architectures. Most of existence architectures are based on three layered architecture that is obtained from Model-View-Controller (MVC). Three layered architecture is consist of user interface layer (UIL), business logic layer (BLL) and data access layer (DAL).

Our proposed approach for software product line specialized for information systems, is a three layered architecture. In this approach for developing each layer an extendable and modifiable framework is proposed that each layer can manage all of its duty. As presented in Fig.1 all of system use cases will be performed in "Workflow Engine Framework".

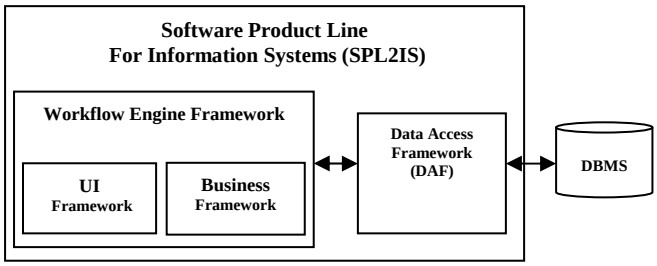

Fig. 1 Suggested Framework for Software Product Line

All components of project for fetching and storing data use "Data Access Framework" (DAF). DAF does all task pertaining to database. For developing DAF we suggest an architecture which is shown in Fig.2. This architecture composed of three main modules including MVSDAP, MVSTools and DBManager.

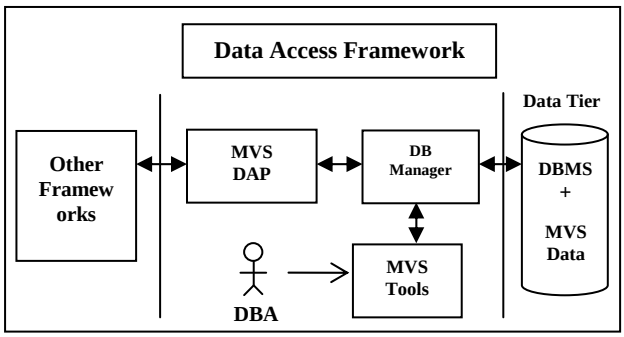

Fig. 2 Suggested Architecture for DAF

DBManager connects us to database and encapsulate implementation details of database connection. Instead of

DBManager one can use available stand alone modules like Hibernate. There is a list of such modules in [9, 10, 12]. [13] has a comparison between some of them. Our aim in this paper is to provide a new pattern called "MVSDAP" and its subsidiary tool called "MVSTools" in companion with a "DBManager" does all of data access tasks.

Firstly we are going to investigate existence data access patterns and then propose our new pattern with more extendibility, modifiability and better security in comparison with existence patterns. We had to follow the steps shown in the following to provide our new pattern.

1. Provide a Meta-Model or Architecture to DAL operations.
2. To collect and categorize all previous methods and patterns.
3. To investigate and obtain parameters of extension and modification in DAL.
4. To investigate and obtain parameters of security and validation in DAL.
5. To provide a new pattern and describe its modules.
6. To analyze extensibility and modifiability and security of previous methods.
7. To describe extensibility and modifiability of provided pattern.
8. To describe control and validation of input values
9. To describe security of provided pattern
10. To evaluate and compare quality attributes of provided pattern.

2. Previous Data Access Patterns

So far, plenty of data access patterns have been provided for information systems. In the literature there are out to 50 patterns with different names provided for data access layer. After an investigation in to these patterns we reached to a general categorization in a way that every pattern is falling in one of these categories.

DAP0: patterns with no DAL. In these patterns (if even can call it a pattern) all of BLL classes do their operations to DBMS by themselves. Advantage of this method is performance which may be used in real time systems. An instant of this category is introduced in [6] by the name of Transaction Script.

DAP0Sp: this is DAP0 with this difference that BLL classes instead of connecting directly to the database use Stored Procedures to fetch data and alter them. One can use some control and security principle in stored procedures. This pattern enjoy a high performance, however in enterprise information systems where problem domain is very vast and there are complicated use cases, using this methods will make a big problem in development process.

DAP1: it encapsulates access to the database. In this pattern, using one or more classes in data access layer, implementation details like database name, server name, user name and password are kept hidden from upper layers. Now all of BLL classes use these classes to communicate with database. There are methods defined in DAL classes to perform requests coming from BLL classes including create, read, update and delete (CRUD) operations. This pattern is like "Metadata Mapping" introduced in [6]. In this pattern an interface between BLL and DAL has defined supporting all data access methods and for each database type a class has been created and inherited from that interface [7].

DAP1T: same as DAP1 except that this kind of pattern has two methods (we call them "Begin" and "End") in order to support transaction. These two methods are used to start and finish transactions. CRUD operations are executed after a transaction has started (by calling Begin method) and at last it will be finished (by calling End method). The whole transaction will be done (commit state) or rejected (rollback) if there were errors. In the case of rollback, relevant error message will be presented to user and all changes to database (affiliated by this transaction) will be rolled back.

DAP2: in this pattern for each table in project, a class will be created (we call them "Entity" classes) and CRUD operation for that table performed by its class. Advantage is that one can put syntactical and access control on each table separately. This pattern is introduced as "Table data gateway" in [6].

DAP2F1: for setting better control and stronger security and also respecting to modularity principal, this pattern has some improvements to DAP2. In order to give permission to an entity to select data from other entity classes, this pattern define an "Finder" class for each entity class and gather them in a common layer (accessible to all of entity classes). Finder is responsible for selecting data from its respective entity and serves all of entity classes [6].

DAP2F2: same as DAP2F1 except that we have just one general Finder class able to select data from all entity tables and serve all entity classes [7].

DAP2F1Sp and DAP2F2Sp: implemented with stored procedures (for higher performance) and DAP2T: supporting transactions.

With the same concept DAP2F1T, DAP2F2T, DAP2F1SpT, DAP2F2SpT can be defined.

DAP3: similar to DAP2, this pattern is using entity classes for tables. The difference is that respecting to object oriented heuristics all CRUD methods are implemented in a common class and all entity classes inherit these

methods from that. Adding support for transactions, implementing with stored procedures and separating Finder classes from entity classes are possible in this pattern to define new patterns. “Object-Relational Metadata Mapper” pattern introduced in [6] falls in this category.

DAP3M: same as DAP3 except that all of the Meta-data of the tables and fields are stored and the parent class which all entity classes are inherited from that can check validity of table names, view names, field names and data type and so on, using this Meta-data. In this pattern parent class is able to create CRUD executable statements dynamically.

DAP4M: unlikely DAP2 and DAP3 this pattern doesn’t create an entity class for each table. In order to improve extendibility and modifiability this pattern uses Meta-data (like DAP3M) and defines some general classes in DAL to do all tasks. In this pattern when a BLL class wants to do a CRUD operation, sends all information (table name, fields name and values and action) to DAL and then validity of information are checked. If all information were valid, an executable statement (depending on request) has made dynamically and execute. “Layer supertype” pattern introduced in [6] is like this pattern.

2. MVSDAP Pattern

Major headings are to be column centered in a bold font without underline. They need be numbered. "2. Headings and Footnotes" at the top of this paragraph is a major heading.

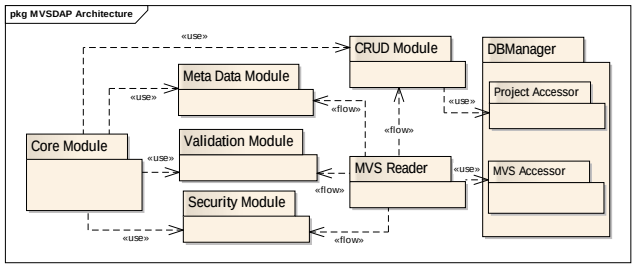

Fig. 3 MVSDAP Architecture

MVSDAP is consisting of six main modules.

1. Core Module: Manages all operations performing by pattern. In other words this module is just a coordinator and all of actions will performed by other modules and controlled by this module.
2. Meta Data: Stores all of information about project itself consisting of information about all tables, views, attributes, etc.
3. Validation: Stores and handle all of information needed for data access layer validation checks.
4. Security: Manages access level for each entity and its related data.
5. MVS Tools: We call three recent modules as MVS. Handling these modules will be possible by means of an application which we call it “MVS Tools”.
6. MVS Reader: Since MVS information is stored in database, executing every transaction needs to fetch some data from database and use in provided pattern modules. For overcome this time consuming communication and increasing performance “MVS Reader” is added to model. After storing Meta Data, Validation and Security information in database by database administrator using MVS Tools, as soon as the project starts to run this information will be fetched from DB and will be available for other modules.

CRUD Module: This Module receives CRUD operations from core module, makes the executable statement dynamically and executes that using DBManager.

4. Static Aspects: MVS Data Model

In this section we are going to define a data model which will be stored by MVS Tools, read by MVS Reader and serve other modules.

4.1. Meta-Data Data Model

This part holds schema information including the information of tables, views and their ID’s. In favor of supporting different schemas in different databases and that may be project had composed of several modules, there is a Module class in our model. Every module has several tables or views each having names and aliases. Each table or view has several fields which we can store their information in “Field”. As different databases have different data types, this model include a “DataType” class for storing and managing data types. Meta-data data model is shown in Fig.4.

4.2. Validation Data Model

In this part all validation checks of data are kept and managed. Based on 3 layer architecture, we can only set validation of data access layer but no business rules. Hence we separate validations into two categories. First category consists of syntax errors which check input data syntactical and manage its relevant errors. As an example of validations which falls in this category are isNumber, isString, inRange, etc. Second category is of referential consistency; they are called semantic errors.
From Fig.5, Every field has several syntax errors. For example a field can have three validation checks: “isNotEmpty” means field is not allow null; “isInt” means field’s value should be integer and “isIntRange” means

that field's value not only should be integer but also should be in a specific integer range. DBA can set an special error message for each of checks that will be presented to user when validation check is not successful. "MethodName" attribute will hold the name of the function which will be called to perform validation check.

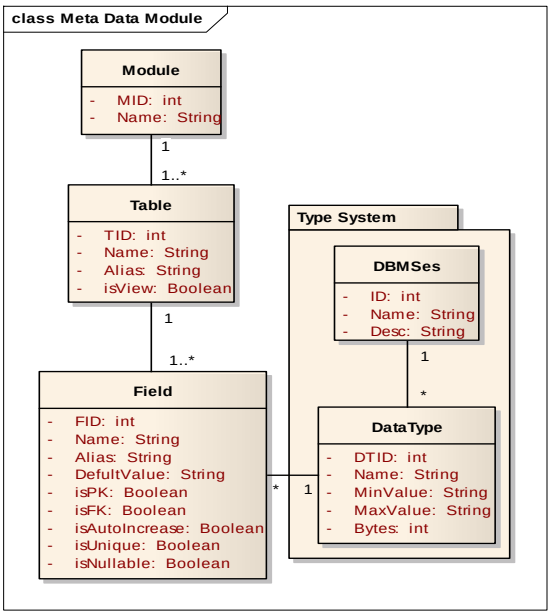

Fig. 4 Meta-Data Data Model

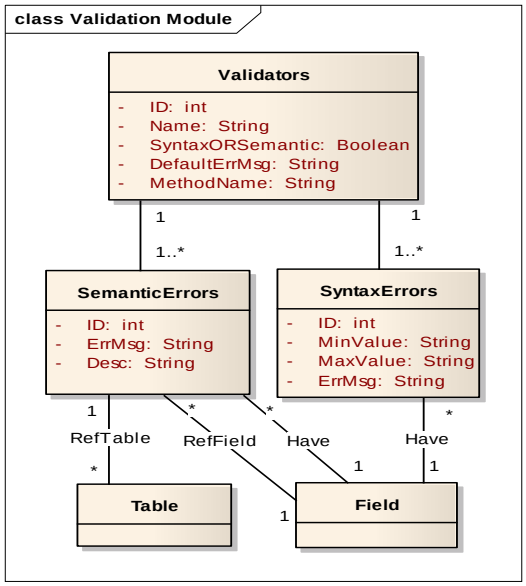

Fig. 5 Validation Data Model

4.3. Security Data Model

In this part information about authorization rules for each table are kept. Authorization rules indicate that witch operations on witch tables are allowed. Since variety of security models is vast, we will describe just "insert" operation (Fig. 6). For each table we can define several insert operations in which data insertion for every attribute is allowed and its error has defined separately. For example every attribute in every insert statement can have a state like authorized for set value, set as null or set default value. Such a model can be provided for other statements like select, update and delete.

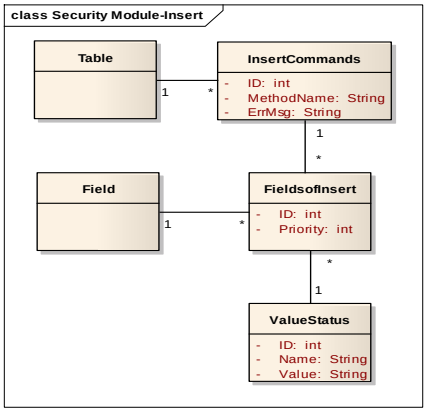

Fig. 6 Security Data Model

4.4. MVS Tools

To manage all MVS information by database administrator, an application is needed that we call it MVSTools. Certainly this application must have 3 main parts for Meta-Data, Validation and Security settings. In each part of software some heuristic can be used that database administrators can easily work with this software. For example we can get Meta-Data of any project from database itself. All meta-data of project is saved in a part of database called Schema_Information. Validation part of software is shown in Fig.7. All of the codes of this software (except security module) is available at [14].

```
try {
    MVSDAP Em  =  new MVSDAP();
    Em.setEntity("Book");       // Em.setEntity("EntityName");
    Em.Fields["Title"]="C++"; //Em.Fields["FieldName"]="Field Value";
    Em.Fields["Pages"]=null;
    Em.insert();
}
catch( Exception e) {
    MessageBox.show(e.Message);
}
```

Fig. 8 Insert sample code

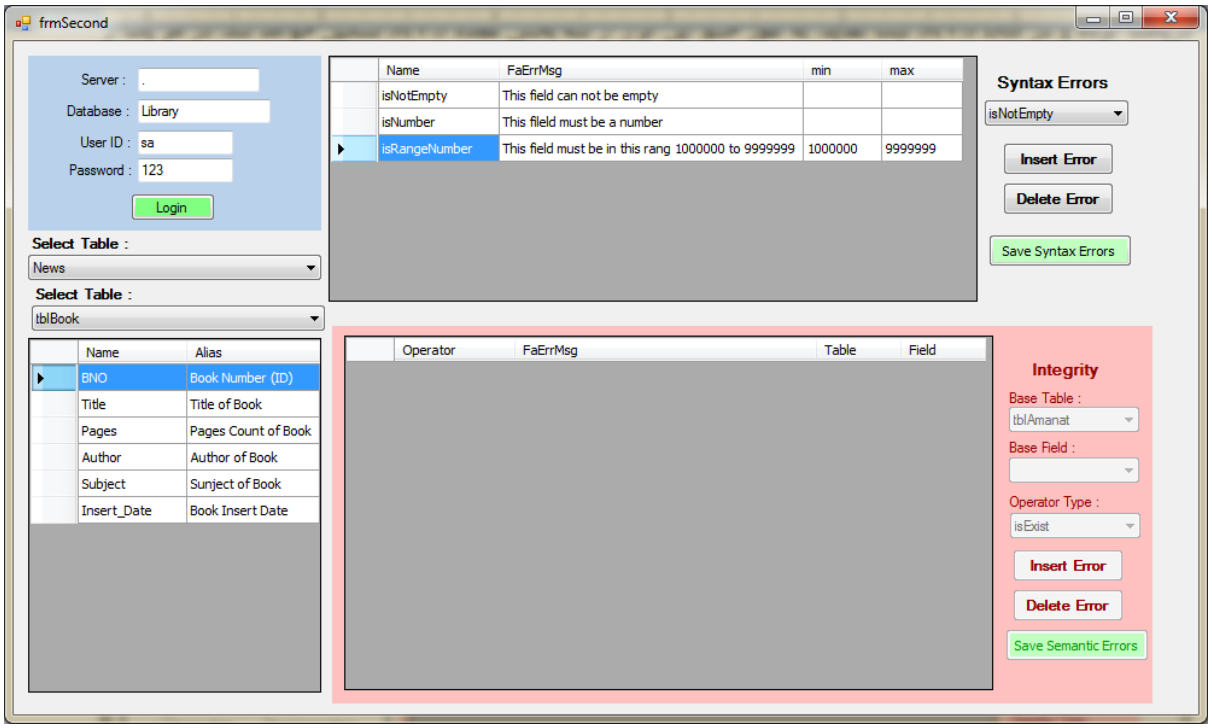

Fig. 7 Validation part of MVS Tools

5. Dynamic Aspects

To describing dynamic parts of system, Insert scenario has been selected. Fig.9 shows the full sequence of codes which is shown in Fig.8. Consider that LoadMVS method loads all MVS information as soon as project starts to run, and put it in MVSReader module.

6. Evaluation

Provided pattern by adding new features to data access layer makes changes in quality attributes of this layer. Extensibility, modifiability, security and performance are quality attributes that influenced by new pattern. These topics will be discussed in the reminder.

6.1. Extensibility and Modifiability

To compare extensibility and modifiability of provided pattern with other patterns, effective parameters in extensibility and modifiability can be used. First of all, the parameters must be gathered and categorized. Next parameters list must be prioritized based on degree of importance. Then the degree of simplicity of applying extensibility and modifiability of each pattern must be calculated. Let Pi as effective parameter in extensibility and modifiability and Ei as degree of importance of each pattern and Deg(Pi) as degree of simplicity of applying of each pattern, then degree of extensibility and modifiability of each pattern can be obtain by formula 1.

$$Deg(DAP_x) = \sum_{i=1}^{Count(P)} (E_i \times Deg(P_i)) \qquad (1)$$

Parameters of extensibility and modifiability in data access layer can be categorized in table 3. In this table, only those parameters have been listed that satisfies three conditions. First, extension and modification of that parameter occur in the system frequently. Second, extension and modification of that parameter influence source code of project. Third, the parameter must not have similar effects in all of compared patterns. For example changing the Username and Password of a database rarely occurs in system development. Therefore only those patterns have been added to table 3 that have influence source code of project and frequently occur in the system.
On the other hand, the degree of importance of some patterns depends on type of project and expertise of project developer team and some other things. So some ambiguous parameters have been eliminated from table 3.
On the other hand, from all categorized pattern only DAP3M and DAP4M have been developed with extensible approach and other patterns have not. So in the comparison, MVSDAP has been compared just with DAP3M and DAP4M.

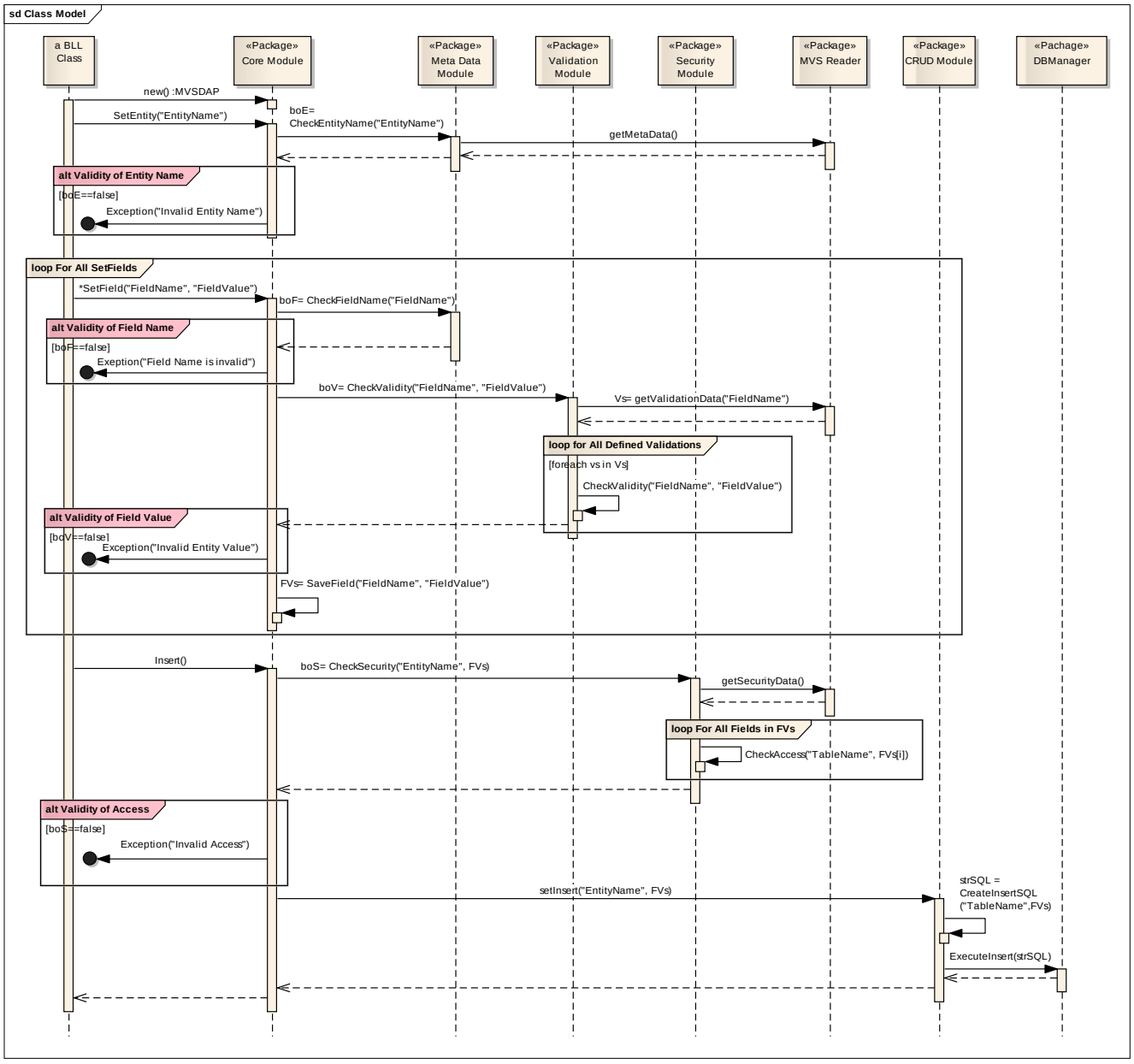

Fig. 9 Sequence of Invocations in Insert Method

To calculate the degree of simplicity of each pattern, source codes of each pattern have been investigated comprehensively. Based on analysis of source codes of each pattern, each extension and modification can be one of the following scenarios in table 2.

Degree of applying extensibility and modifiability in state 3 is very small and this action performs so quickly rather than other states. So simplicity degree of applying state 3 can be set to 1. For state 1, if change alters just one part (block) of code, the degree can be set to 4 and if changes alters several parts of code, the degree can be set to 10. For state 2, if change alters just one part of code, the degree can be set to 10 and for several parts of code, the degree can be set to 25. This degree can be considered as a unit of time. Note that maybe the expertise of database administrator and data access developers can affect these degrees, so multiplying these degrees with appropriate factor can be turn to unit of time. For example if factor of 10 has been considered to high experienced developer then change in one part of code takes 4×10 seconds.

All results of extensibility and modifiability have been shown in table 3. In this table D1 is Deg(DAP3M), D2 is Deg(DAP4M) and D3 is Deg(MVSDAP).

Based on comparisons and results of table 3, extensibility and modifiability of provided pattern is significant.

Table 2: Steps of extensibility and modifiability in different patterns

State1: Steps of change in code with IDE	State 2: Steps of extend code with IDE and Generator Application	State 3: Steps of modify and extend with MVS Tools
1. Open IDE application	1. Open IDE application	1. Open MVS Tools application
2. Change code and apply with DBA	2. Add new written or generated codes	2. Apply changes or extensions
3. Debug modified codes	3. Debug modified codes	3. Save project in MVS Tools
4. Compile DAL layer	4. Compile DAL layer	
5. Replace DLL or JAR file in project	5. Replace DLL or JAR file in project	

Table 3: Extensibility and Modifiability Parameters in DAL

Category Name	Parameter Name (P)	E(P)	D1	D2	D3
Parameters of extensibility and modifiability in a Database	Change Database	.1	4	4	10
	Change Table	1	4	1	1
	Insert a new Table	.5	10	1	1
	Delete a Table	.5	4	1	1
	Add a new Field into a Table	.7	10	1	1
	Delete a Field from a Table	.7	4	1	1
	Change a Field Name	.5	4	1	1
	Change Auto Null ability	.4	1	1	1
	Change Field Type	.3	4	1	1
	Change Field Length	.5	1	1	1
	Change Field DEFAULT value	1	1	1	1
	Change Field to/from Primary Key	.5	1	1	1
	Change Field Identity Type	.7	1	1	1
	Change Field to/from Unique ability	1	1	1	1
	Change Field to/from Index ability	.7	1	1	1
	Change Field to/from Foreign Key	1	1	1	1
	Change View Name	1	4	1	1
	Insert a new View	1	10	1	1
	Delete a View	.7	4	1	1
	Delete a Field from a View	1	4	1	1
	Add a new Field into a View	1	10	1	1
	Change Alias in a View's Fields	.7	4	1	1
Parameters of extensibility and modifiability of Security	Add Insert permission to a table	1	10	25	1
	Add Insert permission to each fields of a table	1	10	10	1
	Change Insert permission to a table	.7	4	4	1
	Change Insert permission to each fields of a table	.7	4	4	1
	Add Update permission to a table	1	10	25	1
	Add Update permission to each fields of a table	1	10	10	1
	Change Update permission to a table	.7	4	4	1
	Change Update permission to each fields of a table	.7	4	4	1
	Change permission of condition in Update action	.5	4	4	1
	Add Delete permission from a table	1	10	25	1
	Change Delete permission from a table	.7	4	4	1
	Change permission of condition in Delete action	.5	4	4	1
Parameters of Validation	Add a new validation to a field	1	10	25	1
	Change a validation type of a field	.7	4	4	1
		Result :	144.6	156.6	27.6

6.2. Security

Security of database and data access layer performs in different levels. Controls of permissions to a database and its Tables and other information can be managed with DBMS. But the security that has been added in this paper is defining different part of security that cannot be managed by DBMS. To perform the additional type of security, some codes must be written in Stored Procedures or Functions or Triggers in DBMS or in projects code in Entity Classes. In all cases extensibility and modifiability of code is reduced. But in the new provided pattern all security features perform dynamically and has high quality of extensibility and modifiability.

Table 4: Provided Parameters in Security of DAL

Category Name	Security Parameter
Access to Insert Action	Control insert action into a table
	Control insert action into a field
	Control insert NULL value into a field
	Control insert DEFAULT value into a field
Access to Update Action	Control update action in a table
	Control update action in a field
	Control update a field to NULL value
	Control update a field to DEFAULT value
	Control condition of update based fields values
Access to Delete Action	Control delete action in a table
	Control condition of delete based fields values

6.3. Performance

Performance of data access layer depends on many factors that have been widely investigated in [11]. Extensibility and modifiability of a system have trade-off with performance of system. In this paper extensibility and modifiability features have been added to many part of data access layer. So this action reduces the performance of system in general. So to prevent excessive loss of performance, many object oriented heuristics have been used in implementation of MVSDAP. Accordingly, a static method called MVSLoad written in MVSReader that immediately called after first running the project and this method retrieve all data related to MVS from Database and saved in MVSReader. After this time there is no need to retrieve data for all MVS functionalities. So the time of reading all MVS data in MVSLoad add to total performance of system.

Now to insert a new record in database, based on codes in Figure 8, in line 2, firstly we must check the existence of Table Name through Meta-Data. Then to perform this action there is no need to retrieve from Database. Because all MVS data retrieved before in MVSLoad. So the time of these checks will equal like a simple linear search, O(Tc) that Tc is the number of all Tables of project. In line 3, 4 and 5, input values get and validate with MVS data. Normally these validations perform in any projects but in this case, data of validation search in MVSReader. This time is Ac*O(Rc) that Ac is count of Fields and Rc is count of records in Syntax Error Table.

In line 6, Insert method is invoked. Firstly permission of Insert check with Security Module and this time is O(Ic) that Ic is count of records of Insert Commands Table. If permission is true, all data of Insert send to CRUD Module. In this module firstly SQL Command will create then Insert operation executes. Creation of SQL Command and execution exist in any projects. So all additional time of this pattern for Insert method is shown in formula 2.

$$T_{insert} = O(Tc) + Ac \times \frac{}{O(Rc) + O(Ic)} \quad (2)$$

As regards Ac, Ic, Rc and Tc are constant for any projects then the overall time is a small number. Also the MVSLoad method is calculated once time for any projects.

7. Conclusion and future works

In this paper a new extensible and modifiable pattern (MVSDAP) for data access layer has been provided. In addition, data security in Tables, Views and Fields level are major achievements of this paper that performs in high level of extensibility and modifiability.

This pattern (except Security Module) has been implemented and used in many projects and performance of extensibility and modifiability of it experimentally has proved.

With the full implementation of this pattern, database administrators can dynamically manage all data access layer functionalities. On the other hand extensibility and modifiability is an essential principle for some of methodologies such as agile methodologies like XP and Scrum. So MVSDAP as a dynamic DAP can be used in any three layered and agile projects.

Implementation of Security Module is one of the future works of this paper. After full implementation, it can be used in proposed framework in part 1 (introduction). The idea of this paper can be used in other scope of computer programming. For example we can combine this paper with Modular thinking and provide a modular extensible data access layer. By adding XML (eXtensible Markup Language) as a protocol of communication, this pattern can be used in Service Oriented scopes.

8. References

[1] Gomma, H., Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures, Addison Wesley, 2004.

[2] Greenfield, J., Short K., Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools, John Wiley & Sons, 2004.

[3] Crawford W., Kaplan J., J2EE Design Patterns, O'Reilly, 2003.

[4] Lhotka R., Expert C# 2008 Business Objects, Apress, 2009.

[5] Carnegie Mellon University. The Software Engineering Institute, Software Product Line, available online at: http://www.sei.cmu.edu/productlines.

[6] Martin Fowler, David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, Randy Stafford. Patterns of Enterprise Application Architecture, Addison Wesley, 2002.

[7] Clifton Nock. Data Access Patterns: Database Interactions in Object-Oriented Applications, Addison Wesley, 2003.

[8] Klaus Pohl, Günter B♡ckle, Frank van der Linden, Software Product Line Engineering, Springer, 2005.

[9] Java Persistence Layer Source Codes, available online at: http://Java-source.net/persistence.
[10] C# Persistence Layer Source Codes, available online at: http://Csharp-source.net/persistence.
[11] John Goodson, Robert A. Steward. The Data Access Handbook, Achieving Optimal Database Application Performance and Scalability, Pearson Education, 2009.
[12] George Reese. Java Database Best Practices, O'Reilly, 2003.
[13] Roland Barcia, Geoffrey Hambrick, Kyle Brown, Robert Peterson, Kulvir Singh Bhogal. Persistence in the Enterprise: A Guide to Persistence Technologies, IBM Press, 2008.
[14] MVS Tools, Full source code available online at: http://www.4shared.com/rar/fTXhRwkB/PayaMDT.html