Research on an Efficient Software Framework for Developing

Guoxiang Zhou, Ruirui Cao, Lei Shi*, Yun Hu
School of Computer and Information, Hefei University of Technology, Hefei, Anhui, China.

* Corresponding author. Tel.: +8655162901397; email: shilei@hfut.edu.cn
Manuscript submitted August 31, 2018; accepted October 10, 2018.
doi: 10.17706/jsw.13.11.613-620

Abstract: With the continuous development of information technology in society, the demand for software continues to increase. An excellent software development framework can greatly improve the efficiency for programming, and can also improve the software portability and maintainability. We propose an optimized software development framework, which we called the MOCV (Model-Operate-Controller-View) framework. The MOCV framework is a four-layer structure framework. It is based on the traditional three-tier architecture and the MVC (Model-View-Controller) design pattern, but it can make the relationship between layers in the MVC design much clear and can decouple the complexity of layers in the three-tier architecture. In this paper, we discuss in detail the idea and the design of the MOCV framework. In simulation, starting from the five major indicators such as the "maintainability index" and the "circle complexity" of the code, we analyze a project example developed using the MOCV framework, which shows the MOCV framework's enhancement of software development efficiency, portability and maintainability.

Key words: Three-tier architecture; MVC; software development framework.

1. Introduction

Nowadays with the development of computer and information technology, software have become very large and complex. To design these complicate software, the develop frameworks for software design have also become more and more complex. Therefore, many researchers proposed the concept of universal software design framework and built some different general software developing models. A good software development framework can not only reduce the difficulty of software design, but also make the post-maintenance simple.

The idea of a three-tier architecture and the design patterns based on Model-View-Controller(MVC) have been widely used in the field of software development at present. The three-tier architecture is a layered structure used in the design of software frameworks, which can effectively manage various components and make the components more efficient and synergistic in working order. Specifically, the three-tier architecture refers to the presentation layer, the business logic layer, and the data access layer. In detail, the presentation layer is responsible for collecting information from the client, sending it to the business logic layer for processing, and receiving the processing result from the business logic layer, and finally displaying the result to the user. The business logic layer is responsible for receiving input from the presentation layer, executing the designed business interactively with the data access layer, and eventually sending the processing results to the presentation layer. The data access layer is responsible for data access and data maintenance[1]. By using the three-tier architecture to develop software, the developers will only focus on one layer of the entire structure. And these will reduce the dependency between layers and layers on the whole software framework, furthermore, these are also beneficial to the reuse of all layers of logic[2]. Therefore, the three-tier architecture has been widely studied and applied. For example, In [3], the authors proposed a template based on a three-tier architecture on the .NET platform, which reduces the developer's workload; In [4], the authors proposed an extension point for each layer of the three-tier architecture to improve the reusability and flexibility of the software framework; In [5], the authors proposed the concept of data interface and data pool on the basis of the three-tier architecture to solve the cascade phenomenon existing in the three-tier architecture and improve the performance of the system.

In order to further reduce the degree of coupling between the presentation layer and the business logic layer, many researchers have added the MVC design idea to their studies. MVC refers to the Model, the View and the Controller. It is a kind of software design pattern in software engineering, and its purpose is to realize the functions' division of Web system. Among them, the Model layer is responsible for the business logic in the system and the View layer is responsible for interacting with the user. The Controller layer is served as a bridge between the Model layer and the View layer. It can dispatch user's requests and select appropriate views for display, and can also interpret user's input and map them to actions that the Model layer can perform. For example, In [6], the authors proposed an idea that combines the MVC design pattern with the three-tier architecture which has a good effect in solving the problem of high coupling between the presentation layer and the business logic layer; In [7], the authors proposed a concept of separating the business logic layer from the MVC Controller layer, which makes the system more clear and concise; In [8], the authors proposed a five-tier design pattern, based on the MVC design pattern, to reduce the coupling degree between the presentation layer and the business logic layer.

The ideas proposed above are only improved in some aspects of the three-tier architecture or the MVC design pattern. By decoupling the layer from the layer, the development framework is clearer and the efficiency of the development is improved. However, we stand on the whole of the development framework, jump out of the current concept of using MVC as the presentation layer framework, and use the MVC as the framework main body while incorporate the layered idea of the three-tier architecture. Finally, we propose a software development framework ——Model-Operate-Controller-View(MOCV) framework. The framework has clear structure and clear hierarchy, and solves the problems of low development efficiency, portability and maintainability of the three-tier architecture and MVC design pattern in enterprise-level development. It can provide reference for the development of software systems.

The second section describes and analyzes the details of the MOCV framework and the process of evolution from the basic idea. The third section applies the MOCV framework to a practical project development, designs the overall structure of the project, and uses the code metrics to analyze the performance of the framework. The fourth section summarizes the research work in the paper.

2. MOCV Framework

This section describes the overall idea of the MOCV framework, and describes the implementation of the MOCV framework details by analyzing the construction process.

2.1. Framework Structure

The MOCV framework is based on the MVC design pattern and incorporates the layered idea of the three-tier architecture. I It solves the problem that the business logic and data logic layering in the MVC design pattern is not clear. At the same time, the concept of Controller and Model is used to deal with the problem of high coupling between the presentation layer and the business logic layer in the three-tier architecture.

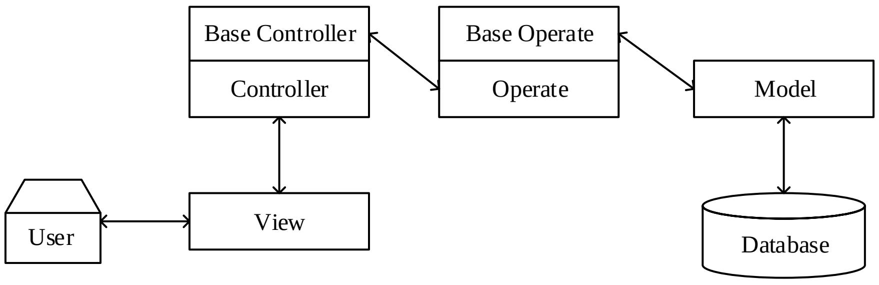

Fig. 1. MOCV framework.

Fig. 1 shows the overall architecture of the MOCV framework. The MOCV framework consists of three layers and four component modules. The four modules refer to Model, Operate, Controller, and View. Among them, the View module and the Controller module together constitutes the presentation layer, the Operate module constitutes the business logic layer, and the Model module constitutes the data access layer.

In the presentation layer, the View interacts directly with the user. It is a user interface layer component that is responsible for displaying the data required by the user and providing an input interface for user's operations. The Controller receives the information collected in the View, and passes it to the corresponding Operate module for logical processing. When it is receiving the processing result, it will return the data processed to the View for displaying. In this layer, we defined a concept of "Base Controller". It is a part of the Controller module but has no view that matches it. As the parent class of normal controllers that match a view and can implement a specific function (We call these controllers 'Specific Controller'), it encapsulates the Specific Controller's common actions or methods which include common function code. Some of them are marked by abstract or virtual so that can be overridden in subclass. Moreover, the Base Controller also acts as a bridge between the Specific Controller and the Operate module by some internal code.

In the business logic layer, we proposed the new module concept of Operate, which is used to solve some disadvantages in the MVC design pattern. In traditional MVC design pattern, the Controller layer not only performs data interaction with the View layer, but also processes some business functions. Similarly, the Model layer also needs to handle some business logic and data logic when it processes the data access. These make the code structure of software which is developed by the MVC design pattern orderless. In the MOCV framework, the Operate module is a place to handle business logic and data logic. When it receives the call of the Controller module, it will process the content with the logical method that has been designed and then interact with the Model module. After a series of processing, the result is returned to the Controller module for presentation to the user. Like the Controller module, the Operate module has the concept of 'Base Operate' and 'Specific Operate.' As the parent class in the Operate module, the Base Operate encapsulates some common logic code, mainly based on data logic, in order to be extended by the Specific Operate. And the Specific Operate provides specific business logic processing for the corresponding Specific Controller.

In the data access layer, the Model module is responsible for connecting to the database and doing some simple operations such as reading and writing.

2.2. Framework Detail Implementation

In this section, we will discuss the three processes of building MOCV framework.

2.2.1. Phase 1: Add the idea of the operate

In the traditional development of MVC design pattern, because both the Controller and the Model involve the writing of business logic and data logic code, this situation makes the code in development become confusing, which makes it difficult to accurately divide the work in the layered development of the project. At the same time, it increases the difficulty of post-maintenance, and it also reduces the portability of the development framework. In order to solve these defects, wo try to add one area to store business and data logic code.

However, at the beginning, we did not form a separate module in the framework, but encapsulated the business logic code used in the Controller and placed them in the Model. So there are two types of code in the Model module, one is the logic code representing the Operate module, and the other is the data access layer code of the Model module itself.

In the Controller module, we analyze the code in the Specific Controller and encapsulate the class file of the Base Controller, which can provide a variety of code that is commonly required in the Specific Controller. And the code in the Base Controller can be overridden by the Specific Controller after being extended. There is one function named "GetSpecificOperate," which need to return an object of the Specific Operate when it be overridden by the Specific Controller, in order that the Specific Controller can user the logic code in the Operate module. The relation is shown in Fig. 2.

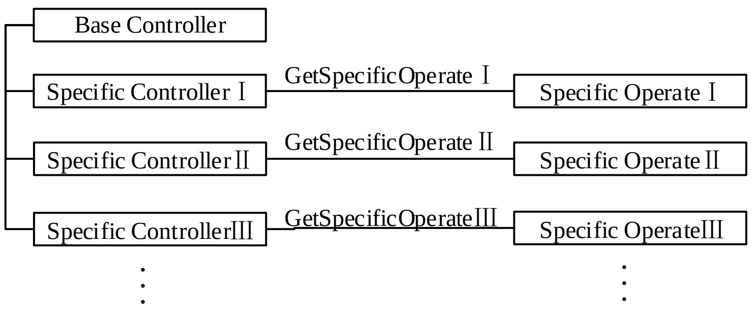

Fig. 2. The relation of controller and operate.

2.2.2. Phase 2: Introduce ORM and factory model

At this phase, we introduced ORM technology in the data access layer. ORM means Object Relational Mapping which is a persistent technology and used to implement conversion between data of different types of systems in an object-oriented language[9]. From the perspective of efficacy, it actually creates a "virtual object database" that can be used in programming languages. Through this mapping, developers can operate relational databases like operations objects without having to deal with complex SQL statements, more focused on business logic development.

The Operate module is completely separated from the Model module to form a separate module. Similar to the Base Controller, we analyze the code in the Specific Operate and encapsulate the class file of the Base Operate. Base Operate encapsulates some methods that are handled with Lambda expressions to implement complex data logic. The Specific Operate can extend from the Base Operate and selectively override some code so that it can implement the business logic and data logic required in the Specific Controller.

In the Controller module, we have added several member variables to store strings with special meanings which can help us identify different Controller-View-Operates. Through the tagging of these variables, we can use the factory pattern to improve the matching method between Controller and Operate. The relation is shown in Fig. 3.

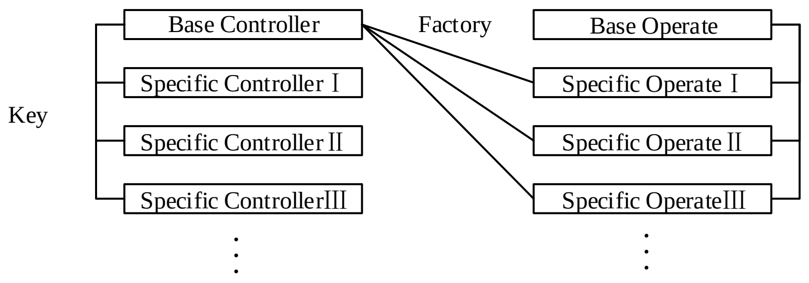

Fig. 3. The new relation of controller and operate.

2.2.3. Phase 3: Improve the operate module

At this phase, we have further optimized the packaging of the Base Controller, including some operations about operate View. And in the Base Operate, wo also added several member variables to store strings with special meanings which can determine the database table for the operation we need.

Because Specific Operate needs to override a large amount of data logic code, Base Operate introduces generic technology to reduce these work. Generics can be understood as "type parameterization." In the encoding process, the implementation of the function logic is independent of the type, and the code written is required to be applicable to all types[10]. In Base Operate, using generic technology to wrap the data logic code, Special Operate saves a lot of repeated data logic writing, and does not reduce system efficiency, so that the maintainability of the code and development efficiency can be improved.

3. Performance Analysis

In this section, a case study of the MOCV framework is carried out with a project example of "competition registration and online review system." The project code design is carried out in three phase of the MOCV idea construction process. The software development platform here is Visual Studio 2015, the database is SQL Server 2012, and it is developed in C# language.

3.1. Design of Overall Structure

Based on the MOCV framework idea, combined with the specific business functions of the "competition registration and online review system," the structural diagram of Fig. 4 is designed.

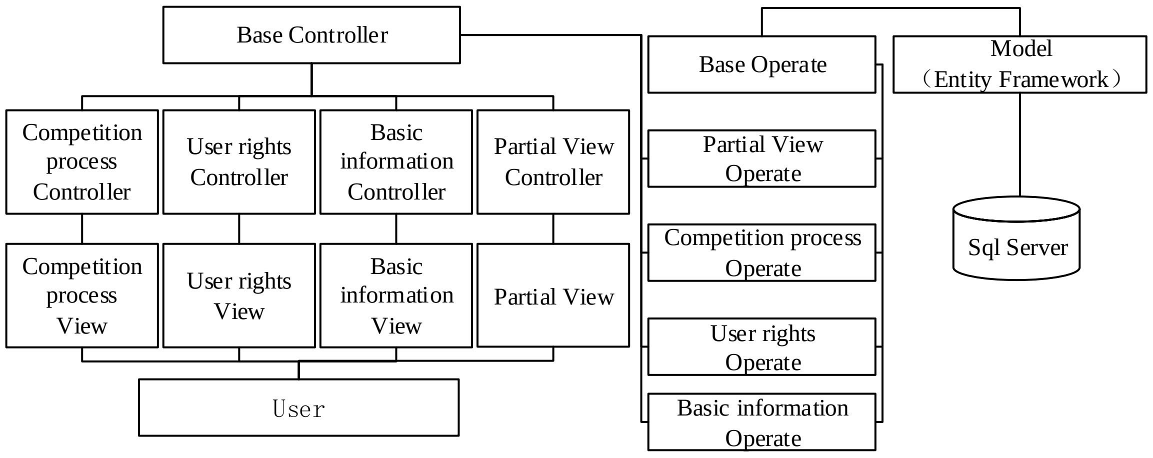

Fig. 4. The overall structure.

3.2. Analysis of Code Quality

We have used the way of framework realization in three phases of construction progress of MOCV to code the users' authority as well as the universal control widget modules respectively, and used the code metrics analysis provided by Visual Studio 2015 to analyze the performance of the framework codes of the three realization versions. We note the three ways of realization here as V1, V2, and V3 respectively. The results are shown in Table 1.

Table 1. Analysis of Code Metrics

	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Code	Maintainability Index
V1	2776	5	354	7965	76
V2	2391	5	441	6553	82
V3	2095	5	385	4917	88

The analysis of code metrics results that Visual Studio calculates include cyclomatic complexity, depth of inheritance, class coupling, lines of code, and maintainability index.

Cyclomatic complexity measures the structural complexity of the code. It can be obtained by calculating the number of different code paths in the flow of the program. The lower the cyclomatic complexity is, a higher quality the code can have. Those programs with more complex control flows need to be tested with more unit tests to achieve a high coverage of codes, and their maintainability is worse.

Depth of inheritance indicates the number of class definitions that extend to the root of the class hierarchy. It can be figured out by getting the maximum of depth of inheritance of all kinds of inheritance in the project. The deeper the hierarchy the more difficult it might be to understand where particular methods and fields are defined or redefined. This may lead to an increase of difficulty of maintainability.

Class coupling measures the coupling to unique classes through parameters, local variables, return types, method calls, generic or template instantiations, base classes, interface implementations, fields defined on external types, and attribute decoration. The measurement does not include basic and built-in types such as int32, string and object. Good software design dictates that types and methods should have high cohesion and low coupling. High coupling indicates a design that is difficult to reuse and maintain because of its many interdependencies on other types, which makes software design difficult to reuse and maintain.

Lines of code Indicates the approximate number of lines in the code. The count is based on the IL(intermediate Language) code and is therefore not the accurate number of lines in the source code file. The count nor includes space, comments, brackets as well as the declarations of members types and namespaces. A very high count might indicate that a type or method is trying to do too much work and should be split up. It might also indicate that the type or method might be hard to maintain.

Maintainability index calculates an index value between 0 and 100 that represents the relative ease of maintaining the code and is based on cyclomatic complexity, lines of code and Halstead Volume method. A high value means better maintainability.

Analysis of table 1:

A: Cyclomatic complexity: the value decreased stage by stage. This is because of a clearer hierarchy with the improvement of versions, which made the code more structural and reduced the redundancy of codes.

B: Depth of inheritance: these three versions have the same depth of inheritance due to the unique model of inheritance of MOCV.

C: Class coupling: we found that the value increased from V1 to V2, while decreased from V2 to V3. The reason of the increase is that encapsulation of the Base Operate was added, due to which the class coupling increases in the progress of specific operate inheritance. The reason of the decrease is that through the use of genericity, class coupling decreases to some extent. Although class coupling of V3 was somewhat higher than that of V1, V3 got a lower cyclomatic complexity as well as fewer lines of code in return.

D: Lines of code: the amount of code during development decreased greatly because of base controller, base operate and ORM technology.

E: Maintainability index: value increases stage by stage. This indicates that with the improvement of details in frameworks, the maintainability of framework is greatly improved in general, which contributes much to hierarchical development and maintenance.

It is to conclude that the general performance of V2 and V3 is evidently higher than V1. To further compare the performance of V2 and V3, we took V2 and V3 to implement whole project functions respectively and analyze the outcome of code metrics. The results are shown in table 2:

Table 2. Analysis of Code Metrics

	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Code	Maintainability Index
V2	4693	5	703	15648	73
V3	4081	5	628	12741	80

Analysis of Table 2:

A: The amount of codes and their complexity grows with the increase of business requirements of projects, and the cyclomatic complexity as well as class coupling of V2 and V3 also grows to some extent. Compared with V2, V3 shows a better performance in terms of the cyclomatic complexity and class coupling.

B: V3 has a stronger advantage in terms of source lines of code with the increase of business requirements. A large amount of codes was reduced, which means that the productivity of developing a software increases significantly.

C: Both two values of index of maintainability in V2 and V3 decreased because of the complication of business requirements. Nevertheless, the maintainability of V3 remains fine, which highlights that MOCV framework is capable of remaining a good maintainability and increasing the productivity of developing simultaneously.

4. Conclusion

In view of the shortcomings of the three-tier architecture and the MVC design pattern in software development, this paper proposes the MOCV framework, including discussing the division of layers and modules in the MOCV framework and analyzing the specific implementation of the framework. It shows the clear structure of the MOCV framework by using this framework to design for the actual project, and through the analysis of the specific code metrics, the numerical results show that the software system development with this framework has more advantages in development efficiency, portability of the framework and maintainability of the software.

Acknowledgment

This paper is supported by the National Natural Science Foundation of China (no.61501161) and the National Key Research Development Program of China (no.2016YFC0801800).

References

[1] Hendricksen, D. (2012). *12 Essential skills for Software Architecture*. Upper Saddle River: Person Education Group.

[2] Mitra, S. (2014). Using UML modeling to facilitate three-tier architecture projects in software engineering courses. *ACM Transactions on Computing Education*, *14(3)*, 1 - 31.

[3] Liu, Y. X., Yao, K. X., & Xu, D. Y. (2012). Automatic code generation based on template on net framework of three layers architecture. *Computer Technology and Development, 22(8),* 13 - 16.

[4] Hao, W., Ai, L. M., & Wang, Y. H. (2009). Hot spot implementation method of three-layer software framework. *Journal of Computer Applications, 29(9),* 2541 - 2543.

[5] Wang, L., Wang, Z. G. (2017). Research and application of improved three-tier architecture. *Computer Engineering and Design, 38(7),* 1808 – 1812.

[6] Xu, Z. H., Fan, Y. T. (2007). Research and implementation of MVC design pattern in development of net with three tier structure. *Journal of Beijing Electronic Science and Technology Institute, 15(2),* 70 - 73.

[7] Tong, Y. (2016). The exploration and research of software development architecture based on ASP.NET MVC pattern. *Journal of China Academy of Electronics and Information Technology, 11(6),* 599 – 602.

[8] Yuan, L. G., Chen, Z. Y., Li, F. P., & Guo, T. (2013). A kind of design and majorization of software development framework. *Computer Systems & Applications, 5,* 70-74.

[9] Gregory, V. (2018). Lessons in persisting object data using object-relational mapping. *IEEE Software, 3,* 1-1.

[10] Zuo, Z. K., & Xue, J. Y. (2015). Research on generic constraints of Apla. *Journal of Software, 6,* 1340-1355.

Guoxiang Zhou was born in 1956, received his B.S. degree in 1981 and M.S. degree in 1996, all from Hefei University of Technology, Hefei, Anhui, China. He is currently a full professor in School of Computer and Information, Hefei University of Technology. His main research area lies in software designing and engineering.

Ruirui Cao was born in 1994, received his B.S. degree in 2016 from Nanjing University of Posts and Telecommunications, Nanjing, Jiangsu, China. He is currently studying for M.S. degree in School of Computer and Information, Hefei University of Technology, Hefei, Anhui, China. His main research area lies in software designing.

Lei Shi was born in 1980, received his B.S. degree in 2002, M.S. degree in 2005, and Ph.D. degree in 2012, all from Hefei University of Technology, Hefei, Anhui, China. He is currently an associate professor in School of Computer and Information, Hefei University of Technology. His main research area lies in software designing and wireless network optimization.