International Journal of Grid and Distributed Computing
Vol. 9, No. 6 (2016), pp.181-196
http://dx.doi.org/10.14257/ijgdc.2016.9.6.18

Patterns for Development of Windows Form Applications and Web Applications

Rahim Lotfi

Department of Software Engineering, University of Isfahan, Isfahan, Iran
mr_lotfi@eng.ui.ac.ir

Abstract

Design pattern is a mechanism to show experience in object-oriented design, as well as an appropriate solution which has been provided by experts for particular problems and which can be used over and over throughout the design. Using design patterns helps improve software quality and reusability. There are different patterns for development of data source (database, file, array, etc.) systems but most of them have features that are not appropriate for code generation. We seek in this research new patterns for development of data source systems that can accelerate the development of such systems and reduce costs and are appropriate for code generation as well. This article deals with challenges related to prototype, singleton and MV (MVC, MVP, MVVM) patterns. As a solution, we will propose two patterns called MVC+ and MVC++ as well as a tool called LCG for code generation. We compare the proposed patterns with similar patterns in terms of efficiency. MVC+ and MVC++ are appropriate for model-driven architecture, code generation and the development of windows form applications and web applications.*

***Keywords**: Design Pattern, Code Generation, Model-Driven Architecture, MV*, MVC+, MVC++*

1. Introduction

Design pattern is a mechanism to show experience in object-oriented design and an appropriate solution which has been provided by experts for particular problems and which can be used over and over throughout the design [1]. Patterns enable developers to reuse a given solution that has been set for the problem by experts. Adopting a design pattern in windows form application and web application can support and provide reusability and better compatibility [2].

This article seeks to provide two new patterns toward the development of data source (database, file, array, etc.) systems that can support acceptable implementation and code generation. There are various patterns to work with data source systems. However, most of these patterns are inappropriate for code generation and can be used only for particular systems and technologies. For example, MVC can be used only for web applications; it cannot be used for windows form applications [3]. The patterns proposed in this work, called MVC+ and MVC++ are appropriate for model-driven architecture (in 2001, the Object Management Group (OMG) published the first version of its model-driven architecture (MDA) specification[4].), code generation and the development of windows form applications and web applications. In these patterns, the data- generating modules are completely separate from the data-using modules and easily support code generation capability.

The idea of using layering is appropriate for code generation, because layers can be generated separately. Using tools for code generation saves time and minimizes the cost. Moreover, in order to improve efficiency and reduce the costs, using singleton or prototype patterns helps avoid creating objects for layers that include only algorithms and methods for running.

The remaining sections of this article are organized as follow. Section 2 deals with the details of the challenges and goals of the research. Section 3 examines the patterns related to this research, i.e... Singleton and Prototype patterns, and the way they can be used. Moreover, MV* family of patterns will be examined in this section. Section 4 will introduce the proposed patterns, i.e., MVC+ and MVC++, will compare them with MVC and three-tier patterns that are used more frequently nowadays, and will deal with the implementation of a simple example for better understanding of the suggested patterns. In section 5 we have one case study as well as some useful suggestions for the implementation and use of MVC+ and MVC++ patterns. Also in this section, we will introduce a code generation tool, called LCG, and the implementation of a practical example with LCG tool. The article will finally end in section 6 with an overall conclusion.

2. Challenges and Goals

As mentioned above, this research aims to provide a new design pattern for implementation of data source (database, file, array, etc.) systems, so that the development of such software systems could be accelerated and the costs would be reduced. We also seek in this study a new design pattern that would be appropriate for Modeling and code generation and can be used for different platforms (web, windows, Mobile). Now, we review the challenges that we faced in this research:

Limited Resources: One of the key goals in software engineering is to make optimal use of the resources (memory, processor, bandwidth), and we faced too limited resources in most circumstances. Mobile devices have but limited power, bandwidth, processor speed and memory [5].

Ease of Implementation: Simplicity is the most important capability. If a design pattern is an optimal one in terms of time, memory and communication, but complicated in terms of implementation, it will certainly be considered inadequate and inefficient. Ease of implementation is very crucial issue. The developer should require no special experience.

Modeling and code generation: Any developer’s main goal is saving the time and minimizing the costs. The framework should have the capability of modeling in order for the developer to model it easily. If different parts of an architecture are able to be implemented separately, the developer can generate the modules of each part by use of tools.

Reusability: The written or generated codes should be able to be reused. This requires each part’s independence.

Flexibility and Scalability: An architecture should support extendibility and make it possible for the developer to extend his or her product incrementally. Moreover, an architecture should have the feature to be personalized so that the developer can change it or combine it with other patterns in order to suit his or her own needs.

Tech Support: One of the goals of this research is to provide an architecture pattern that can be used for different technologies. An architecture should not be dependent on a particular technology, because it should change as soon as the technology changes.

3. Related Patterns

Due to the fact that our work is in the patterns domain, in this section, we review some patterns that are more related to our proposed patterns, i.e., MVC+ and MVC++.

3.1. Singleton Pattern

Sometimes it's important to have only one instance for a class. For example, in a system there should be only one window manager (or only a file system or only a print spooler) [6]. Singleton pattern is a creational patterns that assures us that only one instance of a class will be created. Singleton pattern is used for the internal and external centralized management of resources and creates a common access point for instance creation. Singleton pattern is one of the simplest design patterns that includes but one class and assures that only one instance of a class would be created per all requests during the execution of a program. The purpose of singleton design pattern is to ensure that only one instance of a class will be created per all requests. The performance of this pattern is such that an instance of a class is created within the class itself and when other classes want to have access to that class, they can have access only to this instance and cannot create instances of the class. Figure 1 shows the class diagram of this pattern.

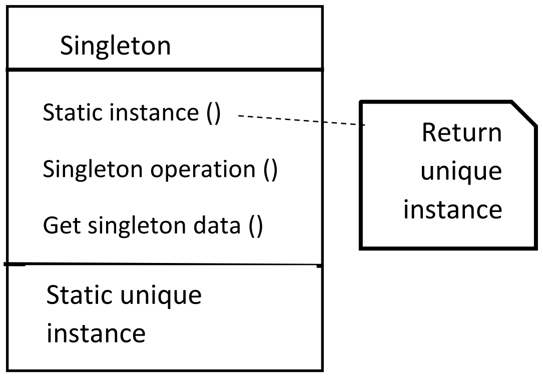


Figure 1. The Class Diagram of Singleton Pattern [7]

As shown in this figure, the constructor of this class is of private (or can use “protected”) type, so one cannot directly create a new instance of this class through a new operator. This class has an attribute called “instance”, which is of the same type as the class itself. This attribute holds the created object within itself. The only way of creating an instance of this class is instance method calling, which returns the created instance. This will allow the requests to be controlled and will assure that only one instance of this class will be created.

3.2. Prototype Pattern

[6]Today’s programming is all about costs. Saving the costs is an important issue in software engineering and creating objects from classes is a costly process. When we talk about creating objects from classes, there is a very interesting design pattern with a simple appropriate idea. The main idea of this pattern is to simulate rather than create objects. When it takes long to create objects out of classes or when the resources are too limited, we can copy them and make use of their functions. In this case, only the first object is created and will be copied in subsequent times. We use methods of creating instances without knowing about the details of the classes and being engaged with the details of instance creation. There are two methods for copying an object:

Shallow: In this type of copying, all attributes are copied, even those that are used for reference to an object. As a result, a change in the target object will lead to a change in the source object. In this copying type, the target object copies all values of the source object.
Deep: In this type of copying, the target object and the source object are created separately and independently. Figure 2 shows the class diagram of prototype pattern.

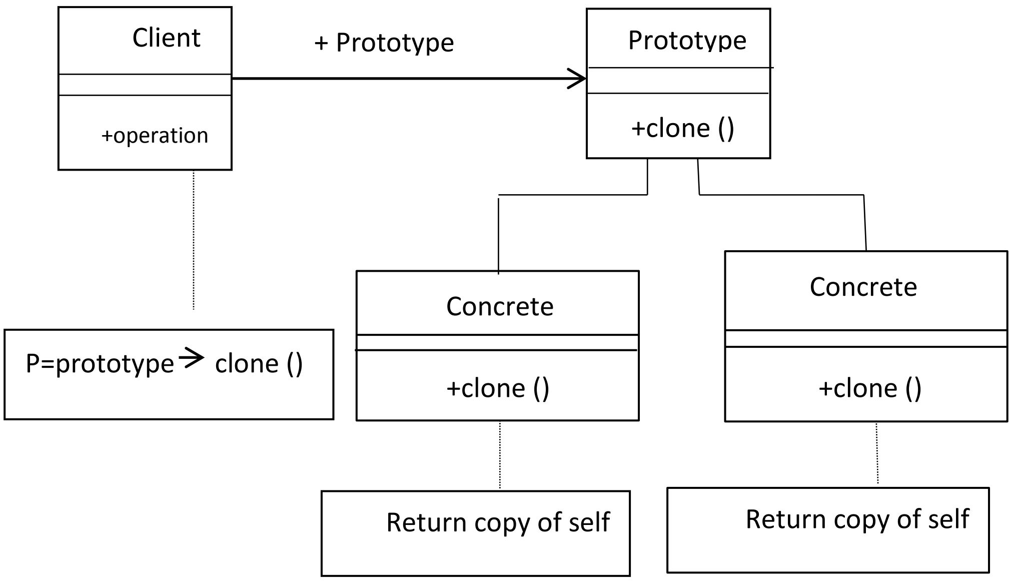


Figure 2. The Cass Diagram of Prototype Pattern [9]

3.3. MVC Pattern

MVC design pattern was first presented by Trygve Reenskaug in 1970 and was documented by Krasner and Pope in 1988 [8]. Any software that interacts with its users requires a user interface. MVC pattern is one of the most common design patterns for such programs [3]. MVC is a multi-layer pattern for isolating different parts of a program. As shown in figure 3, this pattern consists of three parts, namely model, view and controller.

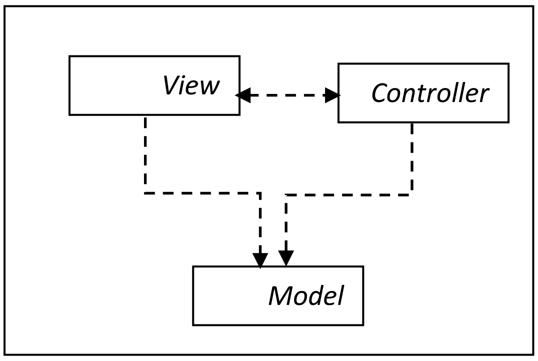


Figure 3. Classic MVC [9]

Model is a non-visual object to interact with the data stored in the data source [9]. Data source includes not only the data stored in the databases of SQL, Oracle and so on, but also the file, array of numbers and any other thing. View includes only the controllers and objects that display information. The controller is the interface between the model and the view. The controller takes the user's requests, extracts the required data from the model based on the requests, performs the preparations required and finally returns the results to the user properly. The above explanation is shown in figure 4:

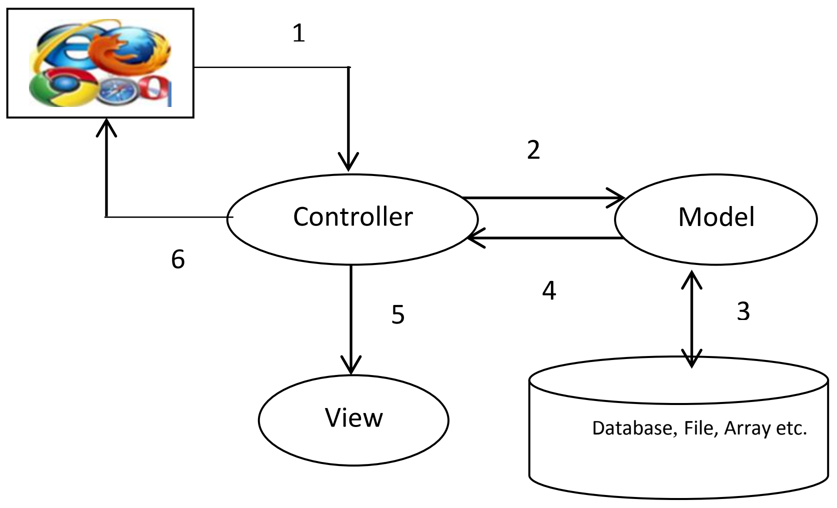


Figure 4. The Procedure of MVC Design Pattern

Nowadays, MVC is known as an appropriate design pattern for the development of web applications. Although MVC is doubtlessly a valuable and useful pattern for such systems, a series of features mentioned below show its lack of appropriateness for code generation. Besides, MVC pattern cannot be implemented for applications [3]. Here are a few of these disadvantages. However, despite these disadvantages, we cannot say that MVC pattern is not a good one, because any pattern has its own conditions and uses; that is, it can be the best and the most useful choice in certain cases and under certain conditions.

1) Each controller usually controls one view, but if it controls several views simultaneously, the developer should manage it himself or herself. MVC pattern has a poor performance in controlling the states of views [3]. This shortcoming has been removed in MVVM pattern.

2) The user's request is dispatched to the controller [9]. This is not secure against DOS attack and the requests are not authenticated. Even, Microsoft resolved this problem in asp.net MVC in iis, which causes additional overhead.

3) A change in the model will lead to a change in the controller and view and the cost of frequent updates will slow down the speed.

4) MVC pattern increases complexity from the designer's perspective, because he or she has to adapt the problem with the nature of MVC pattern. We should not resolve the problem with a particular solution, but the solution should be proportionate with the problem. Pattern is a common vision toward a problem, but should not prevent us from thinking about it.

MVC has a frozen structure and cannot be personalized or customized. In other words, we place MVC pattern on a problem like a frame and resolve the problem with our own solution. As a result, this pattern cannot be used when view, controller and model cannot be separated, such as widgets [3].

5) The designer should be experienced enough, because MVC pattern is hard to implement and is inappropriate for small and medium-sized projects [2].

After the discovery and documentation of MVC pattern, other patterns such as MVP and MVVM were presented. These patterns had been derived from MVC pattern (and improved some shortcomings and disadvantages of MVC pattern and added a new capability to it. MVP and MVVM patterns will be described in sections 3.4 and 3.5).

3.4. MVP Design Pattern

MVP design pattern was presented by Mike Potel in 1996 [12]. As shown in figure 5, this pattern includes three parts, namely view, presenter and model, and has been derived from MVP design. The function of model in MVP pattern is to manage the data and report it to view for changing the data [3]. The view part displays the data and authenticates the users. However, there is a problem here, and that is, if we have several views, they may have the opposite effect for similar data and performance and efficiency may come down. The presenter is the core of this pattern and is in direct relation with the model and view and indeed coordinates the interactions [3]. Figure 5 shows how this design pattern works.

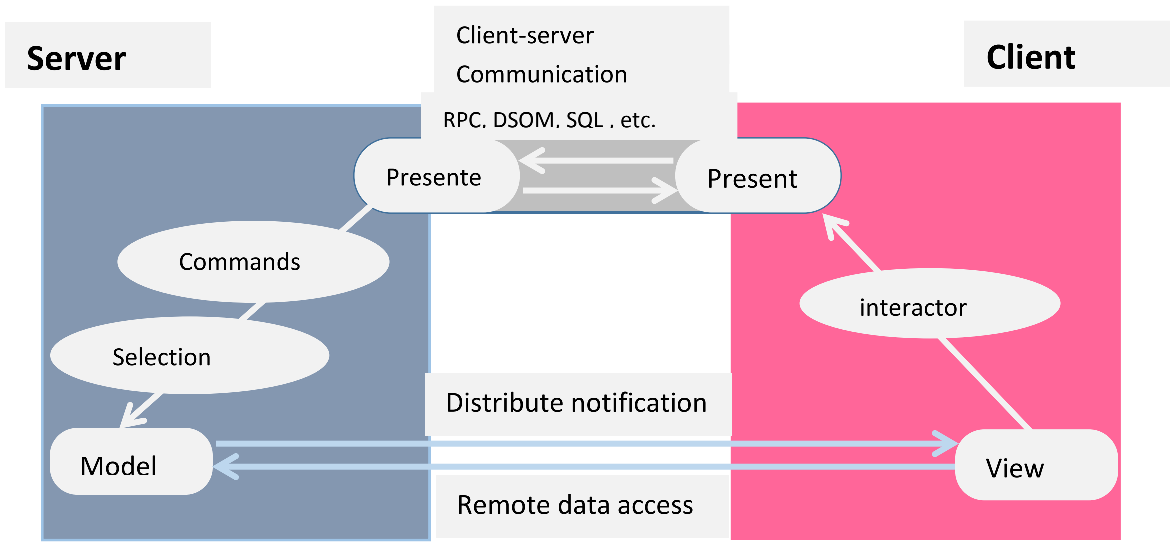


Figure 5. The Procedure of MVP Architecture [12].

3.5. MVVM (Model-View-View Model) Design Pattern

One of the shortcomings of MVC design pattern was the management of the state of views, which resulted in presenting MVVM pattern [3]. MVVM pattern is one of the most popular patterns of user interface, which was presented by John Gossman in 2005 for WPF, Silverlight [14]. MVVM pattern is similar to PM (presentation model) pattern, which was presented by Martin Fowler in 2004 [13]. The two patterns focus on behavior and state in view. MVVM pattern is a specific pattern for WPF platform, Silverlight, which has been derived from MVC pattern. MVVM pattern has been presented to remove the shortcomings of MVC design pattern for view state management [3].

As shown in figure 6, MVVM pattern consists of three parts, namely view, view model and model. In MVVM pattern, view is in direct relation with model. The model part has the function of managing the data. The view has controls for displaying the data and the view model manages the user's inputs. Figure 6 shows how this pattern works.

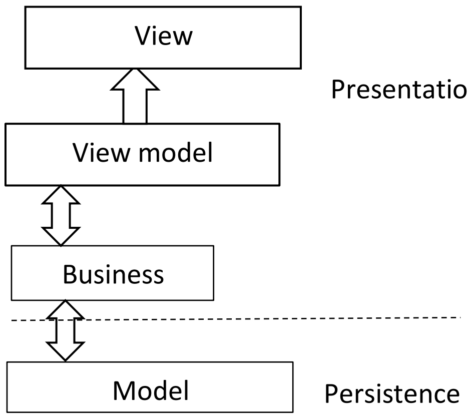


Figure 6. The Procedure of MVVM Pattern [14].

3.6. Three-tier Architecture

Nowadays, most windows form applications and web applications are written based on this three-layer architecture. This pattern consists of three tiers, namely data access (or DB broker), user interface (or presentation) and business logic. Presentation includes only the controllers and objects that display information. Data access is the only part that has access to the data source (database, file, array, etc.) and business logic is the interface between the Data access and user interface. Figure 7 shows a specimen of a three-tier application.

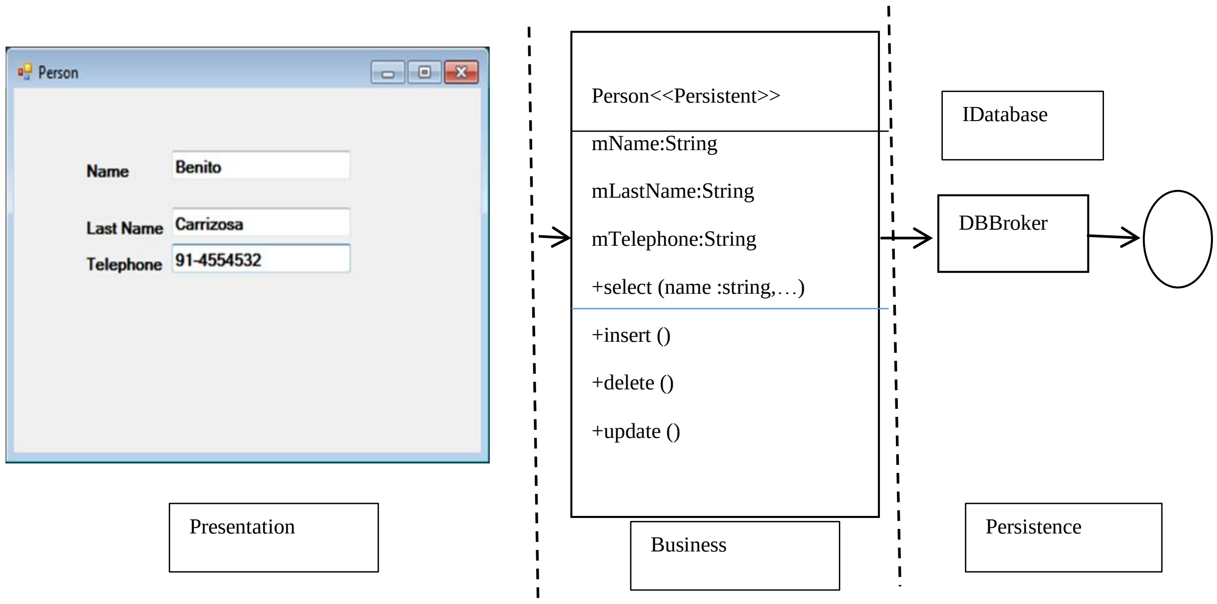


Figure 7. A Specimen of a Three-Tier Application [15]

4. Proposed Patterns

Two new patterns called MVC+ and MVC++ are introduced in this section. These patterns can be used for windows form application and web applications. Furthermore, they can be used for model-driven architecture and code generation. At the end of this section, we will compare the proposed patterns with similar patterns in terms of efficiency.

4.1. MVC+ Pattern

Separation and division of tasks and functions among different layers has many proponents, which is a true belief and is very important and crucial. Among the advantages of this method is usability of layers with the introduction of new technologies. The new design pattern of MVC+ is introduced in this section. This pattern is derived from MVC pattern and it is its combination with other patterns that makes it usable for windows form application and web application.

This pattern consists of four parts, namely model, view, controller and data object. View includes pages or forms used to display the data. There is no relation between view and model and view does not know where the data have come from and how they are prepared.

Model is the only part that has access to the data source (database, file, array, etc.) and controller is the interface between the model layers and view. We have considered a data unit called data object to exchange the data among the layers. In this case, the model layers, view and controller deal with and recognize only the data object. As a result, they are light and it is faster to create object from them, which improves its performance or efficiency. Efficiency is a key issue in software development, which should be taken into account in system design in order to help minimize the use of resources. We have separated the data- generating modules from the data-using ones. Attempts have been made to have data independent of one another in order for the development team to be able to work in parallel. We have tried to separate the challenges of each part in order to make optimal use of programmers' expertise. Any manager is concerned about whether the architecture allows the team to work independently and interact with one another in a controlled and systematic way [11]. The architecture and decisions should not choose the linear trend of software output in order for the development team to be able to work in parallel. Figure 8 shows the primary template of MVC+ pattern. It is very useful to separate the layers and make use of data object for code generation. For code generation, we require an architecture that can generate the layers independently. MVC+ pattern makes it possible for each layer to be generates separately.

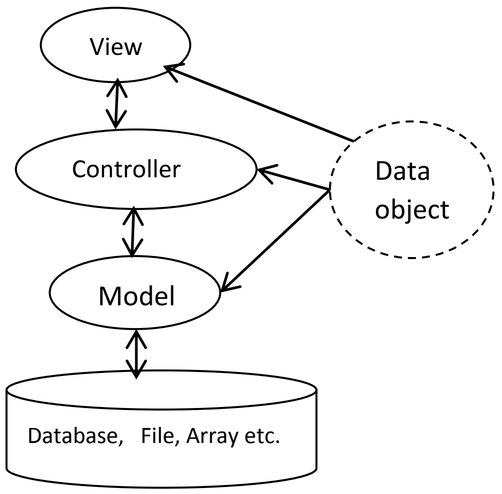


Figure 8. Primary Template of MVC+ Pattern

Since we have no data field inside the layers of the model and controller, and they just include algorithms and methods for running, the singleton pattern can be easily used so as to make the layers of model and controller singleton so that only one object can be made at runtime. This saves memory and time (which will be dealt with thoroughly in section

4.5). The most important advantage of the use of MVC+ pattern is the separate designing of the layers of model and controller. These layers are executed as a single object at the runtime phase. In other words, only one object of the controller and model is created at runtime, and they are designed separately and run in an integrated way. At runtime, two objects of controller and model stick together and are run together as they are singleton. Due to the separation of the data from the layers of model and controller, these layers only include commands and methods for running and there is no data field inside the layers. As a result, they have no interference during concurrent and parallel running and there is no need to synchronize the commands, because synchronization lowers performance sharply. Actually, encapsulation which is an important issue in object-oriented programming has been observed in this pattern. Moreover, memory consumption and speed have been improved. Figure 9 displays two views of the runtime and design time of MVC+ pattern. However, prototype pattern can be used instead of singleton pattern in order to prevent from the repetition of the layers during runtime, but the latter pattern seems easier to use.

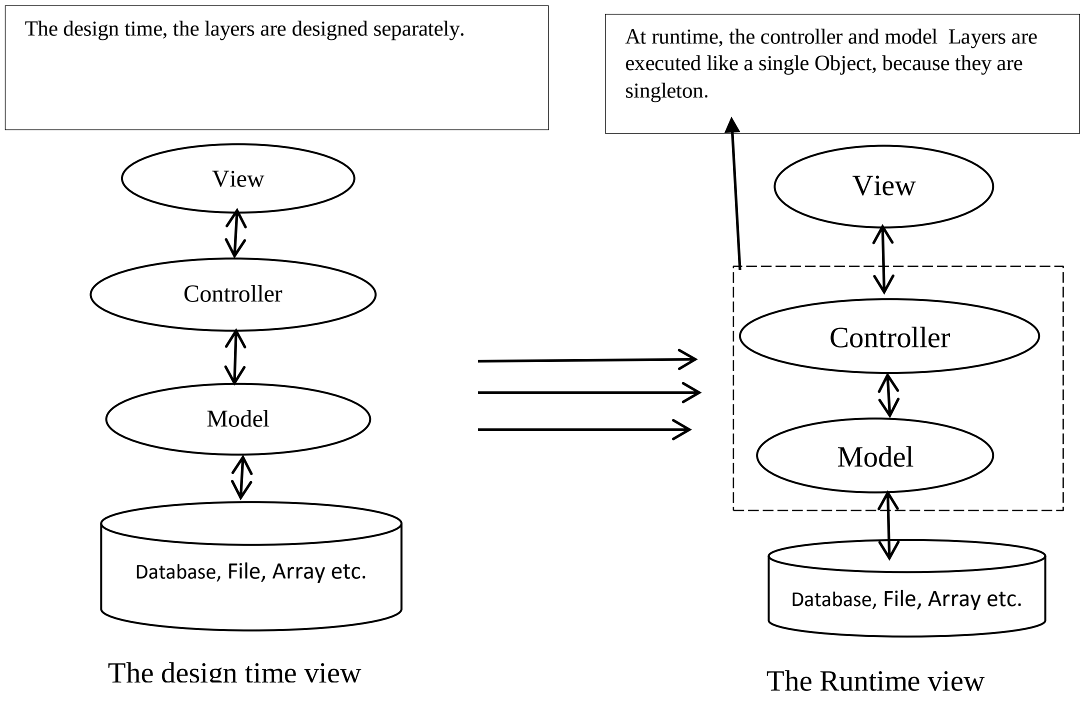

The design time, the layers are designed separately.
At runtime, the controller and model Layers are executed like a single Object, because they are singleton.
View
Controller
Model
Database, File, Array etc.
The design time view
View
Controller
Model
Database, File, Array etc.
The Runtime view

Figure 9. The View of MVC+ Pattern During Design Time and Runtime

The use of singleton pattern creates a common point in the layers to create object from class, which prevents disorder. Creation of a common relation for creating objects in order to prevent from disorder is the main idea of factory pattern, which has been used in singleton pattern. Design simplicity is another feature of MVC+ pattern, because it needs no special experience and the developer can implement it easily. MVC+ pattern can be personalized easily and therefore the developer can adjust the layers in proportion with his or her needs and model the functions of the layers for code generation. It is better in model-driven development method to have specified structures and architecture in order to be able to generate each part separately.

Furthermore, if we are concerned about testability of the system, we should pay attention to the testability of each one of the elements. Separating the different parts from one another helps one test each part separately.

MVC+ pattern can be used for projects with different sizes. MVC+ pattern is appropriate for agile methodology, because the parts can be constructed separately and the software can be developed incrementally. The incremental method is adequate for agile methodologies [11].

4.2. MVC++ p\Pattern

In MVC++ pattern, the functions are divided among the layers, but the model layer has two functions. First, it receives the data from the controller and takes them out of the data object and stores them in the data source, then it pulls the data out of the data source and places them in the data object and delivers to the controller. This leads to centralization of further part of the process in the model. The model layer has immense functions and the most and heaviest operations are centralized in this part. In large projects where there are abundant requests, this causes the model to act as a bottleneck. It seems that the division of tasks has not been done accurately. An architecture should be such that it makes the constructors of elements (view, controller and model) consider balance. Therefore, this is better to be done in two stages. We divide the model layer into two layers for this purpose and add another class called parent model to MVC+ pattern. The model extracts the data from the data object and delivers them to the parent model or delivers the data from the parent model and places them in the data object and dispatches them to the controller. The parent model works with the data storage frame at the data source level.

After the separation of the model layer into model layer and parent model, one can singleton these layers and the controller layer so that these three objects work as a single object like MVC+ pattern during the runtime, and memory and time can be saved and concerns can be removed completely during the design-time. This leads to independence of the layers as well as their further usability. More importantly, layers can be generated with code-generating tools more easily, because the functions have been divided in a better way and the layers are quite independent.

As the layers just recognize the data object, the layers can be generated with regard to the data object. For example, we can generate a user control for the code generation of the view layer with regard to the data object for each activity (insertion, deletion, editing).
MVC++ design pattern supports much personalization capability. Whereas MVC++ pattern can be used for projects with different sizes like MVC+ pattern, it is recommended that MVC++ pattern be used for large projects. Figure 10 shows MVC++ design pattern.

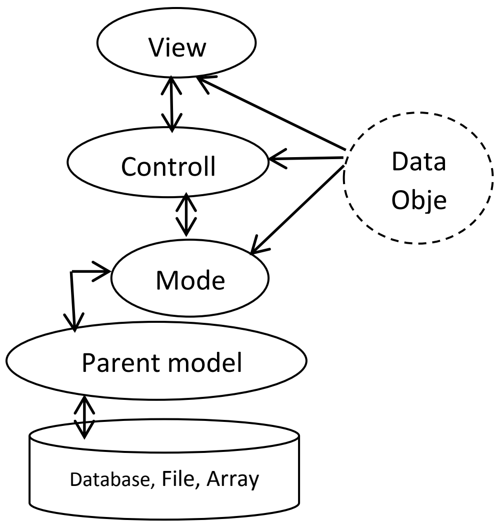


Figure 10. MVC++ Design Pattern

4.3. Implementation of an Application with MVC+ Pattern using Singleton Pattern

In order to show how to implement MVC+ pattern, we give a simple and small example in this part due to the limited number of pages. In this example, we have a "Person" entity with the fields of code, name and last name. Person entity by default has the methods of adding, editing, deleting, searching and listing. The class diagram of person entity using singleton pattern has been shown in the following:

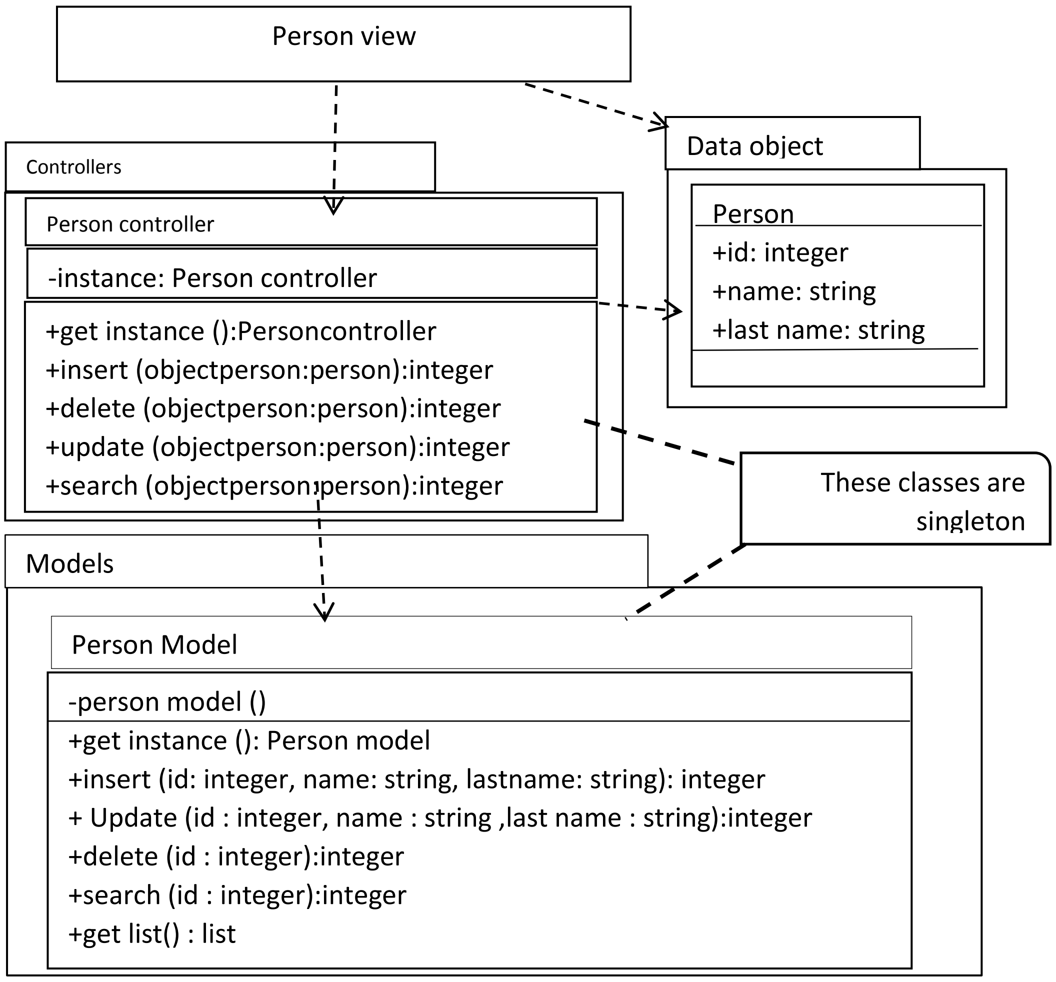

Person view
Data object
Controllers
Person controller
-instance: Person controller
+get instance ():Personcontroller
+insert (objectperson:person):integer
+delete (objectperson:person):integer
+update (objectperson:person):integer
+search (objectperson:person):integer
Person
+id: integer
+name: string
+last name: string
These classes are singleton
Models
Person Model
-person model ()
+get instance (): Person model
+insert (id: integer, name: string, lastname: string): integer
+ Update (id : integer, name : string ,last name : string):integer
+delete (id : integer):integer
+search (id : integer):integer
+get list() : list

Figure 11. The Class Diagram of Person Entity in MVC+ Pattern

The data are exchanged among the layers in the form of objects from person class. The model and controller are in the form of singleton. The model works with the storage and calling of procedures at the data level (it means SQL server here). When we need listing, it can be done without the need to create an instance of the person class.

4.4. Implementation of MVC+ using Prototype Pattern

When creating objects, one can use prototype pattern to improve the performance and speed of the system. The main idea behind prototype pattern is copying rather than creating objects without knowing about the details of objects and classes.

We show in this section how to implement MVC+ pattern using the prototype pattern. We used the singleton pattern in section 3.4 in order to prevent from repetition and improve the performance of the model and controller layers. We show in this part how to use prototype pattern instead of singleton pattern. When using prototype pattern, we should have one instance of an object available in order to be able to copy it. For this purpose, we should have one object for each one of the layers at any time so that we can easily copy it. As a result, there is a need for a class or several static fields so that we can have one instance from each layer (model, controller) at the runtime and copy them for subsequent instances. This class is useful both for maintaining the initial object of the layers and for wrapping the layers. Wrapping the layers makes complexity of the layers and their design hidden to the users. This is the very idea used in façade design pattern. As a result, besides the layers of model and controller, we need another class (named FactoryPerson here). The class diagram of person entity with MVC+ pattern using prototype pattern has been shown in the following figure:

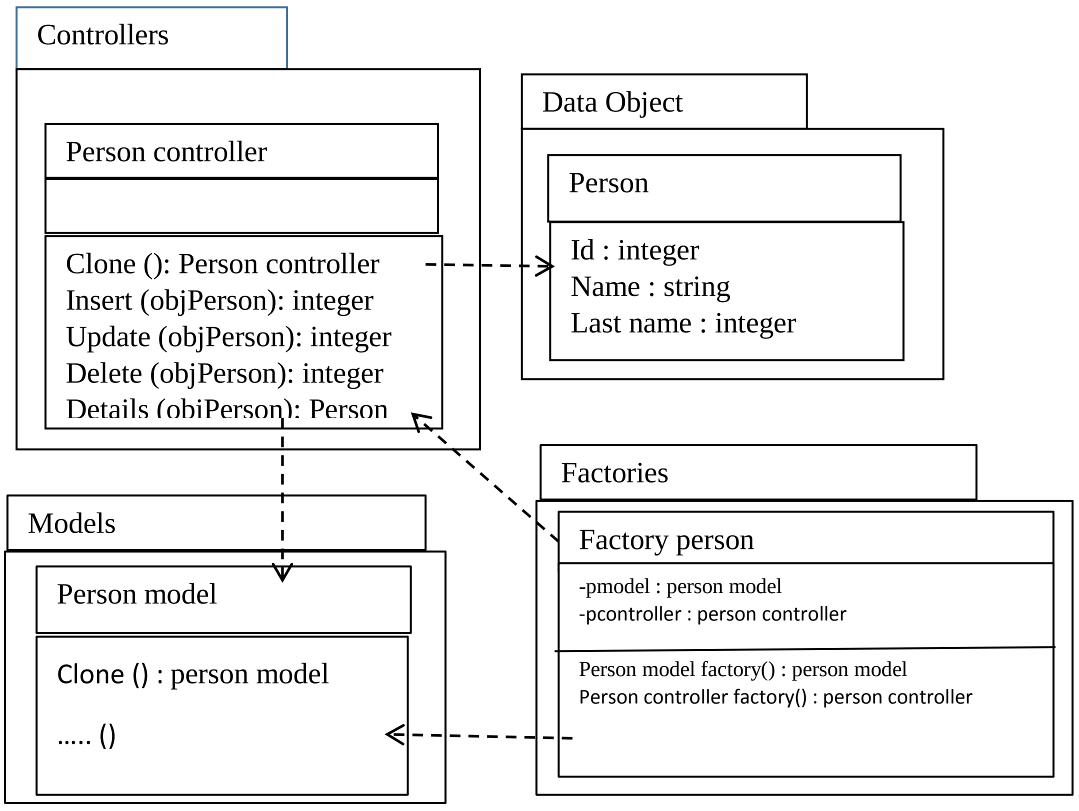

Controllers
Person controller
Clone (): Person controller
Insert (objPerson): integer
Update (objPerson): integer
Delete (objPerson): integer
Details (objPerson): Person
Data Object
Person
Id : integer
Name : string
Last name : integer
Models
Person model
Clone () : person model
..... ()
Factories
Factory person
-pmodel : person model
-pcontroller : person controller
Person model factory() : person model
Person controller factory() : person controller

Figure 12. The Class Diagram of Person Entity with Prototype Pattern

The class of FactoryPerson creates an object from each one of the model and controller layers when the first request arrives and copies them for subsequent requests.
Each one of the model and controller layers should have a copy method in order to be copied easily.

4.5. Evaluation of the Suggested Patterns

Memory and time are two key issues in software development and it is costly to create objects from classes [10]. There are various patterns which we can use to reduce the objects creation time, such as prototype pattern and pool pattern, which focus more on creation objects from classes.

Some programs require the construction of a large number of objects. For instance, web applications need a large number of objects due to the high number of requests. Much time and memory is consumed even if these objects are small. The MVC+ and MVC++ patterns focus on the creation time and memory consumption of objects in order to improve the time and memory. We use the formal method in order to evaluate the memory rate consumed and the runtime. This evaluation is only for generation of objects in different patterns in terms of runtime and memory and has no relation with the time when algorithms are run in the program. Since view and data object are nearly equal and an inevitable issue in all patterns, they are ignored for the sake of simplicity of evaluation. In MVC pattern, an object is constructed for the controller and another one for the model per request. It is also the case in Three-tier architecture where an object is created from business layer and data layer per request. This means that two objects are created per request.

4.6. Formal Evaluation

In this section, we compare the proposed patterns with similar patterns (MVC, Three-tier) in terms of efficiency. Generally, if the number of the incoming requests is considered as N in a website, the memory and time rates for MVC, Three-tier, MVC+ and MVC++ patterns can be shown as follow, considering parameters as below:

Req: number of requests
T_{CC}: the time needed to create an object from the controller
T_{CM}: the time needed to create an object from the model
T_{CD}: the time needed to create an object from the data Layer
T_{CB}: the time needed to create an object from the business layer
T_{ALL}: the total time needed
We consider the object creation time a time unit. In the following, we evaluate time and memory separately.

4.7. Temporal Evaluation of MVC Pattern

In MVC pattern, one object of the controller and model is created per request. As a result, the object creation time for MVC pattern can be assessed as follows:

T_{ALL}=Req (T_{CC}+T_{CM})
Req=n
T_{CC}=1
T_{CM}=1
T_{ALL}=n (1+1) =2n

However, we have not considered the View generation time in MVC pattern. This pattern consumes more time than other patterns.

4.8. Temporal Evaluation of the Three-tier Pattern

In Three-tier pattern, one object is created from business and data layers. As a result, the consumption time for object creation in this pattern can be considered in the following way.

T_{ALL}=Req (T_{CB}+T_{CD})
Req=n
T_{CB}=1
T_{CD}=1
T_{ALL}=n (1+1) =2n

4.9. Temporal Evaluation of MVC+ Pattern

In MVC+ pattern, only one object is created for the controller and model per all requests. Since the layers are singleton and only one object is created for them, the runtime has no relation with the number of requests, and the runtime is quite fixed:

T_{ALL}= T_{CC}+T_{CM}
Req=n
T_{CC}=1

T_{CM}=1
T_{ALL}= 1+1=2
Evaluation of MVC++ pattern can be done the same as MVC+ pattern. Only for the parent model object, an execution unit is added for the first request. Time is equal to 3 in MVC++ pattern

4.10. Memory Evaluation

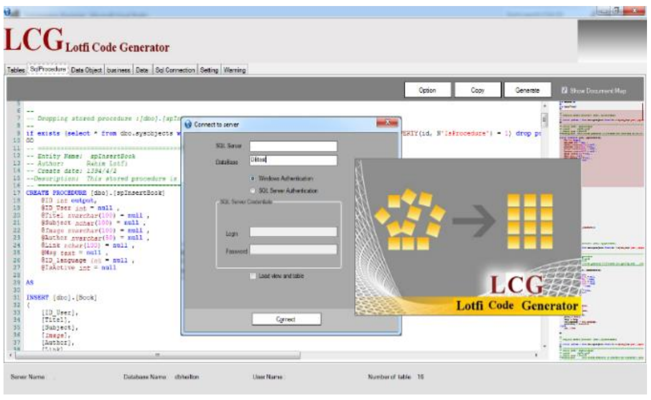
Evaluation can be done for memory consumption exactly like the runtime. Finally, we showed that MVC+ pattern and MVC++ pattern act more optimally than other patterns both in terms of memory and in terms of time. It should be noted that this review is related to the runtime and memory consumption in the Pattern used and has no relation with the runtime of algorithms and activities.

5. Case Study

In this section, we will introduce a proposed code generation tool, called LCG, and show the implementation of a practical example with LCG tool.

5.1. LCG tool

LCG (Lotfi Code Generator) is the tool for code generation based on three-tier, MVC+ and MVC++ architectures. This stored procedure tool generates the codes of the model and controller layers completely for C# language and it can be easily extended for other languages such as Java and vb.net. Figure 14 displays the LCG tool.



```
--
-- Dropping stored procedure :[dbo].[spIn
--
if exists (select * from dbo.sysobjects w                ERTY(id, N'IsProcedure') = 1) drop p
GO
-- ====================================
-- Entity Name:  spInsertBook
-- Author:       Rahim Lotfi
-- Create date: 1394/4/2
--Description:  This stored procedure is
-- ====================================
CREATE PROCEDURE [dbo].[spInsertBook]
    @ID int output,
    @ID_User int = null ,
    @Titel nvarchar(100) = null ,
    @Subject nchar(100) = null ,
    @Image nvarchar(100) = null ,
    @Author nvarchar(50) = null ,
    @Link nchar(100) = null ,
    @Msg text = null ,
    @ID_language int = null ,
    @IsActive int = null

AS

INSERT [dbo].[Book]
(
    [ID_User],
    [Titel],
    [Subject],
    [Image],
    [Author],
    [Link]
```

Server Name   Database Name dbhelton   User Name   Number of table 16

Figure 14. LCG Tool

It is easy to work with LCG tool, after the design of the database in SQL Server environment, the LCG tool can be run and you can log in. The warning tab shows some information from all tables of the database. You can see the list of all your tables and views in the tab of Tables. You can select your desired tables and views and can generate the code of each part separately by going to each tab. In order to generate the code of all parts, you can go to the setting tab and generate the stored procedure. The initial version of this tool that only generated code for Three-tier architecture was tested and used by different programmers. To see the initial version and the programmers' comments, you can use the following link:

http://www.codeproject.com/articles/690105/A-Code-Generating-Program-for-three-layer-Architecture

The naming rules and the points mentioned in section 5.1 were observed in LCG tool. LCG tool can be downloaded from the following link:
http://www.4shared.com/get/CNVXzLDbba/LotfiCodeGenerator.html

5.2. Implementation of a Practical Example

The environment that we have considered for implementation is a language-teaching school (it should be noted that the environment, the analysis method and UML diagrams are not important here, because LCG tool generates the code through the database scheme and is not dependent on the steps before the database design). The class diagram of the language-teaching school is shown in figure 15.

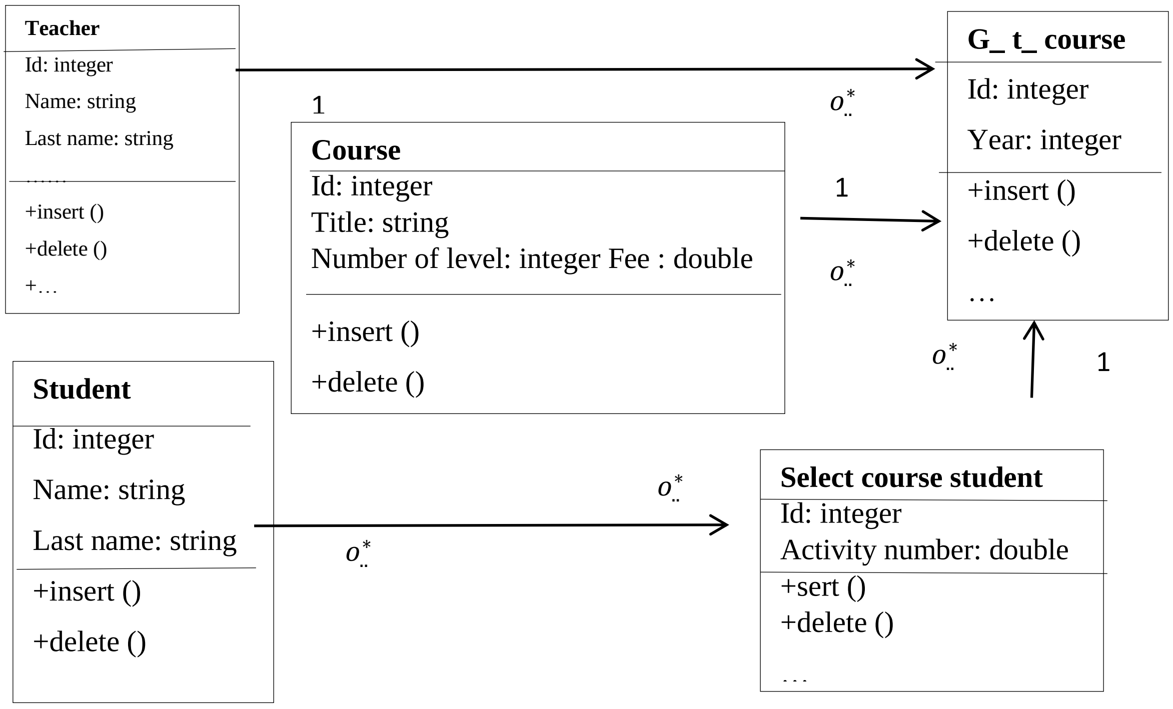


Figure 15. The Class Diagram of the Language Teaching School

All of the codes generated by LCG tool can be downloaded from the link: http://www.4shared.com/rar/sP05XCx3ce/SourceCode.html

6. Conclusion

We first introduced in this article the design patterns of singleton, prototype and MV* family patterns and examined some features of MVC design pattern. We finally propose the two newly designed patterns of MVC+ and MVC++, which are appropriate for designing windows form applications and web applications. These layers make the layers independent from one another so that they can be reused. They also separate the data-generating modules from the data-using ones and are appropriate for code generation. In order to generate the view part, one can generate a user control for each activity (insertion, deletion etc.). If the structure and architecture of the application is useful for code generation, all parts of the program can be generated.

Preventing from the repetition of the construction of the layers that only include codes and methods for execution leads to an increase in the speed and a decrease in the memory consumption.

Here are some points that we have tried to consider in MVC+ and MVC++ patterns:

- Ease of personalization
- Mapping between the elements of the architecture
- Reducing the cost of development
- Make optimal use of programmers with special expertise
- Allocating responsibility

- Coordination model
- Data model
- Resource management

The LCG tool generated the code of programs completely based on MVC+, MVC ++ and three-tier patterns. Code generation using this tool will lead to reduced time and cost of the software development.

References

[1] Z. Moudam and N. Chenfour, "Design Pattern Support System: Help Making Decision in the Choice of Appropriate Pattern," in 2nd International Conference on Computer, Communication, Control and Information Technology(C3IT-2012) on February 25 - 26, 2012.

[2] 2. P. Lan Thung, C. Jian Ng, S. Jing Thung, and v S.Sulaiman, "Improving a web application using design patterns: A case study," 2010 International Symposium in Information Technology (ITSim), vol.1, pp.1,6, 15-17 June 2010 .

[3] 3. A. Syromiatnikov and D. Weyns, "A Journey through the Land of Model-View-Design Patterns," in IEEE/IFIP Conference on Software Architecture (WICSA), pp.21,30, April 2014.

[4] 4. Whittle, Jon, John Hutchinson, and Mark Rouncefield. "The state of practice in model-driven engineering."

[5] Software, IEEE 31.3 (2014).

[6] 5. Sam Malek, George Edwards, YuriyBrun, HosseinTajalli, Joshua Garcia, Ivo Krka, NenadMedvidovic, MarijaMikic-Rakic, Gaurav S.Sukhatme. "An architecture-driven software mobility framework." Journal of Systems and SoftwareVolume 83, Issue 6, Pages 972-989

[7] Sep. 2015http://www.oodesign.com

[8] E. Gamma, R. Helm, R. Johnson, and John M. Vlissides, Design Patterns Elements of Reusable Object Oriented Software, Addison-Wesley Professional ,August 1997.

[9] Glenn E. Krasner and Stephen T. Pope, "A cookbook for using the model-view controller user interface paradigm in smalltalk-80," J. Object Oriented Program, vol. 1, no. 3, pp. 26–49, Aug. 1988.

[10] 9. M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, and R. Stafford, Pattern of Enterprise Application Architecture, Addison Wesley, (2002)

[11] NelioCacho, Claudio Sant'anna, Eduardo Figueiredo, Francisco Dantas, Alessandro Garcia, Thais Batista, "Blending design patterns with aspects: A quantitative study,"Journal of Systems and Software, Volume 98, Issue null, Pages 117-139

[12] . L. Bass, P. Clements, and R. Kazman, Software architecture in practice 3th ed,Addison-Wesley Longman Publishing Co, (2013).

[13] Potel, Mike. "MVP: Model-View-Presenter the taligent programming model for C++ and Java." TaligentInc (1996).

[14] Jarnjak, Fran. "Flexible GUI in robotics applications using Windows Presentation Foundation framework and Model View ViewModel pattern." 4th International Conference on New Trends in Information Science and Service Science. 2010.

[15] B. Gao, S. Zhang, and N. Yao, "A Multidimensional Pivot Table Model Based on MVVM Pattern for Rich Internet Application," in 2012 International Symposium on Computer, Consumer and Control (IS3C), pp.24,27, 4-6 June 2012.

[16] 15. M. Polo, J. ngel Gmez, M. Piattini, and F. Ruiz, "Generating three-tier applications from relational databases: a formal and practical approach," Information and Software Technology, vol. 44, no. 15, pp. 923-941, December 2002

Author

Rahim lotfi
Degree: MSc
Department of Software Engineering, University of Isfahan, Isfahan, Iran
Thesis Title: A New Pattern for Architectural Design of Enterprise Applications to Facilitate Automatic Code Generation