



International Journal of Emerging Technologies in Computational and Applied Sciences (IJETCAS)

www.iasir.net

APPLICABILITY OF AOP ON DESIGN PATTERNS FOR DATACENTER MONITORING

Bhagyashri Khodke^a, Dr. Emmanuel M^b

^aME IT Student, PICT, Pune University, Pune, Maharashtra, India.

^bHOD, Department of Information Technology, PICT, Pune University, Pune, Maharashtra, India.

Abstract: A Data Centre (DC) is a facility used to house computer systems and their related components, such as storage and communications systems. It usually also includes backup power supplies, environmental controls, data communications connections and security devices. Crucial aspects of most organizational operations around the world are IT operations. One of the main concerns is business stability; companies rely on their information systems to run their operations. Company operations may be impaired or stopped completely if a system becomes unavailable. A DC must therefore keep high standards for assuring the integrity and functionality of its hosted IT environment. The computer infrastructure in datacenter needs to be monitored for high availability. The software used for monitoring the DC must adhere the design principles such as reusability, modularity, maintainability etc. Several design patterns used in monitoring software cross cut rudimentary structures of classes. It is observed that many aspects are scattered across the functionalities of the monitoring software. This encouraged us to examine the impact of Aspect Oriented Programming (AOP) on the software designs used in various software products in Data Centre. In this paper, impact of AOP on a few gang of four design patterns is analyzed along with an implementation case study of Singleton Pattern.

Keywords: DataCenter, AOP, Aspect, OOP, Software Design Patterns and Singleton Pattern.

I. Introduction

Object oriented Programming (OOP) took the software programming to the next level. A design pattern in programming is a solution that can be reused to commonly repeated issue in a specific design. It is nothing but a model for how to solve a problem, which can be reused in many different conditions [6]. In today's IT era, speed of software development matters and it can be achieved only by using well known design patterns. It is all about the object creation and their interactions in terms of object oriented concepts.

Creational Design patterns abstract the instance creation process. Following is the list of patterns involved under the category of creational patterns:

1. Abstract Factory: Provides an interface for creating families of related or dependent object without specifying their concrete classes.
2. Builder: Creation of object is separated from its representation so that the same creation process can be used for different representations.
3. Prototype: Specify the kinds of objects to create using prototypical instances, and create new objects by copying the prototype.
4. Singleton: Ensures that a class can be instantiated only once, and give a global access point to it.
5. Factory Method: Define an interface and allow subclass to decide which class to instantiate for object creation. Factory method allows the class to defer instantiation to subclasses.

There are two themes in creational patterns. They all encapsulate the knowledge about which concrete classes the software uses and subsequently they hide how instances of the classes are created and put together. Sometimes creational patterns are competitors and sometime they are complementary. For example Prototype complements singleton whereas prototype and abstract factory are competitors.

The OOP helps modularizing code but do not address crosscutting concerns that must be included in multiple modules [3]. Code tangling is seen if a module handles multiple concerns at a time, the code from other modules present in a module. Code scattering is seen when a single problem is implemented in multiple modules. Since crosscutting concerns are spread over multiple modules, its corresponding implementations also scattered between multiple modules. Code tangling and code scattering result in lower code reuse. If any module is implementing multiple facts, it is difficult to maintain and update. If any concern is scattered in many modules or components, if the component requires reimplementations, many components have to be updated. This violates of Open Close Principle [4]. Unfortunately, mostly object oriented abstractions are unable to modularize the crosscutting concerns, which in turn minimize the system maintainability and reusability. Many design patterns crosscut the rudimentary structure of classes modifying roles, adding behavior in relationships. An IT concern is the information that has an impact on the code of a particular program. Cross cutting concerns are aspects of the software that affect other concerns. Cross cutting represents the situation when a concern is

met by inserting code into objects by the system but the code itself does not relate to the functionality designed for those objects [8]. The concerns may not always be decoupled from rest of the code in the design and implementation of software product. Error and exception handling, synchronization, logging, scheduling and optimization are a few examples of crosscutting concerns. AOP aims to encapsulate some of the cross cutting concerns into aspects to retain modularity. Compared to OOP, AOP offers one possible solution, repetitive code chunks are exhibited in various aspects that crosscut other modules [1]. In AOP, the main aim is to help programmer in the task of clearly isolating crosscutting concerns, using mechanisms to abstract and compose them to produce the software product. The aspect-oriented programming extends other programming techniques (structured, object oriented or functional) that do not offer suitable abstractions to address crosscutting [5]. In case when we apply the aspect oriented principles the implementation becomes spotless which easily develops and aids to remove the crosscutting concerns of the OOP [3]. In this mechanism of AOP, every object has one responsibility. Although there are benefits of AOP versus OOP, programmers must be aware of AOP Language for implementation [4].

II. Study Format

We have divided the paper in a few sections as follows; Section III will highlight some of the challenging work done in the field of aspect oriented in general, and issues in traditional implementation. Section IV will focus on the basics of singleton pattern and its design with respect to OOP and AOP. Section V will focus on the IT Infrastructure monitoring example, its object oriented perspective and the effect of aspect orientation on it. Section VI will focus on implementation and analysis of the creational – singleton design pattern with case study of IT Infrastructure monitoring using OOP. Section VII will focus on the implementation and analysis of the creational – singleton design pattern with case study of IT Infrastructure monitoring using AOP. With this example, we will talk about cross-cutting, code scattering and ways to reduce it using AOP. Finally, the conclusion is presented.

III. Related Work

Almost all monitoring software's available in market contain common functionality like authentication, cache management, logging, exception handling and verification [7]. Many of these monitoring software products follow the ITIL standards. However, these products code is also prone to code scattering and crosscutting concerns. They could also be traditionally coded as one structural program. A good architectural design for addressing the issues of crosscutting concerns is required [9]. The software design patterns give reusable solution in software designing and implementation. It is very important to analyze the AOP to reduce the code scattering and crosscutting concern so that we have good quality supportable, reusable, software product. We analyzed difference between usage of Object Oriented and Aspect Oriented terminologies at the design phase. It was observed that crosscutting concern is present in software which can be minimized using the aspect oriented approach at the design phase [3]. Most of the surveys and studies conducted in this area of software development are not extensive. They are limited to theoretical work. Some studies were done to compare OOP versus AOP implementations based a metrics of measurements measuring the cohesion, coupling and scope of both the implementations [9]. In object-oriented implementation for singleton, client's requires to call global access point of Singleton class instead of calling the usual constructor. This means as many clients to the class that many access point call. Aspect-oriented approach tries to resolve this problem. AOP allows defining an aspect that modifies the behavior of constructor, which is invoked by new operator to create new instance of a class. The singleton pattern could be used in many IT Infrastructure monitoring software products like event management, impact management etc. The monitoring software typically complies with the well-known ITIL Standards. ITIL standards are nothing but set of practices for IT infrastructure and services management [1]. They also talk about aligning IT infrastructure with business etc. Availability of the IT infrastructure could be managed using any of existing event management system which complies with ITIL.

IV. Singleton with OOP and AOP

The intent of Singleton Pattern ensures that a class can be instantiated only once, and give a global access point to it. A class implementing Singleton pattern itself is responsible for keeping track of its sole instance. An operation named getInstance can be provided which allows callers access its unique instance and responsible for creating its own unique instance. Client can access singleton instance only through Singleton's getInstance operation.

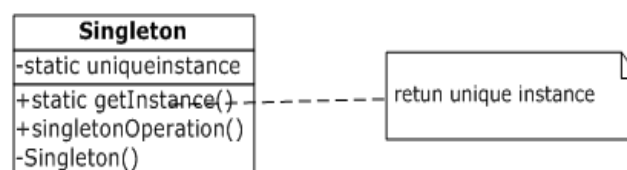


Figure 1: Singleton Pattern in OOP

Advantages

- Singleton class encapsulate its sole instance, it can have strict control over how and when client access it.
- Singleton pattern is flexible to allow more than one instance of Singleton class. The same approach allows controlling number of instances that application uses.
- Singleton pattern is improvement over global variables.

Disadvantages

- More precautions required while defining the Singleton in multithread environment. Singleton Pattern out of box is not thread safe.
- When an application requires more than one class to be defined as singleton, each class needs to implement same lines of code.

In object-oriented implementation singleton, client's requires to call global access point of Singleton class instead of calling the usual constructor. This means as many clients to the class that many access point call. Aspect-oriented approach tries to resolve this problem. AOP allows defining an aspect that modifies the behaviour of constructor, which is invoked by new operator to create new instance of a class.

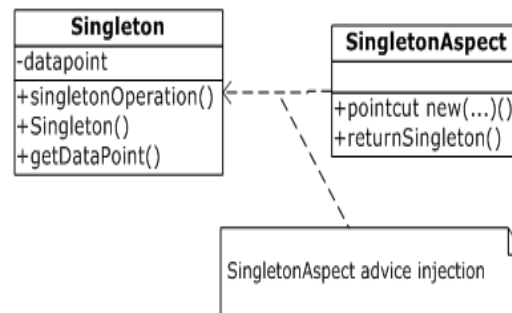


Figure 2: Singleton Pattern in AOP

Above diagram shows how singleton pattern works in aspect-oriented implementation. The aspect called SingletonAspect is defined to have point cut on new operator. So, whenever any client invokes new on Singleton class, the aspect will inject the advice defined to return only single instance of that class.

Advantages

- AOP implementation keeps all the advantages of Singleton pattern. In addition to that it reduces the repetition of code when multiple classes need to be implemented as singleton.
- It also reduces the amount of change require when there exists needs for a class to revoke Singleton behaviour. At this point client remains unchanged.

Disadvantage

- If there are multiple Singleton classes in the system, a hierarchy of aspect may need to define if the source singleton classes are not fowling under same hierarchy.

V. Monitoring Example

Consider IT Infrastructure Monitoring System in Data Centre. The term IT Infrastructure is defined in ITIL V3 as a combined set of hardware, software, networks including all of the information technology in order to develop, monitor, test and support IT Service.

IT Infrastructure Monitoring System provides the entry point for many service operation processes and activities. The Objective is to provide the way of comparing the actual performance and behavior against the design standards and service level agreements.

- Provides ops information, as well as errors and warnings, to aid automation
- Supports continues service improvement activities of service assurance and reporting and service improvement.

Event management can be applied to any aspect of service management that needs to be controlled. These comprise:

- Monitoring the Services
- Monitoring the database systems
- Monitoring the Disks
- Security (e.g. intrusion detection)

Consider the above design for an IT Infrastructure Monitoring System, the system consists of Subjects as AssetMonitor and observers as AssetObserver. The system may require monitoring multiple services, process and disk on the same or from the different machines in infrastructure. The user interface of the system is unique towards any type of monitor. Also activity of send an email when critical event occurs in the system can be

served by single object. Activity of updating database on event occurrence also needs only single instance to indicate something is change with what is change. This indicate that AssetObserver i.e. observers on the system are good examples for singleton pattern. The functionality can be provided completely by keeping single instance of each observer type in the system. Keeping multiple instances of observes may lead to error prone situations, run-time exceptions.

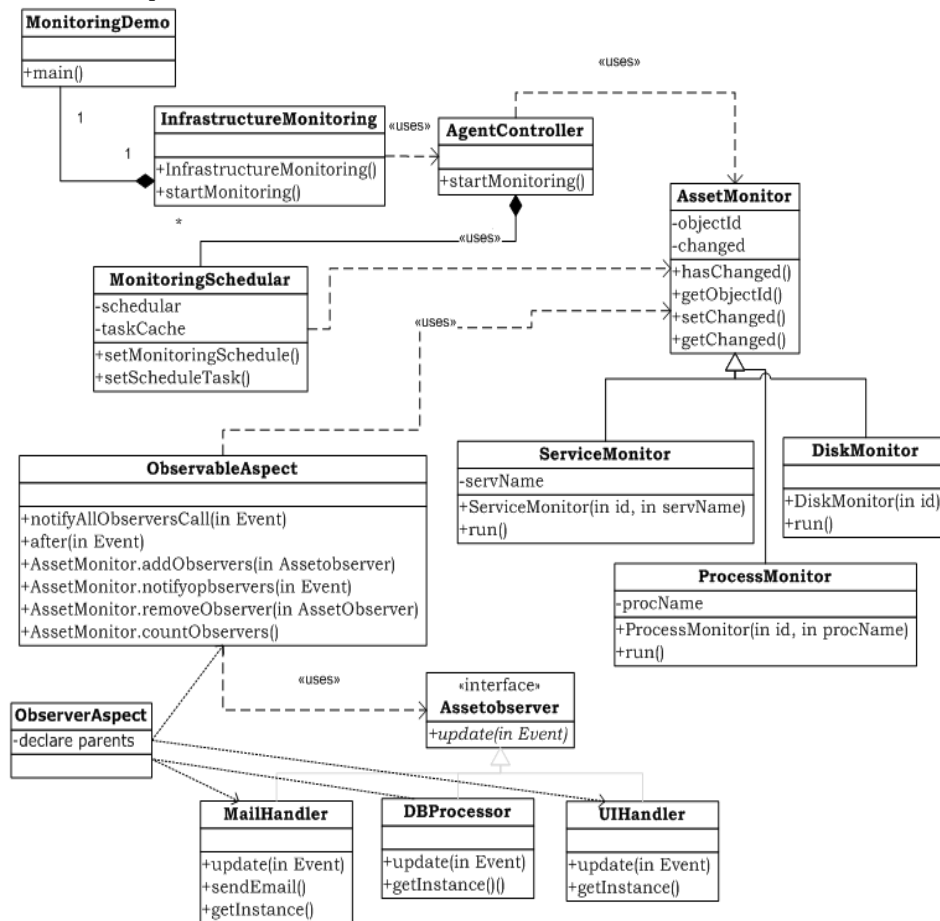


Figure 3: Overall structure of IT Infrastructure Monitoring System Class diagram.

VI. OOP Implementation

Object-oriented implementation of singleton pattern in the system is made for all types of observers. Following is the class diagram for the observers in the system defined as singleton. Here, each observer is having private constructor and implements the getInstance() method. Each of these singles classes is responsible for maintain their own instance. The client accesses these classes by invoking their static method.

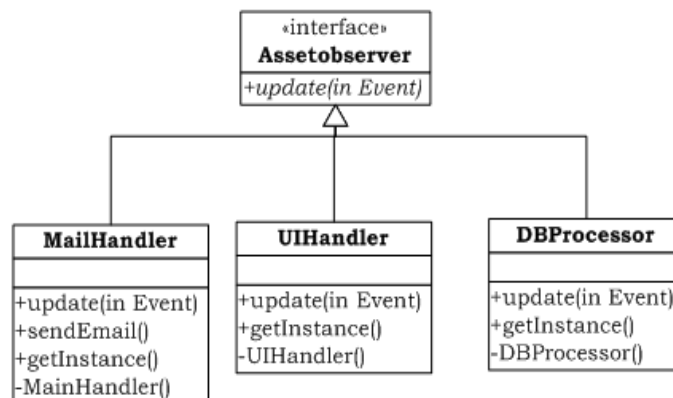


Figure 4: Singleton Pattern in Infrastructure Monitoring System, OOP implementation

Following are the code snippets: -

1) *DBUpdater*

```
public static DBUpdater getInstance()
{
    if( dbupdater ==null)
    {
        synchronized (DBUpdater.class)
        {
            if( dbupdater ==null)
            {
                _dbupdater = new DBUpdater();
            }
        }
    }
    return _dbupdater;
}
```

Database processor (updater) observer is define with private constructor and getInstance() function is provided as global access point for the client. DB Updater defined separately as a singleton class.

2) *UIHandler*

```
public static UIHandler getInstance()
{
    if (null == instance)
    {
        instance = new UIHandler();
    }
    return instance;
}
```

UI handler observer is define with private constructor and getInstance() function is provided as global access point for the client. UI handler is defined separately as a singleton class.

3) *MailHandler*

```
public static MailHandler getInstance()
{
    if( emailHandler ==null)
    {
        synchronized (MailHandler.class)
        {
            if( emailHandler ==null)
            {
                _emailHandler = new MailHandler();
            }
        }
    }
    return _emailHandler;
}
```

Mail handler is define with private constructor and getInstance() function is provided as global access point for the client. Mail handler defined separately as a singleton class.

In the Infrastructure Monitoring system, we have all observers implemented using singleton pattern. Each of these class implements getInstance() method. If we check, the only difference in all these implementation is the constructor class. This indicate the apart from calling constructor entire lines are getting repeated in all of them. Also, developer may make mistakes in repeating the code as seen in the UI Handler singleton class, the double check of instance is missing, which is not safe.

As client needs to invoke getInstance method instead of calling new operator, client subject to change if there exists the need to revoke singleton property.

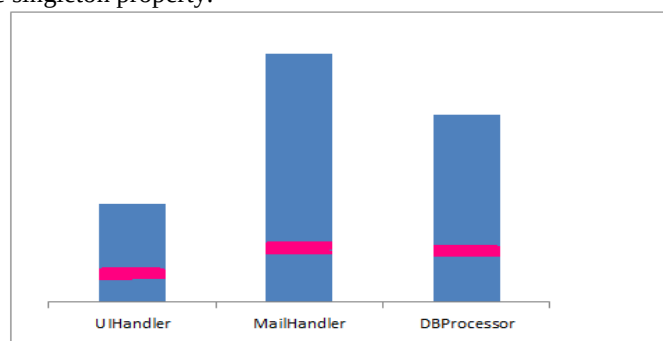


Figure 5: Code scattering with singleton Classes

VII. AOP Implementation

AOP introduces an aspect to implement the crosscutting and code scattering behavior of singleton pattern when an application contains multiple singleton classes. In AOP implementation we have added SingletonAspect which takes care of returning and maintaining single instance of class. SingletonAspect also makes observers free from unique object maintenance and client can get their instance by invoking new operator on class normally.

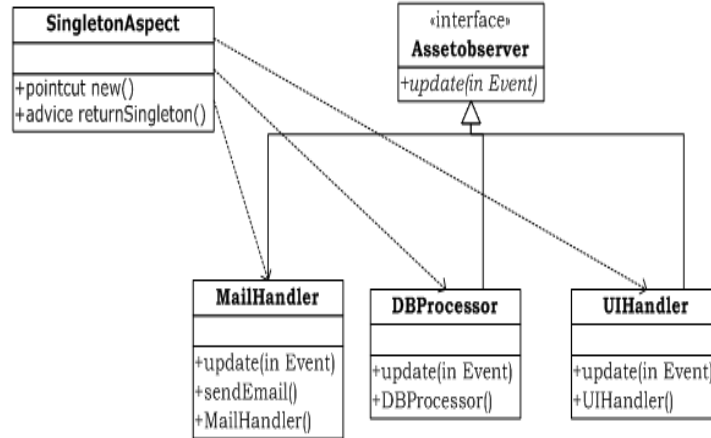


Figure 6: Definition of Singleton Pattern, AOP Implementation

SingletonAspect contains Hash Table which keeps the track of objects per class. The join point is established by defining the pointcut on new operator of a class. In the system, all required classes i.e. observers implements single interface AssetObserver. So we can easily define the point cut on every implementer of AssetObserver rather than on individual class.

1) Singleton Aspect

```

Public aspect singleton
{
    Declare parents: monitoringscheduler extends object;
    Private hashtable<class, object> singletons = new hashtable<class, object>();
    Pointcut returnsingleton(): call((assetobserver +).new (..));

    Object around() : returnsingleton()
    {
        Class singleton = thisjoinpoint.getsignature().getdeclaringtype( );
        Synchronized(this)
        {
            If (singletons.get(singleton) == null)
            {
                Singletons.put(singleton, proceed( ));
            }
        }
        Return (object) singletons.get(singleton);
    }
}
    
```

Singleton Aspect defines an abstract point cut with new operator of a class. It makes sure that one and only one instance is running in system for class whose new operator is called. Around() advice traps the execution of joint point. The proceed() call is used to invoke action of getting single instance for observer class. Singleton Aspect provides a definition for returnsingleton() pointcut which provides single instance for requested observer class.

2) UIHandler

```

public class UIHandler
{
    private static UIHandler instance = null;
    public void update (Event event)
    {
        System.out.println("Update called");
        OperationConsoleBuilder.getInstance().addrowToCurrentView(event);
    }
    public UIHandler()
    {
        System.out.println("I am called UIHandler");
    }
}
    
```

While displaying event information on the UI for current event and event history tab every time connection operations with database such as creation and termination are carried out through new operator of UIHandler observer class for inserting and fetching data from database.

3) Monitoring scheduler

```
private void setSchedulerTask(AssetMonitor asset, long startTime)
{
    Asset.addObserver(new uihandler());
    Asset.addObserver(new dbprocessor());
    Asset.addObserver(new mailhandler());
}
```

Monitoring scheduler class calls a new operator for UIHandler, DBProcessor and MailHandler observer. IT also responsible for the get the scheduler, set timer for tasks, maintain cache for timer tasks, add the task to cache.

4) DBProcessor

```
public class DBProcessor
{
    private static DBProcessor dbupdater = null;

    @Override
    public void update(Event event)
    {
        System.out.println("In DbUpdater saving event");
        DbUpdater proc = new DbUpdater();
        proc.SaveEvents(event);
        OperationConsoleBuilder.getInstance().addrowToHistoryView(event);
    }

    public DBProcessor()
    {
        System.out.println("I am called: DBUpdater");
    }
}
```

While inserting and updating event information to the database every time connection operations such as creation and termination are carried out through new operator of DBProcessor observer class which are written in database aspect. Aspect-oriented way provides solution when multiple classes in an application implements singleton. We have defined aspect SingletonAspect which takes care of having only single aspect of each of observer class. With definition of SingletonAspect, each observer is now implements public constructor, thus no other global point of access is required. Client using these classes need to invoke new operator on class to access the functionalities defined by class. This also eliminates the requirement for modification in client, if one of the classes needs to revoke singleton property. Only change is required in SingletonAspect.

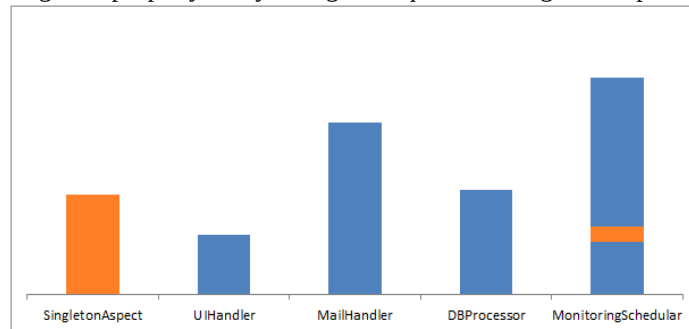


Figure 7: Injection on Singleton in AOP implementation

When client MonitoringScheduler calls new operator for observer, SingletonAspect injects the code which ensures that the classes whose new operator is called has one and only one object in system running. The aspect defines the point cut on any implementer of AssetObserver, so if there is need to add new observer in the system; aspect remains unchanged. Following figure shows how join point is established at new operator call. So, we observed that AOP follows the design principals and avoids the code scattering and code tangling resulting in reducing crosscutting concern. In order to assess the patterns implementations with OOP and AOP, it is necessary to use some performance metrics.

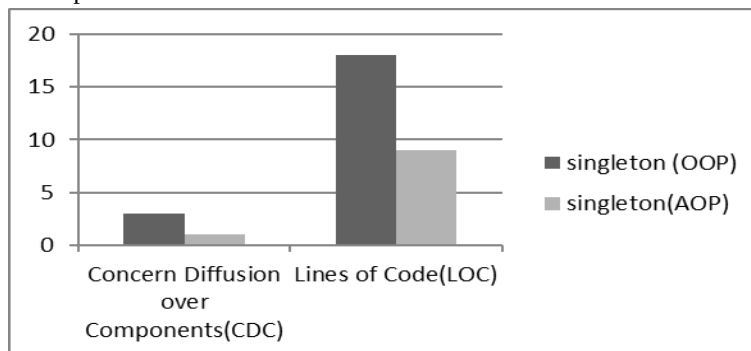


Figure 8: Quantitative assessment of Singleton design pattern

CDC Counts the number of classes and aspects whose main purpose is to contribute to the implementation of a concern and the number of other classes and aspects that access them and LOC count the lines of code. The quantitative assessment of the IT Infrastructure Monitoring system, done for both versions, showed that the software quality improved with AOP implementation Singleton pattern improves the quality and modularity of the system when used in aspect-oriented way.

Conclusion

The study shows that singleton pattern has limited capabilities and can cause execution errors. One of the problems with singleton is multiple constructors. Once the instance is created we always return the same instance irrespective of type of constructor called. One of the solutions that AOP provides is to keep one instance per type of constructor per class. Here the advice needs to understand number of parameters passed and return the object corresponds to called constructor type. Data Centre runs on IT Infrastructure, and IT Infrastructure needs to be monitored for performance and availability. While there are many monitoring software's available in the market they have quality issues because of code duplication, scattering and cross cutting. Applying AOP to design patterns lead to building effective and quality solutions. Implementing IT Event Management which has Singleton Pattern under the umbrella of Aspect Orientation decreases the crosscutting and code scattering concern and improves the extensibility, reusability, maintainability and transparency.

References

- [1] Weidlich M, Ziekow H, Gal A, Mendling J, Weske M, "Optimizing Event Pattern Matching Using Business Process Models", IEEE Transaction on Knowledge and Data Engineering, Vol.26, Issue 11, Nov. 2014.
- [2] Boticki I, Katic M, Martin S, "Exploring Educational Benefits of Introducing Aspect-Oriented Programming Into a Programming Course", IEEE Transaction on Education, Vol.56, Issue 2, May 2013.
- [3] El Maghawry, N., Dawood, A. R., "Aspect oriented GoF design patterns", Informatics and systems (INFOS), The 7th International Conference, March 2010.
- [4] Ramnivas Laddad, "AspectJ in Action: Practical Aspect Oriented Programming", Addison- Wesley, 2005.
- [5] Gamma, E. et al, "Design Patterns- Elements of Reusable Object Oriented Software". Addison-Wesley.2009.
- [6] Lori M Kaufman, Bruce Potter. "Monitoring Cloud Computing by layer - part1 and part2", Security & Privacy, IEEE, vol. 9, no. 2, pp. 66 – 68, April 2011.
- [7] Eduardo Kessler Piveta, Luiz Carlos Zancanella, "Observer Pattern using Aspect- Oriented Programming" SugarloafPloP Conference, 2003.
- [8] Christa Schwanninger, Egon Wuchner, Michael Kircher, "Encapsulating Crosscutting Concerns in System Software", In Proceedings of Third AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software, 2004.
- [9] Wang Xiaohuan, Zhang Shu, Tang Wanmei, "A New Programming Fashion in OOP and AOP", Journal of Chongqing University of Arts and Sciences (Natural Science Edition), Vol.26 Issue14, pp.41-44, 2007.