Separation of Concerns and Code Enhancement: Aspect-oriented Programming Versus Customization Approach Followed in Open Source Software

Sidra Sultana
Department of Computer Software Engineering
National University of Sciences and Technology (NUST)
Islamabad, Pakistan
sidra.sultana88@gmail.com

Fahim Arif
Department of Computer Software Engineering
National University of Sciences and Technology (NUST)
Islamabad, Pakistan
fahim@mcs.edu.pk

***Abstract*— In order to facilitate the separation of concerns and code enhancement without modifying the original code, open source software (OSS) offers a package containing the core code. Depending upon the design or architecture pattern followed in the specified package, the ways to facilitate code enhancement are provided. Hook Architecture is followed in Wordpress, Drupal, etc., in customizing plugins or modules, and Model View Controller (MVC) pattern is followed in Joomla, open source content management systems. Aspect-oriented Programming (AOP) is a programming paradigm that addresses the same code scattering and code tangling issue, and thus, ensure code enhancement without modifying the core code. The research question is whether AOP supports the separation of concerns and allows the enhancement in functionality without modifying the core code; then, hook architecture and other open source customization patterns are there to facilitate the goal. What different features does it offer, as compared to AOP? This research paper differentiates between the separation of concerns and code enhancement addressed by OSS and AOP.**

Keywords-Aspect-oriented Programming (AOP); Open Source Software (OSS); Advice; Joinpoint; Pointcut; Hook Architecture; MVC pattern; Aspect-oriented Model View Controller (AOMVC)

I. INTRODUCTION

Aspect-oriented Programming (AOP) [1] is a programming paradigm that complements Object-oriented Programming (OOP) [2] by separating *concerns* of a software application to improve modularization. The separation of concerns (SoC) aims at making software easier to maintain by grouping features and behavior into manageable parts, which all have a specific purpose and business to take care of. It is the decomposition approach followed in the conventional modular programming that leads to code tangling (code mingling) and code scattering (replication and duplication of same code chunk at many places).

Third party tools, off-the-shelf components, and open source modules are there to be used by the current application; if the application is flexible enough to utilize it without modifying the core code and by simple joining the new functionality from a point where changing (adding/removing) additional code is easier to maintain.

Thus, an effortless and unified approach is offered by AOP in terms of making the dynamic switching of complete features along with providing the conciseness, evolution, and testability. Aspect-oriented approach focuses on the argument related to the maintainability and readability of the constructed software.

Section II offers a brief literature survey. The comparative analysis is performed in Section III. Research Results are presented in Section IV. Section V provides a discussion. Conclusion is given in Section VI.

II. LITERATURE SURVEY

AOP is designed to formulate code easier to query about, trace, develop, enhance, maintain, and modify certain verity of application code. For the sake of validating these potentials claimed by AOP and to verify the impact of AOP on the program structure, Robert et al. conducted two investigatory experiments [1]. AspectJ version 0.1 [14] was the language in which the requirements are implemented to trace change and debugging process supported by AOP. Developer's ability to trace and then resolve the issues (programming fault) of the multi-threaded program is analyzed in the very first experiment. In the other experiment, existing distributed system is focused on checking the ease in change management provided by the AOP.

A. Modularization in AOP

Kiczales et al. [2] have familiarized AOP for providing more organized and well managed way of capturing the code while enhancing the scope of the program concerns. Software programmers explicitly manage the separation of some concerns within the code by the help of built-in functionalities provided by the selected programming language. Explicit language support is provided by AOP to help functional decomposition in the program and to be well modularized upon the design decisions.

B. Usability of AOP

Usability and usefulness of AOP are well proved in the experimental results [3]. The core code that is functionally decomposed and aspects' interface has some characteristics highlighted by the experiment, to show that programming benefits can be accrued best with the understanding of it. Vital feature as per the completeness point of view of AOP approach is that, it is beneficial in totality [4]. This refers to the fact that partial benefits cannot be extracted by the partial implementation of separation of concerns. Well

 ISBN: 978-1-61208-304-9

defined scope of the aspect effected across the boundaries, is necessary to provide the refined (narrowed) scope of the aspect without digging deep the core code for extensive analysis. Thus, when the separation is more complete, i.e., interface is narrow, only then the AOP approach will be more promising [5] [6].

C. Design Quality in AOP

With regards to the design quality and software development efficiency [7], a web based system is developed to empirically study its behavior in both AOP and OOP fashion. The study reveals that if the number of subjects undertaken in the experiment increases, then benefits offered by AOP will be much more as compared to those underline in the present study. To produce high quality, design aspects are very vital so, Madeyski et al. [8] aimed at providing empirical evidence of the impact of AOP on design quality metrics and software development efficiency.

III. Comparative Anaysis

In order to facilitate the separation of concern and code enhancement without modifying the original code, OSS provides with a package containing the core code. Depending upon the design or architecture pattern followed in the specified package, the ways to facilitate code enhancement are provided. Hook Architecture is followed in Wordpress, Drupal, etc., in customizing plugins or modules, while MVC pattern is followed in Joomla, FLOW3, etc., open source content management systems. AOP is a programming paradigm that addresses the same code scattering and code tangling issue and thus, ensure code enhancement without modifying the core code. The research question is whether AOP supports the separation of concerns and allows the enhancement in functionality without modifying the core code; then, hook architecture and other open source customization patterns are there to facilitate the goal. What different features does it offer, as compared to AOP?

For the comprehensive analysis, three aspects are implemented in FLOW3 (an open source framework) to address all cross cutting concerns in components of MVC.

For potential cross-cutting concern in Model Class, Logging Aspect is used to log the delete details, in other case; it can be mistakenly added as a part of business logic in Model class of the package.

To address potential cross-cutting concern in View Class, Flash Message Aspect is used to inject html element (i.e., styled div) with specific list of actions, thus addressing the cross cutting concerns at interface level or View class of the package.

For potential cross-cutting concern in Controller Class, Manipulation Aspect is used to provide control access for number of controller's actions so in terms of addressing control flow, manipulation aspect resolves cross cutting concerns in Controller Class.

Kato et al. [21] also presented the Context-Oriented Programming implementation along with the OOP and AOP comparison but lacking the comprehensive metrics analysis. The novelty of the conducted research lies in the wide domain discussion of the concerned problem in functional and non-functional requirements domain like maintainability, re-usability, scalability, code organization, dynamics, etc.

This section differentiates between the separation of concerns and code enhancement addressed by OSS and AOP and thus, giving an insight of AOMVC and MVC cross-cutting concerns resolved by MVC.

A. OSS

OSS like CMS [8] or frameworks provide with the general package containing backend (administrator view) and front end (user view) of the application. Some of the cross cutting concerns like security (Manipulation Aspect), logging (Modeling Aspect), flash messaging (View Aspect) etc., are addressed by the CMS and frameworks like Joomla, Drupal, Wordpress, YII, Zend, Virteom, Magento, Oscommerce, etc.

Almost all OSS followed certain programming approaches for handling the separation of concerns and demotivates modifying the core code. Mostly MVC or Hook Architecture is followed to code custom components, modules, or plug-ins. It helps in enhancing the application functionality in a flexible adding/removing way.

B. AOP

"Separation of concerns" principle has been used for many years by software engineers to handle the software system's development [9]. Software programmers explicitly manage the separation of some concerns within the code by the help of built in functionalities provided by the selected programming language. Explicit language support is provided by AOP to help functional decomposition program and to be well modularized upon the design decisions.

AOP is made for code enhancement, so that the cross cutting code related to the design decision is not dispersed throughout the program rather it is expressed in a separate set of coherent code chunks [10]. AOP owns a better way of modularizing cross-cutting concerns, resulting in the more readable and less complex developed system implementation.

C. Cross Cutting Concerns

Allowing the modularization of the concerns that usually cross-cut in the object-oriented way of programming application [11], AOP resolved number of programming issues encountered by OOP like code tangling and code scattering, all as result of cross-cutting concerns.

Aspects are declared by using around, after and before advices for the retrieval of properties and intercepting settings.

 ISBN: 978-1-61208-304-9

D. Code Enhancement in OSS

In order to facilitate the code enhancement without modifying the original code, OSS provides with a package containing the core code. Depending upon the design or architecture pattern followed in the provided package, the ways to facilitate code enhancement are specified [12]. Hook Architecture is followed in Wordpress, Drupal, etc., customizing plugins or modules, while MVC pattern is followed in Joomla, etc., Open Source CMS.

E. Code Enhancement in AOP

Code scattering and code tangling are not the only results of implementing security concerns in an application - by following OO approach - but it also because the weaker existence of the security related issues. AOP addresses this code scattering and code tangling issue hence, advocating an improvement in dealing these issues previously in OO way. A number of reasons are there for showing weaker enforcement of security including programming error, inherit design of the system etc.

Conventional software engineering practices failed to modularize cross-cutting concerns and Aspect-oriented Software development offsets this limitation of current software engineering constructs. The advice injected in the point-cut expression is to be bonded after, before or around the code. Also, wildcards (.*) can be used to bind advice with number of join-points. This flexibility of hooking the code at number of places creates the difference and provides an edge to the AOP paradigm.

F. AOMVC

MVC refers to modularizing the application in terms of separating the layers of Control flow and management (i.e., Controller), Interface Design (View) and Database interaction (Model) [13]. MVC framework, in the domain of J2EE [14], has cross cutting concerns throughout the multiple modules (e.g., validation transaction, logging, etc.).

MVC framework is the well-known layered architecture but it has greater limitations and architectural constraints in dealing with cross-cutting concerns. These overlapping concerns lead to code confusion, code tangling and code scattering and finally, result in the difficulty of system maintenance and extensibility. AOP addresses all these problems in every layer of abstraction, i.e., Model, View and Controller. Aspects can be defined to modularize such concerns. All such concerns are well defined by the aspects of AOP.

G. MVC cross-cutting concerns and AOP

The three potential cross-cutting concerns that address almost all components of MVC are presented.

a. Potential cross-cutting concern in Model Class

Logging Aspect is used to log the delete details and hence can be mistakenly added a part of business logic in Model class of the package.

b. Potential cross-cutting concern in View Class

Flash Message Aspect is used to inject html element (i.e., styled div) with specific list of actions, thus addressing the cross cutting concerns at interface level or View class of the package.

c. Potential cross-cutting concern in Controller Class

Manipulation (security) Aspect is used to provide control access for number of controller's actions so, in terms of addressing control flow, manipulation aspect resolves cross cutting concerns in Controller Class.

Thus, by extracting the different cross-cutting concerns from the model, view and controller component of the MVC model, an aspect layer is to be composed to weave with the core functionality.

IV. RESEARCH RESULTS

Some of the factors that distinguished the contribution of AOP and OSS for separation of concerns and code enhancements are: point of access, code management, development time, line of codes, and functional breakdown, etc. These qualitative and quantitative factors that contribute in the estimation of software metrics are analyzed in this section.

A. Point of Access

In case of AOP, aspect classes with variety of advices are defined to be injected at different levels of code. For example, this injection of the wildcard \before ("method (.*Controller->.* Action ())") to all controllers actions will bind the particular advice with all actions of every controller. \before ("method (studentController->.*Action ())") this one-to-many injection will affect all actions of student controller only and \before ("method (studentController->registerAction ())") this one-to-one injection will bind the advice to registerAction of the studentController and for all three injection types, advice will be bound before the action's code. This single class is the single point of access for all related code management in terms of adding and removing the aspect's advices.

For OSS, customization is to be ensured by coding plugins, components and modules as per the coding conventions of the selected OSS. In that case modifications are to be managed in multiple files and thus, there are multiple points of code access that increases the complexity measure.

B. Separation of Cross-cutting Concerns

AOP is designed for handling cross-cutting concerns and thus, resolving them by addressing the code tangling and code scattering issues. Code Tangling refers to the phenomena where the concerns are interwoven with each other in a module. Code Scattering occurs when the concerns are dispersed over many modules. It results in a typical design problem of high-coupling and low cohesion. All the components that are specifically fragmented using the traditional techniques for highlighting their role as a

 ISBN: 978-1-61208-304-9

cross-cutting concern, should be well evaluated. For instance, if a logging functionality is implemented in an aspect-oriented way then in large number of modules invocation to the logger necessitates being present in the model.

The interesting insight of the aspect-oriented implementation is that along with providing the modularized solution to cross-cutting concerns there is no negative effect on software size and system modularity with AOP implementation. If any particular task is to be performed at a lot of places, then that particular functionality, for instance logging, will be the part of the application domain logic. All of the functional dependencies related to logging would be then injected into the model. Logging is not the domain model logic, neither its view nor controller. So, it does not fit in any layer of MVC. Aspect logging is the non-functional requirement and an example of cross cutting concern. Therefore, such concerns should be implemented in a separate layer, i.e., the Aspect Layer. Hook Architecture is followed along with MVC to run the code side by side in most of the AOP applications.

Separation of cross-cutting concerns is not addressed in OOP, thus OOP with AOP is suggested for better modularization and code optimization.

C. Change Management

Due to singularity of Aspect Class, maintainability and change management is easy for AOP. For OSS, plugins and components have multiple files, so need to track all related code in case of any required modification.

Insertion and deletion in case of OSS is also complex like change management and thus affected other related metrics like development time, line of codes, coupling and cohesion etc.

D. Code Enhancement

In case of OSS convention modular code enhancement, scope of the customized or enhanced code is specific to that particular module for customization of the package. And the defined code has a limited impact on the package. For hook architecture (Wordpress and Drupal, etc.), flexibility of hooking enhanced code is ensured through a single function definition instead of multi-files modules or components. But the impact of the hooked functionality is at a single code point and there is no way to hook the same code to multiple points of the package's core code.

Wildcard (.*) access in case of AOP advice binding enhanced the impact to advice to wide variety of code clones. For example, this injection of the wildcard \before ("method (.*Controller->.*Action ())") to all controllers actions will bind the particular advice with all actions of every controller.

E. Development Time

Aspects developer requires one time focus to learn the aspects implementation and once learned she can bind advices of aspects to any desired code clone. As no knowledge of the current system is required for aspects implementation, the development time is optimized by aspects customization and the development time is focused on required functionality instead of replicating and testing the same code at number of points.

For OSS customization, knowledge of the current system is required, so development time is also spent on related modules. As per the OSS architecture and conventions, there are variable maintenance time issues.

F. Line of Codes

In order to measure the size of the set of instructions – the computer program – there is a metric named line of code LOC, which simply shows the count of the number of code lines of program. Maintainability, programming productivity and effort to be required for developing a program are predicted by LOC. As the cross cutting code is resolved at a single point, line of codes are limited. The same code needs to be coded at all required points, so, line of codes are more as compare to that in AOP.

For instance, there is a requirement of making a detailed entry with timestamp in a logger file whenever any record is deleted. For this simple requirement, wherever delete code is written in the package OSS customization approach will handle the case by coding a plugin, component or module to log the details separately for every code. Thus, if the modified functionality is 'm' and number of clones to be modified is 'n', then the m*n is the number of code lines (LOC) increased in case of OSS customization approach.

In case of AOP, LOC increases by 'm*1', meaning that 'm' lines are added in the original LOC. If there is a single point of change, then, the OSS and AOP approaches are equally to adopt but in common practices logging related codes are required at number of joinpoints. This refers to the strong adoptability of the AOP for large scale projects. In the light of this calculation, it revealed that the usability of aspect-oriented technique directly depends upon the size of application. In case there is a large number of code clones then, the AOP will help in reaping maximum time saving benefits whereas the development speed decreases when this technique is used for small number of code clones.

G. Direction of Functional Breakdown

For a student manager, customization in terms of adding student registration functionality, the direction of functional breakdown varies as per nature of the functionality to be focused. For instance, student registration comprises of two main modules, i.e., Managing Student Bio data and Managing Student Courses. Courses Manager is further divided into content manager and batch manager with course information. All these managers are the functional breakdown of registration manager in top to down direction and thus, will be implemented by OSS way of customization as modularization is done in a vertical fashion.

In case of displaying a flash message on every successful insertion of record in registration manager, advices need to

 ISBN: 978-1-61208-304-9

be defined to manage the case in AOP way. For AOP, cross-cutting concerns are handled in the horizontal fashion, i.e., left to right.

Thus, a combination of AOP and OSS customization will be used where the cross-cutting concerns are implemented in AOP to manage code maintainability in single file and other particular module functionalities are implemented in OSS modules, plugins or components.

Summary of these qualitative and quantitative factors that contribute in the estimation of software metrics are tabularized in Table 1.

TABLE I. AOP vs OSS

	AOP	OSS
Point of Access	Single File	Multiple Files
Separation of Cross-cutting Concerns	Resolved	Not Addressed
Code Enhancement	Wide Impact	Limited
Change Management	Easy	Complex
Development Time	Optimized	Increased
Line of Codes	Optimized	Increased (Replication in case of cross-cutting concern)
Direction of Functional Breakdown	Vertical	Horizontal

System having cross cutting concerns can be successfully handled through AOMVC using AOP techniques. AOMVC creates an additional layer of aspects and then declared the aspects in the configuration file in order to provide scalability, maintainability and refined modularization within the system. Also, wildcards can be used to bind advice with number of join-points. This flexibility of hooking the code at number of places creates the difference and provides an edge to the AOP paradigm.

V. DISCUSSION

The potential benefits as per the system's features offered by the AOP approach include the simplicity, readability and modularity. This way, the created system with improved software development efficiency works faster than its object-oriented version.

A. Code Reuse

Reusability of the code refers to the phenomena of writing the code once and using it later on number of occasions as per the scenario defined. Once a code is defined and as per its invocation, it gets weaved and called on multiple locations. Hence, the code duplication is reduced manifold. In case of Manipulation aspect the reusability measure is too high to affect number of code clones. Thus, through single point of access, code gets reused and maintained.

B. Maintainability

System gradation is a part of every real world application. Code once developed has to be maintained and to ensure configuration management application maintainability is a vital concern for meeting user's needs. Instead of tracing the code in each and every file for the modification or deletion purpose, AOP offers a woven point defined as per language selection in XML, PHP, JAVA, .NET etc., in the declarative way, in order to delete the cross-cutting concern if it is no longer in need, which progresses the maintainability of the system compared with traditional methods - one by one steps to locate the code.

C. Scalability

Through scalability, demand for the change in functionality of the original system is facilitated. New functional requirement proposed by the user is coded as an aspect in the form of new feature, specified in the configuration files, woven or bind in a respective point instead of updating number of files required to be modified. Hence, aspects provide scalability for a large amount of changes in the current system in the way to incorporating user's emerging requirements with the passage of time.

D. Reduced Development Time

As the line of code is decreased in case of using OOP with AOP, so the development time gets reduced. In case there is a large number of code clones (as in case of Manipulation Aspect) then the AOP will help in reaping maximum time saving benefits whereas the development speed decreases when this technique is used for small number of code clones.

E. Code Organization

Cross-cutting concerns of logging, flash message and manipulation are kept aside from Model, View and Controller classes in case of coding aspects for logging, flash message and manipulation functionality. Thus, the domain logic is not confused with the supporting domain logic (logging entry in file or database) in case of logging aspect implementation.

F. Changeability

Request for change in web application is too common. With the advent of technology changeability should be offered by the web development. Code once developed has to be maintained and to ensure configuration management application maintainability is a vital concern for meeting user's needs. If in case of Logging Aspect, instead of recording entry in file, requirement got changed to record entry in database then a single line of aspect get replaced instead of replacing code in every related file in case of OOP without AOP.

G. Extensibility

Aspects provide scalability for a large amount of changes in the current system in the way to integrating user's evolving requirements with the project advancement. In case of Logging Aspect, if along with recording deletion time in file, recoding an entry in database is required then a

 ISBN: 978-1-61208-304-9

single line code at one place need to be added in the logging aspect class.

H. Dynamics

Dynamics refers to the enabling and disabling of the aspects. If the injected functionality is no more required then the aspect injection code can be commented. In case of Logging Aspect, if the logging of the delete record is no more required then single line code of recording time of the delete can be commented. Similarly, if the Flash Messages are not to be injected then the code can be commented and same is the case for manipulation aspect.

VI. CONCLUSION

OSS customization mostly follows OOP. Replacing the OOP by AOP was an obsolete question and now it reveals that AOP basically complements OOP and cannot be used in isolation because AOP is developed on the basis of OOP. AOP counterbalances the constraint of OOP. When applied together with OOP, AOP is more efficient and complementary in providing an ideal structure for modular programming.

The scope of aspect-oriented implementation –that either it solves a specific cross cutting concern or it can be applied in general to the whole application – is to be well estimated by the metrics, so that to ensure the risks involved and opportunities offered by AOP. There are several factors that affect the performance of the application like main memory size, memory management, cache size and even program size (line of codes, etc.). Switching between the base code and the aspect is more often resulting in the back and forth movement of the control flows of the system, with the potential increase in the number of join points.

ACKNOWLEDGMENT

Many thanks to Saeeda Sultana Sadia, that we are able to complete the research. It could have been near to impossible to achieve the successful completion without the supervision and guidance of Saeed ur Rehman - Mentor. The cooperation of Department and faculty is heartedly acknowledged. Special thanks to Salma Sultana and Sara Sultana for their support and motivation to complete the research successfully.

REFERENCES

[1] R. J. Walker, E. L.A. Baniassad, and G. C. Murphy, “An Initial Assessment of Aspect-oriented Programming, ICSE99 Proceedings of the 21st international conference on Software engineering”, ACM, ISBN: 1-58113-074-0, 1999, pp. 120-130.

[2] G. Kiczales, J. Lamping, and A. Mendhekar, “Aspect-oriented Programming, Proceeding of 11th European Conference of Object-Oriented Programming”, LNCS 1241, 1997, pp. 220-242.

[3] S. K. Otrappa and P. J. Kulkarni, “Multilevel Security Using Aspect Oriented Programming AspectJ, Advances in Recent Technologies in Communication and Computing (ARTCom)”, 2010 International Conference, IEEE, ISBN: 978-0-7695-4201-0, 2010, pp. 369 – 373.

[4] H. Li, M. Zhou, G. Xu, and L. Si, “Aspect-oriented Programming for MVC Framework, Biomedical Engineering and Computer Science (ICBECS)”, 2010 International Conference, IEEE, ISBN 978-1-4244-5315-3, 2010, pp. 1 – 4.

[5] B. Amar, H. Leblanc, B. Coulette and C. Nebut, “Using Aspect-Oriented Programming to Trace Imperative Transformations, Enterprise Distributed Object Computing Conference (EDOC)”, 2010 14th IEEE International, IEEE, ISBN 978-1-4244-7966-5, 2010, pp. 143 – 152.

[6] S. Hanenberg, S. Kleinschmager, and M. J. Walter, “Does Aspect-Oriented Programming Increase the Development Speed for Cross-cutting Code? An Empirical Study”, ESEM '09 Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement, ACM, ISBN: 978-1-4244-4842-5, 2009, pp. 156-167.

[7] J. Zhang, and Y. C. G. Liu, “Modeling Aspect-Oriented Programming with UML Profile”, 2009 First International Workshop on Education Technology and Computer Science, ISBN: 978-0-7695-3557-9, vol. 2, 2009, pp. 242-245.

[8] L. Madeyski, and L. Szała, “Impact of aspect-oriented programming on software development efficiency and design quality: an empirical study”, Software, IET, IEEE, ISSN 1751-8806, 2007, pp. 180 – 187.

[9] D. Zhengyan, “Aspect Oriented Programming Technology and The Strategy Of Its Implementation, Intelligence Science and Information Engineering (ISIE)”, 2011 International Conference, IEEE, ISBN 978-1-4577-0960-9, 2011, pp. 457 – 460.

[10] M. Bartsch and R. Harrison, “An exploratory study of the effect of aspect-oriented programming on maintainability”, Software Quality, vol. 16, no 1, 2007, pp. 23-44.

[11] R. Coelho et al. , “Assessing the Impact of Aspects on Exception Flows: An Exploratory Study Proceedings of the European Conference on Object-Oriented Programming”, 2008, pp. 207-234.

[12] P. Greenwood et al. , “On the Impact of Aspectual Decompositions on Design Stability: An Empirical Study”, Proceedings of ECOOP 2007, pp. 176-200.

[13] K. Gybels and J. Brichau, “Arranging language features for more robust pattern-based cross-cuts”, Proceedings of AOSD, 2003, pp. 60-69.

[14] M. Kuhlemann and C. Kästner, “Reducing the Complexity of AspectJ Mechanisms for Recurring Extensions, In Proceedings of the Second GPCE Workshop on Aspect-Oriented Product Line Engineering (AOPLE)”, 2007, pp. 14-19.

[15] J. Zhang, Y. Chen, G. Liu, and H Li, “An Aspectual State Model and its Realization based on AOP, Proc. of WRI World Congress on Soft. Eng”., vol.3, 2007, pp. 163-166.

[16] T. Osogami and S. Kato, “Optimizing System Configurations Quickly by Guessing at the Performance, Proc. of ACM Special Interest Group on Measurement and Evaluation”, 2007, pp. 145-156.

[17] C. A. Cunha, “Reusable Aspect-Oriented Implementations of Concurrency Control Patterns and Mechanisms, Proc. of the Aspect-Oriented Software Development”, 2006, pp. 134 –145.

[18] W. Liu, C. Lung, and S. Ajila, “Impact of Aspect-Oriented Programming on Software Performance: A Case Study of Leader/Followers and Half-Sync/Half-Async Architectures”, COMPSAC 2011, 2011, pp. 662-667.

[19] G. Kiczales, E. Hilsdale, and J. Hugunin, “An Overview of AspectJ”, In Proc. ECOOP 2001, LNCS 2072, Berlin, June 2001,Springer-Verlag, 2001, pp. 327-353.

[20] R. Douence, T. Fritz, N. Loriant, J. M. Menaud, M. S. Devillechaise, and M. Suedhol, “An expressive aspect language for system applications with Arachne”. 4th Int. Conf. on Aspect-Oriented Software Development (AOSD ’05), Chicago, Illinois, Mar. 2005. ACM, pp. 27–38.

[21] F. Kato, K. Sakamoto, H. Washizaki, and Y. Fukazawa, “Comparative Evaluation of Programming Paradigm: Separation of Concerns with Object-, Aspect-, and Context-Oriented Programming.” Proceedings of 24th International Conference on Software Engineering and Knowledge Engineering (SEKE 2013), pp. 594-599.

 ISBN: 978-1-61208-304-9