A study on Aspect-Oriented Programming

By -
Rahil S. Vijay (IMT2016062)
Swastik Shrivastava (IMT2016055)
Raja Rakshit Varanasi (IMT2016087)

Abstract

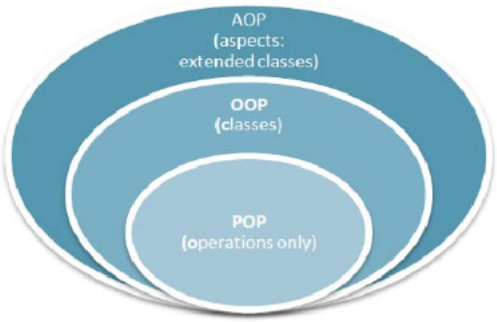
Aspect Oriented Programming(AOP) has been introduced as a potential programming approach for the specification of non-functional programming such as logging, authentication, security. This study discusses AOP concepts, the neccesity that led to it, how it's implemented, it's advantages, it's implementation issues and what next in the field of AOP.

1 Introduction

A typical program is composed of various components. Each of these components has a designated responsibility to complete. Some concerns which cannot be decomposed from the rest of the system in both the design and implementation, and can result in either scattering, tangling or both, are called cross-cutting concerns. Implementation of these concerns is a major issue in old traditional programming approaches such as Object-Oriented Programming, functional programming languages. As a result, Aspect-Oriented Programming(AOP) emerged which aims to solve the problem of Cross-Cutting concern by breaking down the code into two categories "Core Concerns" and "Cross-cutting Concerns" and provides a new approach to deal with Cross-Cutting Concern. It makes the code more modular and simple by better handling the Cross-Cutting Concerns.

1.1 Example

In a bank management system there are various Core Concerns such as account management, balance enquiry, transaction.. etc. These are mainly implemented as functions or classes in OOP and FOP. Some of the other concerns such as logging, authentication, security and many others are an important part of many functions and classes in a program. Therefore these concerns(Cross-Cutting Concerns) are used in multiple locations within the same program, thus reducing the modularity and simplicity of the program. To handle this issue AOP creates aspects which are considered as an extended version of class with some additional features.



2 AOP New Concepts And Terminologies

AOP helps us in making the code modular and simple to understand. This requires new programming concepts and terminologies such as:

- **Aspect -** An aspect is an class that implements Cross-Cutting Concerns that cut across multiple classes such as bank transactions. Aspects can be a normal class configured through Spring XML configuration or we can use Spring AspectJ integration to define a class as Aspect using @Aspect annotation.

- **JoinPoint -** A join point is the specific point in the application such as method execution, exception handling, changing object variable values etc.

- **PointCut -** Pointcut are expressions that is matched with join points to determine whether advice needs to be executed or not. Pointcut uses different kinds of expressions that are matched with the join points.

- **Advice -** Advices are actions taken for a particular join point. In terms of programming, they are methods that gets executed when a certain join point with matching pointcut is reached in the application.Spring aspects can work with five kinds of advice.

  * **Before -** The advice functionality takes place before the advised method is invoked.

  * **After -** The advice functionality takes place after the advised method completes, regardless of the outcome.

  * **After Returning -** The advice functionality takes place after the advised method successfully completes.

  * **After Throwing -** The advice functionality takes place after the advised method throws an exception.

    - **Around -** The advice wraps the advised method, providing some functionality before and after the advised method is invoked.

- **Weaving -** It is the process in which an aspect is added into an object. It can be executed in the compiling time or during the running of the program.

- **AOP Proxy -** AOP Proxy is an Object created by the AOP framework to execute our service. At run time the Spring weaves our service and target object to create the desired proxy object.

- **Spring AOP -** Its a java dependency to enable runtime AOP support in JAVA.

3 AOP Implementation Approaches

This section deals with the types of AOP implementations. Current implementations are based on changes made in the source code or in the object code itself.

- **Compile Time Approach** In this approach, the aspect code is woven into the original source code and a new proxy AOP class is created. This AOP proxy class creates a new method by combining the target method and the aspect where it was called based on the advice. The compile-time approach produces a well formed program, as the code have to pass through a compiler, which may help to optimize the code and improve it's performance. One of the tool that uses compile-time approach is AspectJ, which is a dependency of Java.

- **Run-time Approach** In a run-time environment, one can weave aspects through an application or interface by locating the support for weaving and executing aspects directly into the executing environment. This interface or application uses meta-object protocols (MOP's) to achieve the run-time approach. MOP's provides the protocols to access and manipulate the structure and behaviour of systems of objects. Some of the functions of MOP's includes:

    - Create or delete a new AOP proxy class.
    - Generate or change the code defining the methods of a class.
    - Create a new property or method.

    This application/interface uses MOP and at the same time hide their complexity. In JAVA, this interface is called Java Virtual Machine Aspect Interface(JVMAI). This allows aspect and application to come to life independently. Aspect instances can be created externally to the targeted application and then be woven at an arbitrary point in time. One of the tool that uses run-time approach is Spring AOP.

4 Evalution Approaches

This section of the study evaluates the AOP approach and compare it to conventional programming approaches. The outcomes based on various criterions are as follows :

- **Code Size -** According to the research finding in this matter, there was a notable reduction in code size by approximately 40% reduction in the line of code (LoC) as well. In addition, there was a reduction in certain types of codes such as exception handling. This led to the conclusion that AOP is actually effective in minimising the code size positively most of the time. If not, it will be more or less the same as non-AOP approaches.

- **Modularity -** Modularity results were positive, by gathering the code that deals with the same aspect in one module avoiding the redundancy of crosscutting concerns.

- **Evolvability -** Evolvability is defined as the ability to adapt to continous changes in the user reuirements and operational environment. Results were positive in this criteria.

- **Language Mechanism -** The way in which AOP deals with the code is certainly different from the other conventional approaches. Exception handling was taken as an example and compered in both OOP and AOP approaches. Results proves that AOP deals better than both OOP and AOP.

5 AOP Advantages

AOP provided a better approach for handling Crosscutting Concerns. Using AOP for implementing software systems will certainly enhance software quality in many ways. Some of the main advantages are:

- **Implementation -** AOP provides a fine mechanism programmers can use to write code representing cross-cutting concerns once and have it appear wherever needed. Programmers can write references to aspects at join points, the appropriate places in code where the aspects belong. The references then call the required aspects. The

AOP compiler reads the reference and weaves the aspects into the application where they belong.

- **Improves Modularity -** Suppose a software is coded using other programming methodologies and if there are many cases of code that need to be changed throughout a program, it is always going to be difficult to find and change each case in a large program. AOP lets programmers change an aspect once and have affect wherever it occurs in an application. Thus AOP helps to improve modularity of an application.

- **Facilitates Reusability -** AOP also facilitates reusability, as it is modular. That is, programmers can use aspect modules wherever necessary in an application without rewriting code.
  Aspects thus eliminate many lines of scattered code that programmers would otherwise have to spend considerable time in writing, tracking, maintaining, and changing. This makes changing and upgrading applications more accurate.

- **Improved skill transfer** The concepts of AOP are reusable and transferable. Therefore, developers training time and cost will be minimised even if they need to learn more than one language. This is because core concerns and design patterns are universal. However, this is not the situation in other frameworks, where developers have to learn from the beginning each time, wasting considerable time and money on training.

6 Challenges

Some of the possible reason related to it still being in unpopular are highlighted in the following section:

- **Understanding Code Flow -** In AOP, understanding the code flow is very tough because there is no referencing of advices, where they will be called inside the core code. It's all defined inside the aspect module.

- **Lack of expertise -** The community members of AOP are approximately only 2000 programmers worldwide, and only 10-15% use AOP in an OOP environment.

- **Concerns -** AOP came to provide a better sepperation between the core concerns and the cross-cutting concerns. But in reality when a system reaches a certain degree of complexity, such separation is very hard to achieve.

- **Standardisation -** AOP introduced new dimensions and standards to programming. This, in general, creates complexity and possible resistance, but it was also the case when the OOP was introduced after the FOP, which indicates that this is a normal scenario.

7 Use cases of AOP

- A website in which every page is generated by a Python function. We'd like to take a class and make all of the web pages generated by that class password-protected. AOP comes to the rescue, before each function is called, We do the appropriate session checking and redirect if necessary.

- We'd like to do some logging and profiling on a bunch of functions in a program during its actual usage. AOP lets us calculate timing and print data to log files without actually modifying any of these functions.

- If we have a module or class full of non-thread-safe functions and I find myself using it in some multi-threaded code. Some AOP adds locking around these function calls without having to go into the library and change anything.

8 Conclusion

To sum things up, AOP has the potential to change the way in which applications are developed, but it also introduces developers to a new variety of issues surrounding application behavior. Also, it should be clear now that AOP is not a successor to OOP, but a new way of looking at object functionality: allowing the behavior of an object to be modularized and used across other components without creating complex class hierarchies, but instead creating complex webs of AOP configuration. AOP is nothing new, but is now facing the heat and the attention of developers everywhere.

References

[1] Heba A. Kurdi. *Review on Aspect Oriented Programming.* (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 4, No. 9, 2013.

[2] Andrei Popovici, Thomas Gross and Gustavo Alonso. *Dynamic weaving for aspect-oriented programming.*

[3] Md. Asraful Haque. *Problems in Aspect Oriented Design: Facts and Thoughts*

[4] *https://www.journaldev.com/2583/spring-aop-example-tutorial-aspect-advice-pointcut-joinpoint-annotations*

[5] *https://plg.uwaterloo.ca/ migod/846/papers/aop-intro.pdf*