A Comparative Study of AOP Approaches: AspectJ, Spring AOP, JBoss AOP

Ravi Kumar, Dalip and Munishwar Rai

ABSTRACT - Aspect-oriented programming is implemented with standard crosscutting concerns much like object-oriented programming is implemented with standard common concerns. Modularization is the important software quality principle which was proposed by Aspect Oriented Programming. The AOP could be seen as a complementary technique that exists with different approaches likes AspectJ, JBoss AOP and Spring AOP. This paper we discussed the three AOP approaches on flexibility as well as on how easily they will fit with our application.

KEYWORDS - Aspect Oriented Programming, AspectJ, JBoss AOP, Spring AOP

I. INTRODUCTION

Now a day, the Aspect Oriented Programming pattern penetrates in several areas of software development. With its increasing marketability, developers are starting to amazement whether they should start looking into it. Aspect Oriented Programming is simply an auxiliary way of designing and précis software [3]. It's normally used in merging with object orientation to let you précis a problem's aspects as well as its design.

According to Markus Völter "**Aspect Orientation is primarily a mindset.** Aspects in Aspect Oriented Programming (AOP) package advice and point cuts into functional units in much the similar way that Object Oriented Programming (OOP) uses classes into methods and package fields. AOP is not supposed to replace the wide-spread Object-Oriented programming methodology but extends it. In OOP, a 'class' is the physical representation of a 'dominant concern' [13]. In AOP, an 'aspect' is the physical representation of an 'aspect' [5] [14] [20]. Aspects are also highly modular, making it possible to develop plug-and-play implementations of crosscutting functionality [18].

Manuscript submission for review on dated July 01, 2019.

Ravi Kumar is a PhD student in the department of Maharishi Markandeshwar Institute of Computer Technology & Business Management (MMICT&BM) at the university of Maharishi Markandeshwar Deemed to be University, Mullana (Ambala)-133207, Haryana, India.
E-mail id: ravisangwan77@gmail.com

Dr. Dalip is an Assistant Professor in the department of Maharishi Markandeshwar Institute of Computer Technology & Business Management (MMICT&BM) at the University of Maharishi Markandeshwar Deemed to be University, Mullana (Ambala)-133207, Haryana, India.
E-mail id: dalipkamboj@mmumullana.org

Dr. Munishwar Rai is a Professor and Head of Computer Science Department at the Sri-Sri University Bidyadharpur, Cuttack- 754006, and Orissa, India.
E-mail id: muniswar.r@srisriuniversity.edu.in

Before we begin, let's do a quick, high-level review of terms and core concepts [4]:

Aspect – A standard code/feature that is scattered across multiple places in the application and is typically different than the actual Business Logic (for example, Transaction management). Each aspect focuses on a specific cross-cutting functionality.

Join point – It's a particular point during execution of programs like method execution, constructor call, or field assignment.

Advice – The action taken by the aspect in a specific join point.

Point cut – a regular expression that matches a join point. Each time any join point matches a point cut, a specified advice associated with that point cut is executed.

Weaving – the process of linking aspects with targeted objects to create an advised object.

An aspect is an entity that looks like a class but models a concern that crosscuts object classes. Point cuts are declarations used in an aspect to identify principled points in the program execution and source code locations where it can be involved. Principled points such as an access or change of a field value, a method call or a method execution are called Join points. Point cuts are particular forms of predicates that use Boolean operators and specific primitives to capture join points and dynamic contextual information such as parameters of a call statement. The aspect code is divided into blocks called advices. They are method-like mechanisms used to declare that a certain code should execute before, after or around the code corresponding to the join points captured by point cuts. Therefore, there are three possible relationships that bind an advice to point cuts: before, after and around [23].

II. PROPOSED WORK

Today the various AOP approaches available in software development and developers face a number of problems and some questions arise in his mind like as:

Q1. Which approach is best appropriate with my existing or new application?

Q2. Which AOP approach is most suitable for implementation?

Q3. How fast will it merge with my application?

Q4. What is the performance elevated?

In this paper, we will introduce JBoss AOP, Spring AOP and AspectJ – the three most popular AOP interfaces with some key areas and find the answering these questions.

ISBN: 978-988-14048-7-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

III. OUTLINE OF SELECTED AOP APPROACHES

AspectJ - AspectJ is a general purpose programming language, which is simple and a practical Aspect-oriented extension to Java. AspectJ extends the Java language with keywords for writing aspects, point cuts, advice code, and intertype declarations. Gregor Kiczales and his team, who has created a new programming paradigm AOP at the Palo Alto Research Center (PARC) [2], has also developed AspectJ—which is now the leading tool for Aspect-oriented Programming. Using this, it is possible to create clean modular implementations of crosscutting concerns such as tracing, login, user session management, synchronization, consistency checking, protocol management etc [7] [8].

AspectJ supports eleven different kinds of join points: method call, method execution, constructor call, constructor execution, field get, field set, pre-initialization, initialization, static initialization, handler, and advice execution join points. There are also nine kinds of point cut designators that match join points according to their kind: call, execution, get, set, handler, static initialization, pre initialization, initialization, and advice execution [23].

By creating, just a few new constructs, AspectJ provides support for modular implementation of a range of crosscutting concerns [1]. Here in dynamic join point model, join points are well-defined points of the program where the advice code will be executed; point cuts are collections of join points; advice are special method-like constructs that can be attached to point cuts; and aspects are modular units of crosscutting implementation, comprising point cuts, advice, and ordinary Java member declarations [15]. AspectJ is a static-typing programming language and also considered as type-safe like its base class Java. However, researches [11] have revealed that, unlike Java, AspectJ does not have a safe type system, a binding between a point cut and an advice can rise to type errors at runtime. Also, AspectJ‘s typing rules severely restrict the definition of certain generic advice behavior. In AspectJ, a cross-cutting concern i.e., memory monitoring and management can be applied at a point cut of the program for better memory management. This approach can be used for both managed and unmanaged resources (files, handles, DB connections etc.). AspectJ provides a rich set of primitive point cuts to specify join points within an aspect [16].

The last concept of AspectJ is the static crosscutting which modifies a program at compile time by specifying new members of a class (called introduction) or specifying what a class extends or implements (called inter-type member declaration) [22].

JBoss AOP - JBoss AOP was designed and developed by Bill Burke [17]. JBoss AOP is a pure Java Aspect Oriented Framework usable in any programming environment or tightly integrated with our application server. Aspects allow you to more easily modularize your code base when regular object oriented programming just does not fit the bill. It can provide a cleaner separation from application logic and system code. It provides a great way to expose integration points into your software. Combined with Java Annotations, it also is a great way to expand the Java language in a clean pluggable way rather than using annotations solely for code generation. It can be used independently or in conjunction with J2EE application server JBoss; in the first case it is called Standalone. From version 4.0, the JBoss Application Server includes as standard the JBoss AOP framework. JBoss AOP is not only a framework, but also a prepackaged set of aspects that are applied via annotations, point cut expressions, or dynamically at runtime. Some of these include caching, asynchronous communication, transactions, security, remoting, and many more. JBoss AOP works with plain old Java objects (POJOs) as opposed to pre-defined "components". JBoss AOP allows you to apply Enterprise Java Bean (EJB)-style services to POJOs without the complex EJB infrastructure code and deployment descriptors. You can develop new aspects and deploy them into the application server for all applications to use. That essentially extends the existing container services. JBoss AOP can also be used in standalone Java applications. A detailed introduction to aspect-oriented programming and the JBoss AOP framework can be found on JBoss web site.

If AspectJ defines point cuts using keywords, the point cuts declarations in JBoss AOP can be done in two ways: in a dedicated XML document (usually called jboss-aop.xml.), or in the class that implements the aspect, as annotation. JBoss AOP allows defining five types of point cuts: method execution, constructor, attribute, class and method call. An invocation is a JBoss AOP class that encapsulates what a join point is at runtime. It could contain information like which method is being called, the arguments of the method, etc [21].

In JBoss AOP, the aspect is a java class. The advices are methods (i.e. code that must be executed). An interceptor in JBoss AOP is a particular type of aspect that has only one advice. The mix-in mechanism provided by JBoss AOP allows extending the behavior of an application. This mechanism is similar to the introduction mechanism applied in AspectJ. Specifically with the mix-in mechanism we can introduce interfaces, fields and methods to the existing classes of an application [12].

Spring AOP - Aspect-Oriented Programming (AOP) complements Object-Oriented Programming (OOP) by providing another way of thinking about program structure [19]. The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect. Aspects enable the modularization of concerns such as transaction management that cut across multiple types and objects. (Such concerns are often termed crosscutting concerns in AOP literature.) [10]

One of the key components of Spring is the AOP framework. While the Spring Inversion of Control (IoC) container does not depend on AOP, meaning you do not need to use AOP if you don't want to, AOP complements Spring IoC to provide a very capable middleware solution. AOP is used in the Spring Framework to provide declarative enterprise services, especially as a replacement for EJB declarative services. The most important such service is declarative transaction management. AOP is used in the Spring Framework to allow users to implement custom aspects, complementing their use of OOP with AOP [6].

ISBN: 978-988-14048-7-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

In Spring AOP, aspects are implemented using regular classes (the schema-based approach) or regular classes annotated with the @Aspect annotation (the @AspectJ style). In Spring AOP, a join point always represents a method execution. Many AOP frameworks, including Spring, model an advice as an interceptor, maintaining a chain of interceptors around the join point. The concept of join points as matched by point cut expressions is central to AOP, and Spring uses the AspectJ point cut expression language by default. Spring AOP allows you to introduce new interfaces (and a corresponding implementation) to any advised object. Since Spring AOP is implemented using runtime proxies, this object will always be a proxy object [15]. In the Spring Framework, an AOP proxy will be a JDK dynamic proxy or a CGLIB proxy. Spring AOP, like other pure Java AOP frameworks, performs weaving at runtime. Since Spring AOP, like AspectJ, provides a full range of advice types, we recommend that you use the least powerful advice type that can implement the required behavior. For example, if you need only to update a cache with the return value of a method, you are better off implementing an after returning advice than an around advice, although an around advice can accomplish the same thing. Using the most specific advice type provides a simpler programming model with less potential for errors. For example, you do not need to invoke the proceed () method on the Join point used for around advice, and hence cannot fail to invoke it.

In Spring 2.0, all advice parameters are statically typed, so that you work with advice parameters of the appropriate type (the type of the return value from a method execution for example) rather than Object arrays [9].

Spring AOP is implemented in pure Java. There is no need for a special compilation process. Spring AOP does not need to control the class loader hierarchy, and is thus suitable for use in a Servlet container or application server. Spring AOP currently supports only method execution join points (advising the execution of methods on Spring beans). Field interception is not implemented, although support for field interception could be added without breaking the core Spring AOP APIs. If you need to advise field access and update join points, consider a language such as AspectJ. Spring AOP's approach to AOP differs from that of most other AOP frameworks. The aim is not to provide the most complete AOP implementation (although Spring AOP is quite capable); it is rather to provide a close integration between AOP implementation and Spring IoC to help solve common problems in enterprise applications [24].

Thus, for example, the Spring Framework's AOP functionality is normally used in conjunction with the Spring IoC container. Aspects are configured using normal bean definition syntax (although this allows powerful "auto proxy" capabilities). This is a crucial difference from other AOP implementations. There are some things you cannot do easily or efficiently with Spring AOP, such as advice very fine-grained objects (such as domain objects typically): AspectJ is the best choice in such cases. However, our experience is that Spring AOP provides an excellent solution to most problems in enterprise Java applications that are amenable to AOP. Spring AOP will never strive to compete with AspectJ to provide a comprehensive AOP solution. We believe that both proxy-based frameworks like Spring AOP and full-blown frameworks such as AspectJ are valuable, and that they are complementary, rather than in competition [23].

IV. ANALYSIS OF AOP APPROACHES WITH DIFFERENT PARAMETERS

Now, we discuss Spring AOP and AspectJ across a different parameter like competency, objectives, weaving, internal framework, join points, simplicity and performance.

A. Competency and objectives

JBoss AOP, Spring AOP and AspectJ have different objectives. JBoss AOP is the ability to introduce an interface to an existing Java class in a transparent way. You can force a class to implement an interface and even specify an additional class called a mix-in that implements that interface. Spring AOP aims to provide a simple AOP implementation across Spring IoC to solve the most common problems that programmers face. It is not intended as a complete AOP solution – it can only be applied to beans that are managed by a Spring container.

On the other hand, AspectJ is the original AOP technology which aims to provide complete AOP solution. It is more robust but also significantly more complicated than Spring AOP. It's also worth noting that AspectJ can be applied across all domain objects. JBoss AOP can also be used in standalone Java applications.

B. Weaving

JBoss AOP, AspectJ and Spring AOP use the different type of weaving which affects their behavior regarding performance and ease of use [2]. AspectJ makes use of three different types of weaving:

a. **Compile-time weaving:** The AspectJ compiler takes as input both the source code of our aspect and our application and produces a woven class files as output

b. **Post-compile weaving:** This is also known as binary weaving. It is used to weave existing class files and JAR files with our aspects

c. **Load-time weaving:** This is exactly like the former binary weaving, with a difference that weaving is postponed until a class loader loads the class files to the JVM.

JBoss AOP used three different modes to run your aspectized applications such as Precompiled, load time or hot swap. JBoss AOP needs to weave your aspects into the classes which they aspectize. You can choose to use JBoss AOP's pre compiler to accomplish this (Compile time) or have this weaving happen at runtime either when the class is loaded (Load time) or after it (Hot Swap).

Compile-time happens before you run your application. Compile time weaving is done by using the JBoss AOP pre compiler to weave in your aspects to existing .class files. The way it works is that you run the JBoss AOP pre compiler on a set of .class files and those files will be modified based on what aspects you have defined. Compile time weaving isn't always the best choice though. JSPs are a good instance where compile time weaving may not be

ISBN: 978-988-14048-7-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

feasible. It is also perfectly reasonable to mix and matches compile time and load time though. If you have load-time transformation enabled, precompiled aspects are not transformed when they are loaded and ignored by the class loader transformer.

Load-time weaving offers the ultimate flexibility. JBoss AOP does not require a special class loader to do load time weaving, but there are some issues that you need to think about. The Java Virtual Machine actually has a simple standard mechanism of hooking in a class transformer through the –java agent. JBoss AOP an additional load-time transformer that can hook into class loading via this standard mechanism. Load-time weaving also has other serious side effects that you need to be aware of. JBoss AOP needs to do the same kinds of things that any standard Java profiling product needs to do. It needs to be able to process byte code at runtime. This means that boot can end up being significantly slowed down because JBoss AOP has to do a lot of work before a class can be loaded. Once all classes are loaded though, load-time weaving has zero effect on the speed of your application. Besides boot time, load-time weaving has to create a lot of Java data structure that represent the byte code of a particular class. These data structures consume a lot of memory. JBoss AOP does its best to flush and garbage collects these data structures, but some must be kept in memory.

Hot Swap weaving is a good choice if you need to enable aspects in runtime and don't want that the flow control of your classes be changed before that. When using this mode, your classes are instrumented a minimum necessary before getting loaded, without affecting the flow control. If any join point becomes intercepted in runtime due to a dynamic AOP operation, the affected classes are weaved, so that the added interceptors and aspects can be invoked. As the previous mode, hot swap contains some drawbacks that need to be considered.

As AspectJ uses compile time and class load time weaving, Spring AOP makes use of runtime weaving.

C. Internal Framework and Application

Spring AOP is a proxy-based AOP framework. This means that to implement aspects to the target objects, it'll create proxies of that object as shown in Fig 1. This is achieved using either of two ways:

JDK dynamic proxy –It is the preferred way for Spring AOP. Whenever the targeted object implements even one interface, then JDK dynamic proxy will be used

CGLIB proxy – If the target object doesn't implement an interface, then CGLIB i.e. Code Generation Library proxy can be used.

We can learn more about Spring AOP proxy mechanisms from the official docs.

AspectJ, on the other hand, doesn't do anything at runtime as the classes are compiled directly with aspects.

And so unlike Spring AOP, it doesn't require any design patterns [23]. To weave the aspects to the code, it introduces its compiler known as AspectJ compiler (ajc), through which we compile our program and then runs it by supplying a small runtime library. An interceptor in JBoss AOP is a particular type of aspect that has only one advice. The mix-in mechanism provided by JBoss AOP allows extending the behavior of an application as shown in Fig 2.

D. Join points

We showed that Spring AOP is based on proxy patterns. Because of this, it needs to subclass the targeted Java class and applies cross-cutting concerns accordingly. But it comes with a limitation. We cannot apply cross-cutting concerns (or aspects) across classes that are "final" because they cannot be overridden and thus it would result in a runtime exception.

The same applies for static and final methods. Spring aspects cannot be applied to them because they cannot be overridden. Hence Spring AOP because of these limitations only supports method execution join points. However, AspectJ weaves the cross-cutting concerns directly into the actual code before runtime. Unlike Spring AOP, it doesn't require to subclass the targeted object and thus supports many others join points as well. That's obviously because when we call a method within the same class, then we aren't calling the method of the proxy that Spring AOP supplies. If we need this functionality, then we do have to define a separate method in different beans, or use AspectJ. Table I show the summary of compatible join points:-

Table I
Summary of Compatible Join points

Join points	Spring AOP Compatibility	AspectJ Compatibility	JBoss AOP Compatibility
Calling Method	×	✓	✓
Execution of Method	✓	✓	✓
Calling Constructor	×	✓	✓
Execution of Constructor	×	✓	×
Execution of Static initialization	×	✓	✓
Initialization of Object	×	✓	✓
Field reference	×	✓	✓
Field assignment	×	✓	✓
Execution of Handler	×	✓	✓
Execution of Advice	×	✓	✓

ISBN: 978-988-14048-7-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

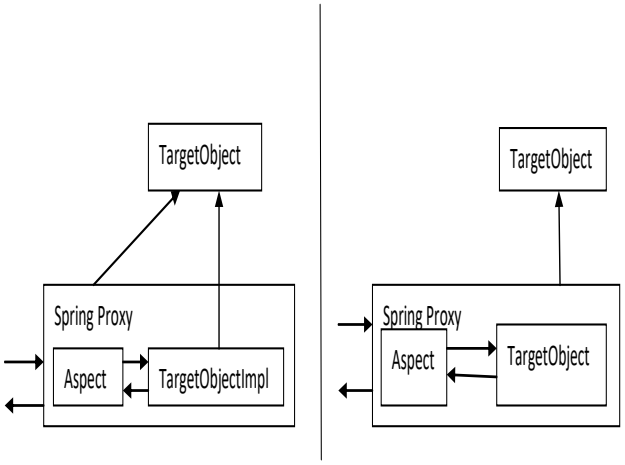

Fig 1 Spring AOP Process

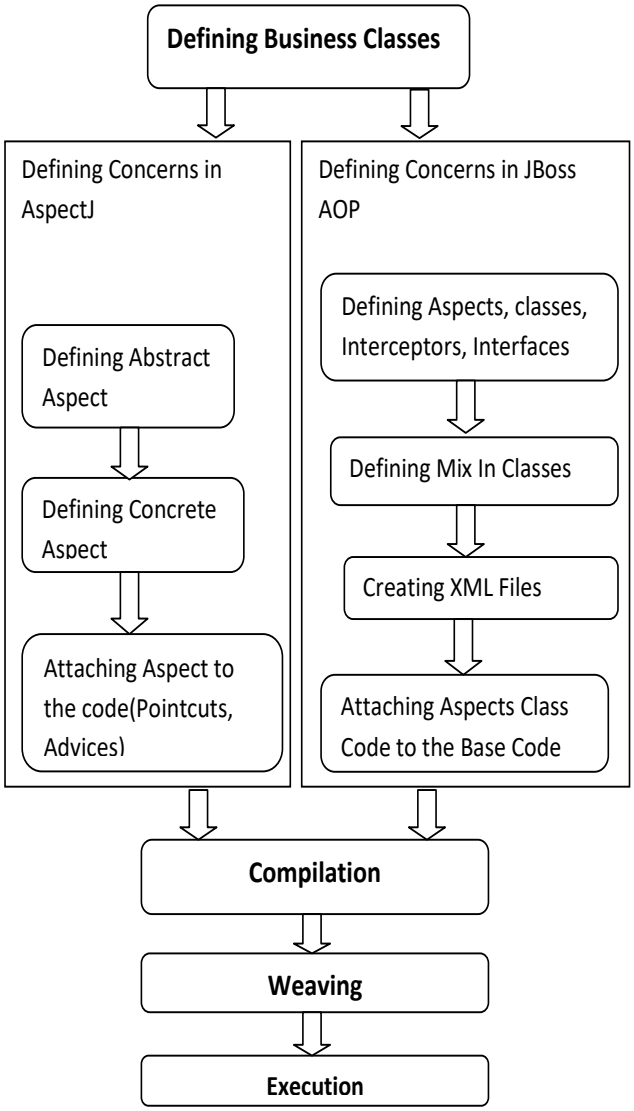

Fig 2 JBoss AOP and AspectJ programming process

E. Simplicity

Spring AOP is obviously simpler because it doesn't introduce any extra compiler or weaver our build process. It uses runtime weaving, and therefore it integrates seamlessly with our usual build process. Although it looks simple, it only works with beans that are managed by Spring AOP.

However, to use AspectJ and JBoss AOP, we're required to introduce the AspectJ compiler (ajc) and re-package all our libraries (unless we switch to post-compile or load-time weaving).

This is, of course, more complicated than the former – because it introduces AspectJ Java Tools (which include a compiler (ajc), a debugger (ajdb) and documentation generator (ajdoc), a program structure browser (ajbrowser)) which we need to integrate with either our IDE or the build tool.

F. Performance

As far as performance is concerned, compile-time weaving is much faster than runtime weaving. Spring AOP is a proxy-based framework, so there is the creation of proxies at the time of application startup. Also, there are a few more method invocations per aspect, which affects the performance negatively.

On the other hand, AspectJ weaves the aspects into the main code before the application executes and thus there's no additional runtime overhead, unlike Spring AOP and JBoss AOP. For these reasons, the benchmarks suggest that AspectJ is much faster than Spring AOP and JBoss AOP.

V. SUMMARY

We summarize the key differences between Spring AOP and AspectJ as shown in Table II. JBoss AOP and Spring AOP Implemented in pure Java but AspectJ programming language Implemented using extensions of Java. Spring AOP, no need for separate compilation process but JBoss AOP and AspectJ needs AspectJ compiler (ajc) unless Load Time Weaving is set up. Only runtime weaving is available in Spring AOP but in JBoss AOP and AspectJ supports compile-time, post-compile, and load-time Weaving. Spring AOP only supports method level weaving but JBoss AOP and AspectJ can weave fields, methods, constructors, static initialize, final class/methods, etc. Spring AOP supports only method execution point cuts but JBoss AOP and AspectJ support all point cuts. If we analyze all the arguments made in this paper, we'll start to understand that it's not at all that one framework is better than another. Simply put, the choice heavily depends on our requirements:

a) **Interface -** If the application is not using Spring interface, then we have no option but to drop the idea of using Spring AOP because it cannot manage anything that's outside the reach of spring container. However, if our application is created entirely using Spring interface, then we can use Spring AOP as it is straight forward to learn and implement.

b) **Flexibility -** Given the limited join point support, JBoss AOP, Spring AOP is not a complete AOP solution,

ISBN: 978-988-14048-7-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

but it solves the most common problems that programmers face. Although if we want to dig deeper and exploit AOP to its maximum capability and want the support from a wide range of available join points, then AspectJ is the choice.

c) **Output -** If we're using slight aspects, then there are trivial output differences. But there are sometimes cases when an application has more than tens of thousands of aspects. We would not want to use runtime weaving in such cases so it would be better to opt for AspectJ. AspectJ is known much faster than JBoss AOP and Spring AOP.

d) **Best between three techniques -** All of these techniques are totally appropriate with each other. We can regular take lead of JBoss AOP, Spring AOP whenever possible and still use AspectJ to get support of join points that are not supported by other approaches.

Table II
Difference between Spring AOP and AspectJ

Spring AOP	AspectJ
Pure Java implementation	Using extensions of Java implementation
Compilation process is not separated	AspectJ compiler (ajc) needed unless Load Time Weaving is set up
Available runtime weaving	Not available runtime weaving. Compile-time, post-compile, and load-time Weaving supported
Method level weaving supported so less robust	Use final class/methods, weave fields, constructors, static initialization, methods, etc. so most robust
Implemented on beans managed by Spring container	Implemented on all domain objects
Method execution point cuts supported	All point cuts supported
Proxies are created of targeted objects, and aspects are applied on these proxies	Aspects are weaved directly into code before application is executed (before runtime)
Much slower than AspectJ	Performance is better than Spring AOP
Easy to learn and implement	More complicated than Spring AOP

VI. CONCLUSION

AOP is a programming technique that target to resolve crosscutting concerns by providing better modularization of the code. This paper delivers a transitory outline of the JBoss AOP, Spring AOP and AspectJ approaches. We compared the three AOP approaches on flexibility as well as on how easily they will fit with our applications. We analyzed JBoss AOP, Spring AOP and AspectJ in different parameters.

REFERENCES

[1] Geeta Bagade and Shashank Joshi, "Exploring AspectJ Refactoring", International Journal of Computer Applications, 2016.
[2] Ramniwas Laddad, "AspectJ in Action: Practical Aspect Oriented Programming", Manning Publications, 2003.
[3] Zambrano Polo y La Borda, Arturo Federico (5 June 2013). "Addressing aspect interactions in an industrial setting: experiences, problems and solutions": 159. Retrieved 30 May, 2014.
[4] Sk. Riazur Raheman, Amiya Kumar Rath, Hima Bindu M, "Dynamic Slice of Aspect Oriented Program: A Comparative Study" IJRITCC, vol.2, Issue: 2, pp.249-259, 2014
[5] Abhishek Ray et. al., "An Approach for Computing Dynamic Slice of Concurrent Aspect-Oriented Programs", International Journal of Software Engineering and Its Applications, Vol. 7, No. 1, January, 2013.
[6] Aspect Oriented programming with Spring; Spring Framework: "http://www.springframework.org/docs/reference/aop. html"
[7] Adam Przylylek, "Impact of Aspect Oriented Programming on Software Modularity", https://www.researchgate.net/publication/224227238/2011
[8] S.Kotrappa and P. J. Kulkarni, "Multilevel Security using Aspect Oriented Programming AspectJ", https://www.researchgate.net/publication/224202556/2010
[9] Erik Gollot. "Introduction au framework Spring. [online]. http://ego.developpez.com/spring/ (accessed June 9, 2006)".
[10] Jyri Laukkanen, "Aspect-Oriented Programming, ACM Computing Classification System (CCS)", 2008
[11] Colyer and Clement, "Aspect-oriented programming with AspectJ," IBM Systems Journal, Vol 44, NO 2, 2005.
[12] "labs.jboss.com. Jobs AOP Reference Documentation". JBossInc./RedhatInc.,2003. "http://labs.jboss.com/portal/jbossaop/docs/1.5.0.GA/docs/aspect-framewo%rk/reference/en/html/index. html".
[13] Parul Rajpal and Prof. Amanpreet Kaur, "Comparative Study of Component – Oriented and Aspect – Oriented Programming", 2015 www.ijarcsse.com
[14] Geetanjali Sharma, "Twist of Aspect Oriented and Component Oriented, International Journal of Computer Science and Communication", Vol. 3, No. 1, January-June 2012.
[15] R. Laddad, "AspectJ in Action", Enterprise AOP with Spring, Manning Publications, 2010
[16] A. Colyer, A. Clement, G. Harley and M. Webster, "Eclipse AspectJ: Aspect-Oriented Programming with AspectJ and Eclipse AspectJ Development Tools", Addison-Wesley Professional, 2004.
[17] Soumeya Debboub and Djamel Meslati, "Quantitative and qualitative evaluation of AspectJ, JBoss AOP and CaesarJ, using Gang-of-Four design patterns", International Journal of Software Engineering and Its Applications Vol.7, No.6 (2013)
[18] Munishwar Rai, Rajender Nath & Jai Bhagwan, "Validation of Cluster Based Reusability Model" International Journal Computer Technology and Applications (IJCTA) Volume 5, Issue 3, June 2014, PP: 829-32
[19] Gregor Kiczales and Mira Mezini "Aspect-Oriented Programming and Modular Reasoning", ACM, ICSE'05, May 15–21, 2005
[20] Gregory Kiczales et. al., "An Overview of AspectJ", published in proceedings of the 15th European Conference on Object Oriented Programming, pages 327-353, 2001.
[21] J.Boss AOP homepage, "http://www.jboss.org/jbossaop/".
[22] Daniela Gotseva and Mario Pavlov, "Aspect-oriented programming with AspectJ" IJCSI vol. 9, Issue 5, No.1. 2012
[23] Jörg Kienzle, "11th International Workshop on Aspect-Oriented Modeling", 2008
[24] www.springframework.org. Spring Framework v 2.0 M5. [online]. "http://www.springframework.org/ "(accessed June 5, 2006)