Aspect Oriented Software Development: Concepts, Issues and Real-World Applications

[1]Priyanka. R. Sarode, [2]Dr. R. N. Jugele
[1]Inter Institutional of Computer Centre, R. T. M. Nagpur University, Maharashtra, India
[2]Shri Shivaji Science College, Nagpur, Maharashtra, India

Abstract
Aspect-oriented software development supports better separation of concerns by presenting a new modular unit, called an aspect, for modularization of crosscutting concerns. As a new type of modular unit, aspect should have clear edges that define the way they communicate with the rest of the system and how they affect other elements. Several programming languages and mechanisms proposed for implementing aspect-oriented systems, and these systems are beginning to use for real-world applications. Based on the collected data this paper gives the conceptual discussion on the key concepts, pitfalls of Aspect Oriented Software Development and discuss the major industrial projects using Aspect Oriented Software Development.

Keywords
Aspect Oriented Software Development, Crosscutting concerns, Modularity, Aspect.

I. Introduction

There is a growing level of irregularity of programming frameworks and the sorts of concerns they address, forcing new difficulties to the standard programming designing standards. The object-oriented paradigm is not adequate to modularize some normal concerns found in most complex frameworks. They have been called crosscutting concerns since they normally cut over the boundaries of different concerns [6-7]. Aspect Oriented Software Development (AOSD) [8] is a developing methodology with the objective of enhancing the separation of crosscutting concerns all through the software development lifecycle. AOSD reflects about standard commitments to separation of concerns and modularity presented by past innovations, while presenting another modular unit, called aspect, for the modularization of crosscutting concerns. The estimated advantages of AOSD are moved forward intelligibility, simplicity of development and expanded potential for reuse in the improvement of complex programming frameworks. A growing area of research in the field of software development is concentrated on bringing aspect-oriented techniques into the scope of analysis and design [8-9]. Aspects may seem in any phase of the software development lifecycle (e.g. requirements, specification, design and execution). Crosscutting concerns can run from high level of security to low-level thoughts like storing and from functional requirements, for example, business standards to non-useful requirements like transactions.

II. AOSD Concepts

Following are the basic concepts associated with modularity and AOSD.

A. Separation of Concerns

To understand the conceptof AOSD, the principle ofseparation of concerns can be really viewed as one ofthe key principles in software engineering. This principle states that a givenproblem involves different types of concerns, which are identified andseparated to adapt with complexity and to accomplish the required engineering qualityfactors such as robustness, adaptability, maintainability and reusability. Modularity is also a fundamental principle for managing software complexity [10-11]. Complex software systems decomposed into a set of highly cohesive modules, each implementing well-defined interfaces and dealing with a single concern. An interface is a well-defined prescription of how the module, which realizes it, interacts with the rest of the system [10, 12].

B. Concerns

The basic modules used in object-oriented software development (OOSD) are classes and objects. However, the modules and the composition mechanisms provided by OOSD may not be sufficient for separating some concerns found in most complex systems. These concerns have been calledcrosscutting concerns since they naturally cut across the modularity of other concerns. Aspect-oriented software development (AOSD) [1, 8] has been proposed as a technique for improving the separation of crosscutting concerns. AOSD addresses the modularization of these concerns by providing a new abstraction, called aspect, which makes it possible to separate and compose them to produce the overall system. Thus, an aspect-oriented (AO) system is composed of two kinds of modules: classes and aspects. The predominant definition for aspects is the one that comes from the AspectJ programming language [2] aspects are implementation-level modules that specify and localize:

1. Refinements and redefinitions of behavior at well-established points localized in the other system's modules.
2. Additions of members (state elements and behaviour) to other system's modules.
3. Modifications of type relationships with existing modules. In aspect-oriented programming, the term concern is comprehensive with the so-called crosscutting properties such as synchronization, memory management and persistency. In AspectJ's terminology

- Is implemented by means of pointcuts and advice.
- Are implemented by means of inter-type declarations [1].

III. Issues of AOSD

A. Modular Decomposition

The separation of concerns principles states that each concern of a given software design problem should be mapped to one module in the system. Or else, the problem should be decomposed into modules such that each module has one concern. The benefit of this is that concerns are localized and as such can be easier understood, extended, reused, and adapted. This decomposition process is illustrated in Fig. 1(a). The design problem is decomposed into concerns (C1, C2…Cn) and each of these concerns is mapped to a separate module (M1, M2…Mn). A module is an abstraction of a modular unit in a given design language (e.g. class or function) [5].

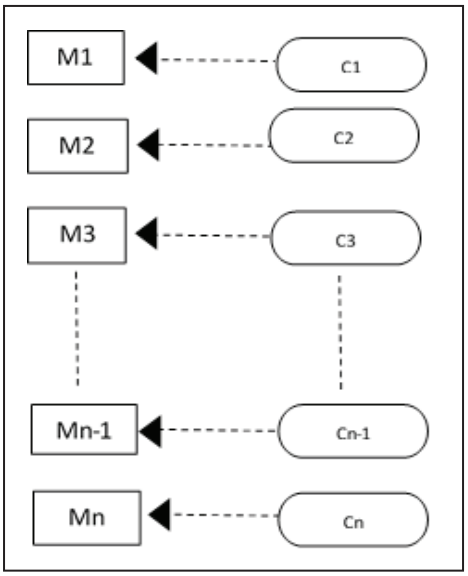

Fig. 1:(a)Mapping concerns c1,c2,..cn to modulesM1,M2,… Mn.

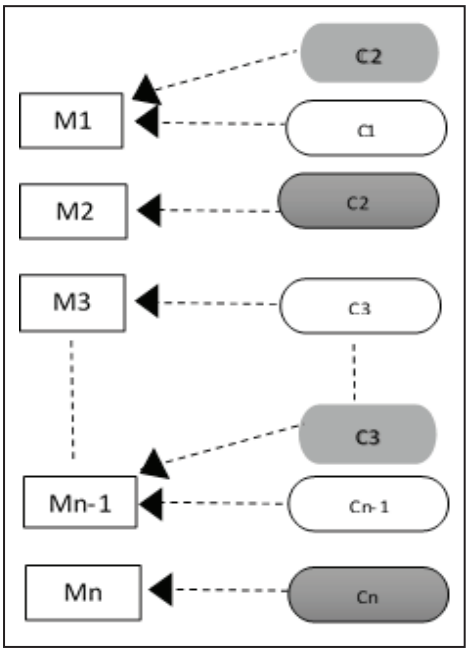

Fig. 1:(b) Concerns c2, c3 crosscut modules M1, Mn-1, Mn.

B. Crosscutting Concerns

Many concerns can indeed be mapped to single modules. Some concerns, however, cannot be easily separated, and given the design language we are forced to map such concerns over many modules. This is called crosscutting. In Fig. 1(b), for example, concern C2 is mapped to the modules M1, M2 and Mn-1. We say that C2 is a crosscutting concern or an aspect. Examples of aspects are e.g. tracing, synchronization, and load balancing. Aspects are not the result of a bad design but have more inherent reasons. A bad design including mixed concerns over the modules could be re-factored to a neat design in whicheach module only addresses a single concern. However, if we are dealing with these crosscutting concerns this is in principle not possible, that is, each refactoring attempt will fail and the crosscutting will remain. A crosscutting concern is a serious problem, since it is harder to understand, reuse, extend, adapt and maintain the concern because it is spread over many places. Finding the places where the crosscutting occurs is the first problem, adapting the concern appropriately is another problem.Things may even worsen if we have to deal with multiple crosscutting concerns. For example in Fig. 2(a) we have to deal with 2 crosscutting concerns C2 and C3.

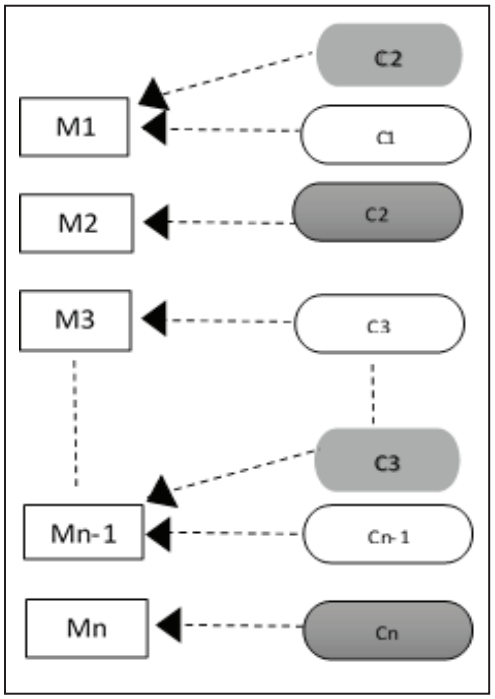

Fig. 2: (a) Crosscutting Concerns c2 and c3

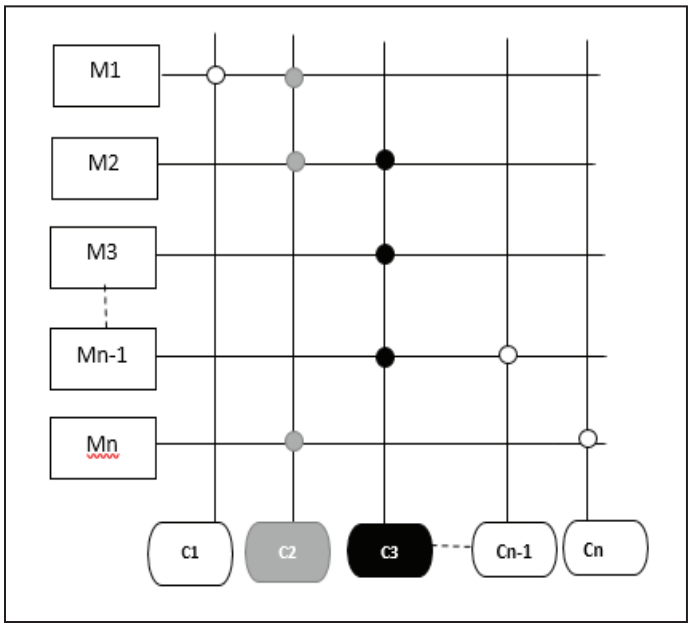

Fig. 2:(b) Join points, Tangling and Cross cutting concerns

C. Tangled Concerns

Since we cannot easily localize and separate crosscutting concerns, several modules will include more than one concern. We say that theconcerns are tangled in the corresponding module. For example in Fig. 2(a), the concerns C2, C3 and Cn-1 are tangled in the module Mn-1. Note that concern Cn-1 is not crosscutting. Join points. In Fig. 2(b) the same information is depicted as in Fig. 2(a). The modules are aligned vertically and the concerns horizontally. The circles represent the places where the concerns crosscut a module. These are called join points. Join points can be at the level of a module (class) or be more refined and deal with subparts of the module (e.g. attribute or operation). Crosscutting can be easily identified if we follow a concern in a vertical direction (multiple join points). Tangling can be detected if we follow each module in the horizontal direction.

D. Aspect Decomposition and Weaving Aspect

A given design problem can have crosscutting concerns and conventional abstraction mechanisms fail to cope appropriately with these concerns. AOSD provides explicit abstractions for representing crosscutting concerns, referred to as aspects. As such, a given design problem is decomposed into concerns that can be localized into separate modules and concerns that tend to crosscut over a set of modules.

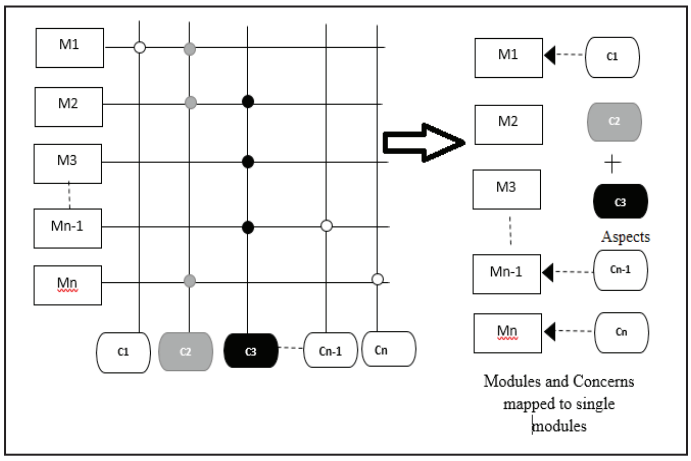

Fig. 3: Aspect Separation

1. Pointcut Specification

To specify the points that the aspect crosscuts a pointcut specification is used. A pointcut specification is, essentially, a predicate over the complete set of join points that the aspect can crosscut. A pointcut specification can enumerate the join points or provide a more abstract specification. In an abstract sense, aspects can thus be specified as follows.

- Aspect name
- Pointcut specification

2. Advice

The crosscutting is actually localized in the pointcut specification. The pointcut specification indicates which points the aspect crosscuts but it does not specify what kind of behaviour is needed. For this the concept of advice has been introduced. An advice is a behaviour that can be attached before, after, instead of or around a join point in the pointcut specification.

- Aspect name
- Pointcut specification
- Advice

3. Weaving

Having separated the aspects, their management (e.g. maintenance or reuse) become easier and consistent. In order to obtain a complete system from the separated artefacts, AO provides the weaving mechanism. Weaving is the process of composing core functionality modules with aspects, thereby yielding a working system. The various AO approaches have defined several different mechanisms for weaving. There are several aspect-oriented approaches and languages. Although they differ in the way of specifying aspects, pointcuts, advices, and weaving adopt the concepts presented above.

IV. Real World Applications of AOSD

The previous decade has seen the expanded utilization of AOSD techniques as a way to modularize crosscutting concerns in programming frameworks as discussed in Section III. In this mannerenhancing advancement organizations working practices and rate of profitability. Variousmechanical quality perspective arranged (AO) programming structures exist, including AspectJ, JBoss and Spring, as do different aspect oriented analysis and design strategies. The "Major Industrial Projects Using AOSD" sidebarhighlights notable applications of AOSD, of which the mostprominent is the IBM WebSphere Application Server.

Developers considering AOSD techniques must ask essential questions like, Does the improved modularity yield real benefits whenengineering and evolving software?Answers to such questions are not readily available, andgathering knowledge from existing works on thetopic is difficult, but [2] have obtained some insights by analyzing several medium- and large-scale projectsemploying AOSD techniques. These projects have beenaccessible to [2] both directly within AOSD-Europe (www.aosd-europe.net), a large-scale academia-industry collaborationfunded by the European Commission since 2004, aswell as indirectly through its liaison channels with interested researchers [2].

A. Major Industrial Projects Using AOSD

1. IBM WebSphere Application Server is a Java application server that supports Java Enterprise Edition (EE) and Web services. WebSphere is distributed in various editions that support differentfeatures, with AspectJ (www.eclipse.org/aspectj) used toisolate these features.
2. JBoss Application Server (AS) is a free, open source Java application server that supports Java EE. The core of JBoss AS uses JBoss AOP (www.jboss.org/jbossaop) to deploy services such assecurity and transaction management.
3. Oracle TopLink is a Java object-to-relational persistence framework that is integrated with the Spring application server (www.springsource.org). TopLink achieves high levels of persistence transparency using Spring AOP.
4. Sun Micro systems uses AspectJ to streamline mobile application development for the Java Micro Edition (ME) platform. Aspects are used to simplify the development of mobile applications for deployment to different operator decks and different mobile gaming community interfaces.
5. Siemens' Soarian is a health information system (HIS) that supportsseam less access to patient medical records and the definition of workflows for health provider organizations. Soarian uses AspectJ and fast AOP (http://sourceforge.net/projects/fastaop) to integrate cross cutting features into an agile developmentprocess.
6. Motorola's wi4 is a cellular infrastructure system that provides support for the WiMax wireless broadband standard. The wi4 control software is developed using WEAVR (an aspect-oriented extension to the UML 2.0 standard) for debugging and testing.
7. ASML, a provider of lithography systems for the semiconductor industry, uses Mirjam, an aspect-oriented extension to C, to modularize tracing, profiling and error-handling concerns.
8. Glassbox is a trouble shooting agent for Java applications that automatically diagnoses common problems. Glass box Inspector uses AspectJ to monitor the Java virtual machine's activity.
9. MySQL is a widely used relational database management system. The logging feature in My SQL is implemented using Aspect J.

All the software systems using AOSD that listed in [2] are medium- to large-scale and span a widerange of domains including enterprise systems, e-health, e-transport, telecommunications,

Web based information systems, multimedia applications and workflow systems.

V. Conclusion and Future Scope

In this paper, an overview of Aspect Oriented Software Development is given in Section II. Section III is all about the discussion of problem statements, which are tackled by AOSD. Finally, in Section IV listed out the major industrial applications, which uses AOSD. Still developers having some fundamental questions like how is AOSD being used in industrial projects today? What do developers need to be aware of when usingAOSD techniques? These questions are still unanswered or gathering the information about it is difficult from the existing survey of AOSD.

References

[1] Christina Chavez, Alessandro Garcia, Uirá Kulesza, Cláudio Sant'Anna, Carlos Lucena,"Crosscutting Interfaces for Aspect-Oriented Modeling, Extended version of the paper Taming Heterogeneous Aspects with Crosscutting Interfaces, XIX Brazilian Symposium on Software Engineering, Uberlandia, 2005.

[2] Awais Rashid, Thomas Cottenier, Phil Greenwood, Ruzanna Chitchyan, Regine Meunier, Roberta Coelho, Mario Südholt, Wouter Joosen,"Aspect-Oriented Software Developmentin Practice: Tales fromAOSD-Europe, Lancaster University, UK, École des Mines de Nantes, France, Federal University of Rio Grande do Norte, Brazil, Siemens AG, GermanyKatholieke Universiteit Leuven, Belgium, 0018-9162/10/$26.00 © 2010 IEEE Published by the IEEE Computer Society, February-2010.

[3] Buschmann, F. et al.,"Pattern-Oriented Software Architecture: A System of Patterns. 1996: Wiley and Sons.

[4] Chavez, C. A Model-Driven Approach to AspectOriented Design. PhD Thesis, PUC-Rio, Rio de Janeiro, Brazil, April 2004.

[5] Chavez, C., Lucena, C. A,"Theory of Aspects for Aspect Oriented Development", Proc. of the Brazilian Symposium on Software Engineering (SBES'2003), Manaus, Brazil, pp. 130-145, 2003.

[6] Kiczales, G. et al.,"Aspect-Oriented Programming. In: European Conf. on Object-Oriented Programming(ECOOP), LNCS 1241, Springer, Finland, pp. 220-242, 1997.

[7] Tarr, P. et al. N Degrees of Separation: Multi-Dimensional Separation of Concerns. In: Proc. of theICSE'99, May 1999, pp. 107-119.

[8] Filman, R., Elrad, T., Clarke, S., Aksit, M. Aspect-Oriented Software Development. Addison-Wesley, 2005.

[9] Seventh International Workshop on Aspect-Oriented Modeling, [Online] Available: http://dawis.informatik.uni-essen.de/events/AOM_MODELS2005/Montego Bay, Jamaica, October 2, 2005.

[10] Meyer, B. Object-Oriented Software Construction. 2nd Edition. Prentice Hall, 1997.

[11] Parnas, D. On the Criteria to Be Used in Decomposing Systems into Modules. In: Communications of theACM, 15 (12), pp. 1053-1058, 1972.

[12] Mezini, M., Ostermann, K. Conquering Aspects with Caesar. In: Proc. of the 2nd Intl Conf. on Aspect-Oriented Software Development (AOSD'2003), Boston, USA, pp. 90-99, 2003.

[13] Clarke, S., Walker, R. J. Generic Aspect-Oriented Design with Theme/UML. In: Aspect-Oriented Software Development, Addison-Wesley, pp. 425-458, 2005.

[14] Colyer, A., Clement, A. Large-scale AOSD for middleware. Proc. of the 3rd Intl Conf. on AspectOriented Software Development (AOSD'2004), March 2004, Lancaster, UK, pp. 56-65.

[15] Chavez, C.; Garcia, A.; Lucena, C. Some Insights on the Use of AspectJ and Hyper/J. In: Work. on Aspect Oriented Programming and Separation of Concerns, Lancaster, UK, 2001.

[16] Filman, R. What Is Aspect-Oriented Programming, Revisited. In: Workshop on Advanced Separation ofConcerns, 15th ECOOP, Budapest, 2001.

[17] Gamma, E. et al. Design Patterns: Elements of ReusableObject-Oriented Software. Addison-Wesley, Reading, 1995.

[18] Garcia, A., Sant'Anna, C., Figueiredo, E., Kulesza, U.Lucena, C., Staa, A. Modularizing Design Patterns withAspects: A Quantitative Study. In: Proc. of the 4thIntl Conf. on Aspect-Oriented Software Development(AOSD'2005), Chicago, USA, pp. 3-14, 2005.

[19] Chavez, C., Lucena, C. A Metamodel for Aspect Oriented Modeling. In: Workshop on Aspect-oriented Modeling with the UML, 1st Intl Conf. on Aspect Oriented Software Development, Netherlands, 2002.

[20] Garcia, A. et al. The Learning Aspect Pattern. Proc. Ofthe 11th Conference on Pattern Languages ofPrograms (PLoP2004), Monticello, USA, September 2004.

[21] Garcia, A. From Objects to Agents: An Aspect-Oriented Approach. Doctoral Thesis, PUC-Rio, Rio de Janeiro, Brazil, April 2004.

[22] Garcia, A., Lucena, C., Cowan D. Agents in Object-Oriented Software Engineering. In: Software: Practice & Experience, Elsevier, 34 (5), pp. 489-521, 2004.

[23] Garcia, A., Sant'Anna, C., Chavez, C., Lucena, C., StaaA. Separation of Concerns in Multi-Agent Systems: An Empirical Study. In: Software Engineering forMulti-Agent Systems II, Springer, LNCS 2940, pp. 49-72, 2004.

[24] Garcia, A., Silva, V., Chavez, C., Lucena, C. Engineering Multi-Agent Systems with Aspects and Patterns. J.Braz. Computer Society, 1(8), July 2002, pp. 57-72.

[25] Hannemann, J., Kiczales, G. Design Pattern Implementation in Java and AspectJ, In: Proc. OfOOPSLA'02, pp. 161-173, 2002.

[26] Harrison, W., Ossher, H. Subject-Oriented Programming (A Critique of Pure Objects). In: Proc. OfOOPSLA'93, 1993, pp. 411-428.

[27] D. Wiese, R. Meunier,"Large Scale Applicationof AOP in the Healthcare Domain: A Case Study," Keynote address, 7th Int'l Conf. Aspect-Oriented SoftwareDevelopment (AOSD 08), 2008.

[28] Kiczales, G., Mezini, M. Aspect-Oriented Programming and Modular Reasoning. In: Proc. of the ICSE'05,New York, NY, USA, 2005. ACM Press, pp. 49-58.

[29] Garcia, A., Kulesza, U., Lucena, C. Aspectizing Multi-Agent Systems: From Architecture to Implementation.Software Engineering for Multi-Agent Systems III.Springer-Verlag, LNCS 3390, pp. 121-143, 2004.

[30] Lieberherr, K., Lorenz, D., Mezini, M. Programming with Aspectual Components. Tech. Report NU-CCS99-01, College of Computer Science, North-easternUniversity, Boston, MA, 1999.

[31] Massoni, T., et al. PDC: Persistent Data Collections Pattern. In Proc. of the SugarLoafPLoP´2001, Brazil, 2001.
[32] Chavez, C.; Lucena, C.,"Design Support for Aspect oriented Software Development", In: Workshop on Advanced Separation of Concerns in Object-Oriented Systems (OOPSLA 2001), Tampa, USA, 2001.
[33] Clarke, S., Walker, R. J. Composition Patterns: An Approach to Designing Reusable Aspects", In: Proc. of the 23rd International Conference on Software Engineering (ICSE), Toronto, Canada, May 2001.
[34] Aspect-Oriented Software Development: [Online] Available: http:// aosd.net
[35] AspectJ Team. The AspectJ Programming Guide: [Online] Available: http://eclipse.org/aspectj/.
[36] Hyper/JWeb Page: [Online] Available: http://www.research.ibm.com/hyperspace/HyperJ/HyperJ.htm, 2001.

Priyanka Sarode received B.Sc. degree in 2006, MCA degree from Rashtrasant Tukdoji Maharaj Nagpur University, Nagpur, Maharashtra, India, in 2009. At present, pursuing Ph.D.in Computer Science and a research fellow at Inter Institutional Computer Center; RTM Nagpur University, Nagpur, Maharashtra, India. Herresearch work is acknowledged by BARTI, Pune.

Dr. Ravikant N Jugele AnAssociate Professor, Department of Computer Science, Shri Shivaji Science College, Nagpur. Maharashtra, India.