ISSN 2047-3338

How to Measure Coupling in AOP from UML Diagram

Sushil Garg[1], K. S. Kahlon[2] and P. K. Bansal[3]
[1]Professor, CSED, RIMT – IET, Mandi GobindGarh, Punjab, India
[2]Dean, Research Dept., GNDU, Amritsar, Punjab, India
[3]Ex-Principal, MIMIT, Malout, Punjab, India

***Abstract*– The aspect-oriented programming (AOP) is a new paradigm for improving the system's features such as modularity, readability and maintainability. Aspect-oriented software development (AOSD) is a new technique to support separation of concerns in software development. In aspect-oriented (AO) systems, the basic components are aspects or classes, which consist of attributes (aspect or class instance variables) and those modules such as advice, intertype declarations, point-cuts, join-points and methods. Coupling is an internal software attribute that can be used to indicate the degree of interdependence among the components of a software system. Thus, in AO systems, the coupling is mainly about the degree of interdependence among aspects, classes and interfaces. To test this hypothesis, good coupling measures for AO systems are needed. In this paper, we apply a coupling metrics suite on UML diagram of AOP. We first present a UML diagram for AO systems which is specially designed to count the dependencies between aspects and classes, aspect and interfaces and other aspect oriented features in the systems. Based on this UML diagram, we formally define various coupling measures in terms of different types of dependencies between aspects, classes and interfaces.**

***Index Terms*– Aspect Oriented Programming (AOP), Unified Modeling Language (UML), Aspect Oriented (AO) Systems and Aspect Oriented Software Development (AOSD)**

I. INTRODUCTION

ASPECT Oriented Programming (AOP) is an emerging discipline in Software Engineering. Aspect-oriented software development (AOSD) is a new technique to support separation of concerns in software development [1] – [5]. Such concerns are known as crosscutting concerns. Examples of crosscutting concerns include tracing, logging, caching, resource pooling and so on. The techniques of AOSD make it possible to modularize crosscutting aspects of a system. Like objects in object-oriented software development, aspects in AOSD may arise at a stage of the software life cycle, including requirements specification, design, implementation, etc. The ability to modularize crosscutting concerns is expected to improve comprehensibility, parallel development, reuse and ease of change [6], [7], [8] reducing development costs, increasing dependability and adaptability.

The current research so far in AOSD is focused on problem analysis, software design, and implementation techniques. The most popular AOP model today is as implemented in AspectJ [8], [9]. AspectJ extends Java with several complementary mechanisms, namely join points (JPs), pointcut descriptors (PCDs), advice, introduction and aspects. Joinpoints represent well-defined points in the execution of a program. Not every execution point is a join point only those points that can be used in disciplined and principled manner. Examples of joinpoints in AspectJ include method calls, access to class members, and the execution of exception handler blocks. A Pointcut Descriptors is a language construct that picks out a set of join points. Joinpoints are described by point cut declaration. Point cuts can be defined in classes or in aspects and can be named or be anonymous. Advice is a code that executes before, after, or around a join point. Basically, advice is a code that executes at each join point picked out by a point cut. Introduction is a member of an aspect but it defines or modifies a number of another type or class. With introduction we can add method to an existing class, add fields to existing class and implement an interface in an existing class. Advice, pointcuts and ordinary data members and methods are grouped into class-like modules called Aspects. Some existing AOP languages and frameworks provide a very similar composition model to the AspectJ one, such as Springs AOP framework and JBoss AOP.

Coupling and cohesion are two structural attributes whose importance is well-recognized in the software engineering community. In this paper, we focus on coupling measurement for AO systems has been studied in [5], [11]. Coupling is the degree to which components depend on one another. There are two types of coupling, "tight" and "loose". Loose coupling is desirable for good software engineering but tight coupling may be necessary for maximum performance. It has been recognized that good software design should obey the principle of low coupling. A system that has strong coupling may make it more complex because it is difficult to understand, change, and correct highly inter-related components in the system. Coupling is therefore considered to be a desirable goal in software construction, leading to better values for external attributes such as maintainability, reusability, and reliability. Recently, many coupling measures

and several guidelines to measure coupling of a system have been developed for object-oriented systems [5], [12], [13].

In an aspect-oriented program, the basic unit is an aspect or class. An aspect with its encapsulation of state (attributes) with associated modules such as advice, intertype declarations, pointcuts, and methods (operations) is a significantly different abstraction in comparison to the class within object-oriented software. Thus, in AO systems, the coupling is mainly about the degree of interdependence among aspects and classes, aspect and interfaces and other aspect oriented features. To test this hypothesis, good coupling measures for AO systems are needed. Moreover, in order to measure the coupling of an aspect-oriented system, we should consider different types of interactions between aspects and classes in the system. However, although coupling metrics has been widely studied for object-oriented systems and for AO systems also, but it has not been applied directly to the real aspect oriented system design.

Unified Modeling Language (UML) is the design language most accepted in software engineering, and is considered as a standard. It provides a powerful set of modeling tools for system analysis and design, for the definition of the system architecture, and also for specifying the system behavior. It also provides a set of mechanisms to extend or adapt UML to a specific domain. This extensibility characteristic makes it more suitable [14]. UML extensions for AOSD have been proposed to model the development process at early stages. Several authors have examined the benefits of AOSD modeling: facilitate the implementation stage or the reengineering of existing systems, obtain more reusable and comprehensive components, document early architectural decisions related in general with requirements and maintain their consistency through all the software development stages[15]. So, to better understand the dependency between various components of AOP we make UML diagram at the design phase.

In this paper, we apply a coupling metrics suite on UML diagram of AOP. We first present a UML diagram for AO systems which is specially designed to count the dependencies between aspects and classes, aspect and interfaces and other aspect oriented features in the systems. Based on this UML diagram, we formally define various coupling measures in terms of different types of dependencies between aspects, classes and interfaces. Because aspect-oriented paradigm significantly different from object-oriented paradigms, we really need to develop a notion of coupling for AO systems, which is an indicator of the degree to which the components in the system interact each other.

We hope that by examining the ideas of the coupling in aspect-oriented systems from several different viewpoints and through independently developed measures, we can have a better understanding of what the coupling is meant in AO systems and the role that coupling plays in the development of quality aspect-oriented software.

This paper is structured as follows: Section 2 briefly introduces the UML diagram for AOP and its notations also, Section 3 defines the coupling metrics for AOP, Section 4 measures coupling from UML in AOP and Section 5 contains Conclusion.

II. UML NOTATIONS FOR AOSD

A. Introduction to UML

UML is a standard Unified Modeling Language used to create and document software artifacts. It includes many useful ideas and concepts that have their roots in various individual methods and theories. UML provides numerous modeling techniques, including several types of diagrams, model elements, notation and guidelines. These techniques can be used in various ways to model different characteristics of a software system. Key features of UML comprise: support for model refinement, extension mechanisms (stereotypes, tagged values, and constraints), and a language for expressing constraints (known as the object constraint language, OCL).

UML has established itself as a well accepted modeling language that provides adequate support for object-oriented and component-based software development. Basically, there are various possibilities of using UML to model crosscutting concerns in a software system. For instance, join points can be represented in UML as model elements, but their effect can also be shown in different diagram types of UML, e.g., collaboration, sequence and state chart diagrams. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software [15].

B. Notations

UML Notations in AOP can be seen from Table 1:

Table 1: UML Notations for AOSD

Concepts	Disciplines		
	Requirements	Analysis	Design
concern			
candidate aspect (crosscutting concern)			
aspect			
composition			
weaving			
join point	≪match point≫		
advice			
pointcut core class			
relations concern - candidate aspect			
relations aspect - class			

C. Goals of UML

The primary goals in designing of UML are:

1. Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
2. Provide extensibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development processes.
4. Provide a formal basis for understanding the modeling language.
5. Support higher-level development concepts such as collaborations, frameworks, patterns and components.
6. Integrate best practices.

D. Why Use UML

- Improve the quality of software.
- Reduce cost and time to market.
- Solve problems like concurrency, replication, security, fault tolerance.

III. ASPECT ORIENTED (AO) COUPLING METRICS

In this section, the Chidamber and Kemerer's metrics suite is revised. Some of the metrics are adapted or extended, in order to make them applicable to the AOP software. Since the proposed metrics apply both to classes and aspects, in the following the term module will be used to indicate either of the two modularization units. Similarly, the term operation subsumes class methods and aspect advices or introductions [16].

A. Weighted Operations in Module (WOM)

- It counts number of operations in a given module.
- WOM captures the internal complexity of a module in terms of the number of implemented functions.
- How much time and effort is required to develop the module is predicted by complexity and number of operations involved in the module.
- Modules with large numbers of operations are likely to be more application specific, limiting the possibility of reuse.

B. Depth of Inheritance Tree (DIT)

- Depth of inheritance of a class is its depth in the inheritance tree, if multiple inheritances is involved. It is the length of the longest path from a given module to the class/aspect hierarchy root.
- The deeper a module is in the hierarchy, the greater the number of operations it is likely to inherit, making it more complex to predict its behavior.
- Deeper trees constitute greater design complexity, since more operations and modules are involved.
- The deeper a particular module is in the hierarchy, the greater the potential reuse of inherited operations.

C. Number of Children (NOC)

- NOC is the number of immediate sub- classes or sub-aspects of a given module. The number of children of a module indicates the proportion of modules potentially dependent on properties inherited from the given one.
- Greater the number of children then greater the reuse due to inheritance.
- If a module has a large number of children, it may be a case of misuse of sub-classing.
- If a module has a large number of children, it may require more testing of the operations in that module.

D. Coupling on Advice Execution (CAE)

- It is the number of aspects containing advices possibly triggered by the execution of operations in a given module.
- There is an (implicit) dependence of the operation from the advice. Thus, the given module is coupled with the aspect containing the advice and a change of the latter might impact the former.
- This kind of coupling is absent in OO systems.

E. Coupling on Method Call (CMC)

- It is the number of modules or interfaces declaring methods that are possibly called by a given module.
- If we use high number of methods from many different modules indicates that the function of the given module cannot be easily isolated from the others. High coupling is associated with a high dependence from the functions in other modules.
- Examples, constructor calls are counted as a method call, calls from introduced methods are counted as a call from aspect, introduced method calls are counted as an aspect's member calls.

F. Coupling on Field Access (CFA)

- It is the number of modules or interfaces declaring fields that are accessed by a given module.
- CFA measures the dependences of a given module on other modules, but in terms of accessed fields.
- In OO systems this metric is usually close to zero, but in AOP, aspects might access class fields to perform their function.
- Examples, field access from introduced methods are counted as a access from aspect, access to introduced fields are counted as an access to aspect's fields.

G. Response for a Module (RFM)

- In this, the number methods and advices potentially executed in response to a message received by a given module.
- The main reason to apply it to AOP software is associated with the implicit responses that are triggered whenever a pointcut intercepts an operation of the given module.

- If a large number of methods can be invoked in response to message, the testing & debugging of the module becomes difficult.
- The larger the number of methods that can be invoked from a class, the greater the complexity of the class.

H. Coupling between Modules (CBM)

- It is the number of modules or interfaces declaring methods or fields that are possibly called or accessed by a given module.
- This is a combination of CFA and CMC metrics.

I. Crosscutting Degree of an Aspect (CDA)

- It is the number of modules affected by the pointcuts and by the introductions in a given aspect.
- This is a brand new metric, specific to AOP software.
- This gives an idea of the overall impact an aspect has on the other modules.

III. MEASURING COUPLING FROM UML DIAGRAM IN AOP

We next apply our coupling metrics that are define in section 3 on UML diagram that is made in aspect oriented systems. Our metrics focuses on coupling caused by dependencies that occur between aspect and class, aspect and interfaces and other aspect oriented features in an AO system. In the following, Figure 1, we describe the types of dependencies between various aspect oriented features through UML diagram.

In this section, the weaving mechanism of AspectJ is implemented in the UML. A new UML relationship is introduced to denote the relationship between aspects and their base classes [17].

The basic subject/observer protocol is designed as an abstract aspect, named *SubjectObserverProtocol* that crosscuts two interfaces, named *Subject* and *Observer* (see Figure 1, upper part). The interfaces specify operations that entities participating in the subject/observer protocol have to provide. These operations (named *addObserver, removeObserver, getObservers, getData, setSubject, getSubject, and update*) are not implemented by the interfaces, though. Instead, they are implemented by the aspect by means of introductions, named *Subject* and *Observer*. Further, the aspect specifies an after advice (given the "pseudo" identifier "*advice_id01*") that implements the notification of observers and gets executed after a join point contained in the pointcut *stateChanges* is reached. That pointcut, though, is left undefined (i.e., "abstract") by the abstract aspect *SubjectObserverProtocol* and must be designated to a concrete set of join point of a particular application domain by a concrete aspect. This is accomplished by the concrete aspect *SubjectObserverProtocolImpl* that applies the subject/observer protocol implemented by the abstract aspect *SubjectObserverProtocol* to the classes *Button* and *ColorLabel*.

To do so, the concrete aspect specifies a pointcut that designates all call join points related to invocations of the *click*

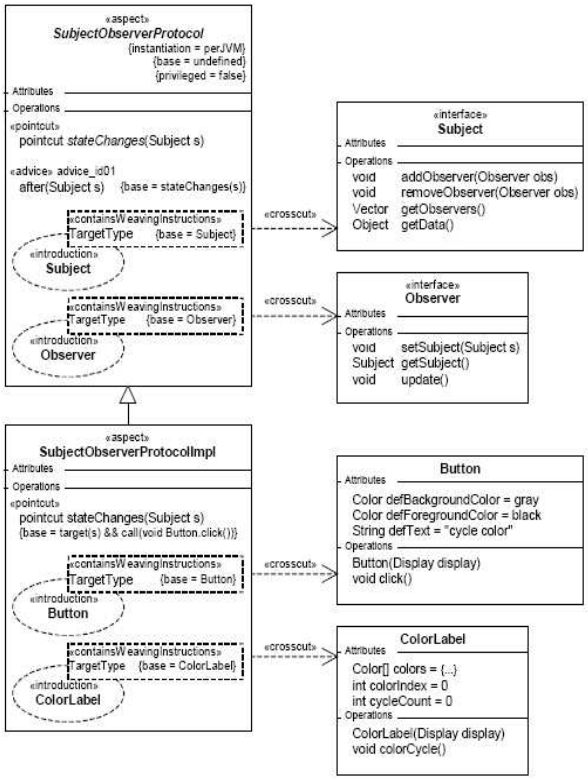

Fig. 1. Aspect Oriented Design Model

operation of a *Button* object. Besides that, the concrete aspect specifies that the *Button* class implements the *Subject* interface and that the *ColorLabel* class implements the *Observer* interface by means of its introductions *Button* and *ColorLabel*. The crosscutting effects of aspects on their base classes are visualized by the «crosscut» relationships that represent the weaving mechanism. This relationship is explained in [17].

Now we apply the coupling metrics on UML diagram of AOP for measuring the interdependency between the various aspect oriented features. This determines that which component or class or aspect is how much coupled to other component in AO systems. Our coupling metrics are defined on counting, for each aspect, the number of dependencies between aspect and classes, aspect and interfaces etc. This can be determined through Table 2.

Table 2: AOP Metrics for Aspect Oriented Design Model

COUPLING METRICS	COUPLING VALUES
WOM	15
DIT	0
NOC	2
CAE	1
CMC	4
CFA	4
RFM	11
CBM	4
CDA	2

IV. CONCLUSION

In this paper, we proposed an AOP metrics suite for measuring the coupling in AO systems. We first, present a UML diagram for AO systems which is specially designed to count the dependencies between aspects and classes, aspect and interfaces and other aspect oriented features in the systems. Based on this UML diagram, we formally define various coupling measures in terms of different types of dependencies between aspects, classes and interfaces. The coupling metrics proposed in this paper focused on dependencies between aspects and classes, aspects and interfaces, between aspects or between classes and interfaces. We also apply our coupling metrics to real aspect oriented system design which helps us to understand the interdependency among different components of the AO system.

REFERENCES

[1] L. Bergmans and M. Aksits, "*Composing crosscutting Concerns Using Composition Filters*", Communications of the ACM, Vol.44, No.10, pp.51-57, October 2001.

[2] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. M. Loingtier, and J. Irwin, "*Aspect-Oriented Programming*", Proceedings of the 11th European Conference on Object-Oriented Programming, pp.220-242, LNCS, Vol.1241, Springer-Verlag, June 1997.

[3] K. Lieberher, D. Orleans, and J. Ovlinger, "*Aspect Oriented Programming with Adaptive Methods*," Communications of the ACM, Vol.44, No.10, pp.39-41, October 2001.

[4] P. Tarr, H. Ossher, W. Harrison, and S. M. Sutton, "*N Degrees of Separation: Multi-Dimensional Separation of Concerns*," Proceedings of the International Conference on Software Engineering, pp.107-119, 1999.

[5] Jianjun Zhao, "*Measuring Coupling in Aspect-Oriented Systems*", Fukuoka Institute of Technology, Japan, 2004.

[6] Avadhesh Kumar, Rajesh Kumar, P.S. Grover, "*AChange Impact Assessment in Aspect-Oriented Software Systems*", International Software Engineering Conference Russia 2006 (SECR-2006), pp.83-87, Dec 2006.

[7] Avadhesh Kumar, Rajesh Kumar, P.S. Grover, "*An Evaluation of Maintainability of Aspect-Oriented Systems: a Practical Approach*", International Journal of Computer Science and Security, Vol -1, Issue-2, pp. 1-9, Aug 2007.

[8] Avadhesh Kumar, Rajesh Kumar, P.S. Grover, "*Towards a Unified Framework for Cohesion Measurement in Aspect-Oriented Systems*", Proceedings of the 19th Australian Conference on Software Engineering. ASWEC 2008, pp.57-65, March 2008.

[9] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. "*An Overview of AspectJ*". In Proceedings of the 15th European Conference on Object- Oriented Programming, pp. 327–355, Springer, 2001.

[10] V. C. Garcia, E. K. Piveta, D. Lucrédio, A. Álvaro, E. S. Almeida, L.C. Zancanella, & A.F. Prado, "*Manipulating crosscutting concerns*" , Proc. 4th Latin American Conf. on Patterns Languages of Programming (SugarLoafPLoP), Porto das Dunas, CE, Brazil, 2004.

[11] J. Zhao and B. Xu, "*Measuring Aspect Cohesion*," Proc. Fundamental Approaches to Software Engineering (FASE'2004*)*, LNCS 2984, pp.54-68, Springer Verlag, Barcelona, Spain, March 29-31, 2004.

[12] S. R. Chidamber and C. F. Kemerer. "*A Metrics Suite for Object-Oriented Design*", IEEE Transactions on Software Engineering, pp.476-493, Vol.20, No.6, 1994.

[13] M. Hitz and B. Montazeri, " *Measuring Coupling and Cohesion in Object-Oriented Systems*", Proceedings of International Symposium on Applied Corporate Computing, pp.25-27,Monterrey,Mexico, October 1995.

[14] Fernando Asteasuain, Bernardo Contreras, Elsa Est´evez, Pablo R. Fillottrani, "*Evaluation of UML Extensions for Aspect Oriented Design*", Universidad Nacional del Sur Av. Alem 1253 – (8000) Bah´ıa Blanca, Argentina, 2003.

[15] Francisca Losavio, Alfredo Matteo, Patricia Morantes, "*UML Extensions for Aspect Oriented Software Development*", Journal of Object Technology, Published by ETH Zurich, Chair of Software Engineering, Vol. 8, No. 5, 2009.

[16] Kotrappa Sirbi, Prakash Jayanth Kulkarni, "*Impact of Aspect Oriented Programming on Software Development Quality Metrics*", Global Journal of Computer Science and Technology, Vol. 10 Issue 7 Ver. 1.0, pp. 28-36, September 2010.

[17] Dominik Stein, "*An Aspect-Oriented Design Model Based on AspectJ and UML*", Master Thesis, University of Essen, Germany, January 2002.