GLOBAL JOURNAL OF ENGINEERING SCIENCE AND RESEARCHES

Aspect Oriented Design Languages- An Analysis

Dr.Ananthi Shesashaayee [1], Roby Jose[*2]
[1] Research Supervisor, [2] Research Scholar
PG & Research Department of Computer Science and Applications
Quaid E Millath College for Women, Chennai
ananthy.research@gmail.com, roby.research@gmail.com

ABSTRACT

Aspect oriented software development is an emerging software development technology that seeks new modularizations of software systems, in computing. Today typical enterprise and internet applications have to tackle "concerns" like security, transactional behavior, logging et al. Many important concerns often crosscut several objects and classes of object oriented systems. AOSD is a favorable model to promote improved separation of concerns, leading to the production of software systems that are easily maintainable. Nevertheless implementation wise AOSD has made remarkable progress and number of efficient technologies has been developed, but there is no satisfactory design solution for AOSD. This position paper presents an analysis of the design languages for aspect oriented programming paradigm.

Keywords: Aspect Oriented Software development (AOSD), Aspect Oriented Design Language (AODL), Design language, Modeling Language.

I. INTRODUCTION

Aspect-Oriented Software Development(AOSD) is a programming paradigm that overcomes the limitations of Object- Orientation (Programming) providing more suitable abstractions for modularizing crosscutting concerns[1,2] that cannot be decomposed from the rest of the software artifacts. Many important concerns crosscut several objects and classes of object-oriented systems. AOSD is a favorable paradigm to promote improved separation of concerns. Aspect Oriented Software Development (AOSD) is an approach where the basic development activities are performed starting from the assumption that the system will be implemented with an Aspect Oriented Programming (AOP) language. AOP allows the modular implementation of crosscutting concerns-concerns whose implementation is scattered throughout system modules. AOSD deals with modularity problems that are not handled well by other approaches including structured programming and object oriented programming. Typical enterprise and internet applications today have to address "concerns" like security, transactional behavior, logging et al. It is observed that object oriented abstractions currently are unable to capture all concerns in a software system. Many key concerns often crosscut several objects and classes of object oriented systems. The basic abstractions of object oriented software development are classes, objects and attributes. These abstractions however sometimes may not be acceptable for separating special concerns. AOSD is a favorable model to support improved separation of concerns, leading to the production of software systems that are easily maintainable. AOSD has been proposed as a technique for improving separation of concerns in the edifice of OO software and supporting improved usability and ease of evolution. So Aspect Orientation complements, not replaces Object Orientation. AOSD uses aspects as a new abstraction and provides a new mechanism for composing aspects and components.

AOP refurbish modularity by developing the cross-cutting concerns, or aspects, in isolation and then combining them with other modules using declarative or programmatic mechanisms that are modular. That is, the points of intersection are defined once, in one place, making them easy to understand and maintain. The other modules require no modifications to be advised by the aspects. This "intersection" process, sometimes called weaving, can occur at build or run time. Aspect oriented programming (AOP) addresses these problems at coding level as can be found e.g. in Aspect[17] and offers low-level support for separation of concerns at the design level. A lack of design support leads to a gap between design and implementation which worsens the desired results. To gain the AOP benefits at earlier stages in the software development lifecycle, similar separation capabilities must be provided also at the design level. Since aspect orientation is not a replacement for object orientation the position paper is about design languages for aspect orientation that extends Unified Modelling Language (UML). UML is the widely accepted design language for object Orientation.

The rest of the paper is organized as follows. Section 2 briefly introduces AspectJ, a general aspect-oriented programming language based on Java. Section 3 give modelling approaches in aspect oriented software development paradigm that extends UML. A discussion is initiated in Section 4 based on study about section3.The paper is concluded in Section 5.

II. ASPECT-ORIENTED PROGRAMMING AND ASPECTJ

The works discussed in the position paper are developed as a design notation for AspectJ. So it is vital to mention AspectJ and the new constructs introduced by AspectJ. AspectJ [3] is a seamless aspect-oriented extension to Java [4] by adding some new concepts and associated constructs. These concepts and associated constructs are called join points, point cut, advice, inter-type declaration, and aspect. The aspect is the modular unit of crosscutting implementation. Each aspect encapsulates functionality that crosscuts other classes in a program. An aspect can be instantiated, can contain states and methods, and also may be specialized with sub aspects. An aspect is combined with the classes it crosscuts according to specifications given within the aspect. Moreover, an aspect can use an inter-type declaration construct to declare fields, methods, and interface implementation declarations for classes. Declared members may be made visible to all classes and aspects (public intertype declaration) or only within the aspect (private intertype declaration), allowing one to avoid name conflicts with pre-existing elements

A central concept in the composition of an aspect with other classes is called a join point. A join point is a well defined point in the execution of a program, such as a call to a method, an access to an attribute, an object initialization, an exception handler, etc. Sets of join points may be represented by point cuts, implying that such sets may crosscut the system.

An aspect can specify advice to define code that executes when a point cut is reached. Advice is a method-like mechanism which consists of instructions that execute before, after, or around a point cut.

An AspectJ program can be divided into two parts: base code which includes classes, interfaces, and other standard Java constructs and aspect code which implements the crosscutting concerns in the program. Any AspectJ implementation must ensure that the base code and aspect code run together in a properly coordinated fashion.

III. ASPECT ORIENTED UML MODELLING APPROACHES

3.1. Aspect Oriented Design Language by Iqbal et al [5]

A new design language for aspects for aspects called AODL is proposed. AODL is UMLs extension to accommodate aspects. AODL based on AspectJ technology introduces design notations for the main constructs [6] of AspectJ discussed in section II. Each notation is designed to reflect the distinctive quality of the construct. The diagrams to design aspect and their elements are Join Point Identification and Behavioural diagram, Aspect Design diagram, Aspect-Class Static Diagram and Aspect-Class Dynamic Diagram.

3.2 The Aspect-Oriented design Modelling (AODM) by Stein et al [7]

UML diagrams such as class diagram, sequence diagram and state charts are used to design AODM.AODM presents Join Point Designation Diagrams to represent conceptual model of point cuts. Static, dynamic, structural and behavioural models helped in selecting join points. This approach generated code from design models but it presented no means to represent weaving process with the base modules.

3.3 The Theme/UML approach by Clarke et al [8]

This approach uses system themes to represent crosscutting and non-crosscutting concerns. Themes are modelled as independent modules and their structures and behaviour are modelled using UML diagrams. Even though weaving process in Theme/ UML approach is carried out by template parameter instantiation the problem with this approach was complexity which made it difficult for traditional UML designers to learn this approach

3.4 AOSD with Use Cases by Jacobson et al [9]

AOSD approach separates cross cutting concerns from system requirements in the form of use case slices. The structural and behavioural properties are modelled with the help of class diagrams and sequence diagrams. AOSD also has no support for modelling weaving process.

3.5 A UML notation for AOSD by Pawlak et al [10]

The design notations presented by Pawlak et al were extensions of UML for modelling aspects and their related point cuts. This approach could design some

efficiency related crosscutting concerns such as security, fault tolerance and others. Structural properties were taken care of by static models whereas there was no support for behavioural properties. For designing the weaving process also there was no support in this model.

3.6 The AOSD profile by Aldawud et al [11]

The AOSD profile with UML-compliant design notations represented and designed aspects and their elements. Structural and behavioral modeling were supported by class diagrams and sequence diagrams. AOSD profile does not provide support for modeling weaving process.

3.7 Aspect oriented UML approach by Klein et al [12]

Aspect oriented UML approach is based on Message sequence charts, a standardized scenario language. This approach designed simplified meta model for sequence diagram using UML 2.0. Even though the UML had the extension capability this approach neither used this nor did it propose new notations as well.

3.8 UML All Purpose Transformer by Ho et al [13]

The UMLAUT is a toolkit that uses aspect oriented UML models for building specific weavers. The UMLAUT provides the user with general purpose operator that can be reused for different application with specific needs. The pro with UMLAUT is that each AO design may be developed with application specific weaver that optimizes weaving process expressed by UMLAUT

3.9 Aspect oriented class design model by Reddy et al [14]

The class design model consists of set of aspect models and primary model. The aspect models and primary models are created to obtain incorporated design view. A composition algorithm and composition directives are utilized to describe the composition approach. This approach included a prototype tool that supported default class diagram composition.

3.10 Aspect oriented UML modeling by Przybylek [15]

This approach proposed an extension to UML creating a new model called AoUML. AoUML had elements that represent basic aspect oriented constructs such as aspect, advice, point cut, introduction and crosscutting dependency.

3.11 Extension to UML meta model by Sharafi et al [16]

This extension to UML meta model captures crosscutting concerns. This newly created Meta model can be represented in standard XMI format. This is a language independent description of aspects and has the capability to support model transformations crucial to software development and maintenance.

IV. DISCUSSION

There are a lot of aspect oriented UML modelling approaches. The AO UML approaches falls in two categories. The first one being the construction of UML profiles. This does not involve any new meta model elements. The construction of UML profiles is based on predefined constraints, values and graphical representations. The second category is the extension to UML meta model. This category proposes meta model to define aspects and its crosscutting feature. The majority of the works are focused on structural modelling. Also majority of the approaches did not demonstrate modelling procedure. Moreover, a few approaches have developed their own tool, the rest used previously available tools.

V. CONCLUSION

Aspect-oriented software development is missing standardized concepts in the design phase. To make AOSD more widely accepted solutions have to be offered for designing cross-cutting concerns. Even though the paradigm have gone distances in terms of providing the code support it needs to be matured to represent concerns at the design phase. Each concern implemented in the code should be declared in the design stage so that aspects are traceable from requirements through source code.

VI. REFERENCES

[1]Kiczales.G, J.Lamping, A.Mendhekar,Ch. Maeda, Ch.Lopez, J.M. Loin ."Aspect-Oriented Programming". European conference on Object Oriented Progamming(ECOOP), LNCS 1241,Springer –Verlag, Finland, June 1997

[2] Tarr, P. et al. "N Degrees of Separation: Multi-Dimensional Separation of Concerns".Proceedings of the 21st International Conference on Software Engineering, May 1999.

[3] The AspectJ team. The AspectJ programming guide 2003

[4] G.Kiczales, J Lamping, A Mendhekar, C.Lopes, J .M Loingteir, and J.Irwin, "An Overview of AspectJ". Proceedings of the 13th European conference on Object Oriented Programming, pp 220-242, LNCS, Vol 1241, Springer-Verlag, June2000

[5]Iqbal S, Allen G "Designing Aspects with AODL", International Journal of Software Engineering Vol 4No:2 July 2011

[6] J Zhao,"Measuring coupling in aspect Oriented system". Proceedings of 10th IEEE International Software metrics Symposium (Metrics '04) Chicago, USA 2004

[7] D. Stein, S. Hanenburg and R.Unland, "A UML-based Aspect-Oriented Design notation For AspectJ",Aspect Oriented Software Development(AOSD), Enschede, The Netherlands

[8] S Clarke, E.Banniassad,"Aspect Oriented analysis and Design: The Theme approach". Addison Wesley, Reading2005

[9] I. Jacobson, P.W.Ng,"Aspect Oriented Software Development with Usecases". Addison Wesley, Reading 2004

[10]R.Pawlak, L.Seinturier, L. Duchien, L.Martelli, F.Legond Aubry , G.Florin, "Aspect Oriented Software Development with Java Aspect components". Aspect Oriented Software Development. In: Filman, R.E.,Elrad, T.Clarke, S.,Aksit,M.(eds.), ch16,pp 343-369,2005

[11]T.Elrad,O.Aldawud,A.Bader "Expressing Aspects using uml behavioral and structural diagrams". Aspect Oriented Software Development. In: Filman, R.E.,Elrad, T.Clarke, S.,Aksit,M.(eds.), ch16,pp 343-369,2005

[12] Klein, Jacques, Franck Fleurey, and Jean-Marc Jézéquel. "Weaving multiple aspects in sequence diagrams." Transactions on aspect-oriented software development III. Springer Berlin Heidelberg, 2007. 167-199.

[13] Ho, Wai-Ming, et al. "A toolkit for weaving aspect oriented UML designs." Proceedings of the 1st international conference on Aspect-oriented software development. ACM, 2002.

[14] Reddy, Y. Raghu, et al. "Directives for composing aspect-oriented design class models." Transactions on Aspect-Oriented Software Development I. Springer Berlin Heidelberg, 2006. 75-105.

[15] Przybylek, Adam. "Separation of crosscutting concerns at the design level: An extension to the UML metamodel." Computer Science and Information Technology, 2008. IMCSIT 2008. International Multiconference on. IEEE, 2008.

[16] Sharafi, Zohreh, et al. "Extending the UML metamodel to provide support for crosscutting concerns." Software Engineering Research, Management and Applications (SERA), 2010 Eighth ACIS International Conference on. IEEE, 2010.

[17] G. Kiczales, E.Hilsdale, J.Hugunin, M Kersten, J Palm and W. Griswold"An overview of AspectJ" Proceedings of 15th ECOOP" LNCS 2072, p 327-353, Springer-Verlag, 2001