I.J. Engineering and Manufacturing, **2017, 2, 11-22**
Published Online March 2017 in MECS (http://www.mecs-press.net)
DOI: 10.5815/ijem.2017.02.02

Available online at http://www.mecs-press.net/ijem

Modeling Aspects with AODML: Extended UML approach for AOD

Vaibhav Vyas [a], Rajeev G. Vishwakarma and C. K. Jha [b]

[a] *Banasthali University, Banasthali, Rajasthan, India*
[b] *IIST, Indore, M.P., India, Banasthali University, Banasthali, Rajasthan, India*

Abstract

Aspect Oriented Software Development (AOSD) has been considered one of the most promising abstractions to make software structure more maintainable and configurable. It also helps to overcome two big issues of current object oriented programming principles, to reduce the problem of code tangling and code scattering. Aspect Oriented Programming (AOP) has been focused largely in the implementation/coding phase. But nowadays the AOP has been matured enough to turn into AOSD, as it the main objective of separation of concerns right through the process of software development. In this paper we deal with the impact of aspect in development of software especially in designing aspect with Unified Modelling Language (UML). We propose visual models to incorporate aspect and aspectual constructs as an UML metamodel approach and new extensions to UML. The proposed language aspect oriented design modelling language (AODML) is an extension for aspect modelling into existing UML specifications. This paper allows designers to specify and realize aspects in the design and implementation phase explicitly. The proposed visual models, supports Aspect, aspectual components and its association with base components i.e. classes to be incorporated into UML. AODML motivates designer to get benefited to develop the system using AOSD paradigm. It allows to model aspects in design diagrams so that it can be implemented in any AOP language effectively.

Index Terms: AOSD (Aspect Oriented Software Development), AOP (Aspect Oriented Programming), UML, AODML (aspect oriented design modelling language), Separation of concerns.

1. Introduction

Aspect-Oriented Software Development (AOSD) facilitates the modularization of different intercrossing concerns in software development. In AOSD, aspect weaving is the composition mechanism that combines aspects and components in an aspect-oriented application. Aspect weaving can be performed passively, at load time or at runtime [4]. These different kinds of weavers may entail a runtime performance and a memory

* Corresponding author.
E-mail address: Vaibhavvyas4u@gmail.com, Rajeev@mail.com, Ckjha1@gmail.com

consumption cost, compared to the classical object-oriented approach. Using the Dynamic and Static Aspect Weaving (DSAW) AOSD platform, we have implemented three different scenarios of security issues in distributed systems [7]. These scenarios were developed in both the aspect-oriented and object-oriented paradigms in order to evaluate the cost introduced by static and dynamic aspect weavers. AOSD allows the separation of intercrossing concerns in software development. The modularization of different application concerns provides higher level of concern reuse, abstraction, higher legibility and improved software maintainability. However, in some scenarios, the use of AOSD may involve an increase of runtime performance and memory consumption [6]. This paper presents measure of memory consumption and runtime performance between three different applications developed using AOSD and the classical object oriented approach.

Aspects and aspect-oriented development (AOD) investigate with new modularization techniques, to separate concerns which are spread throughout an entire software system. These concerns are often entangled and intertwined if a system is structured through traditional development abstractions such as functions or classes. Aspects originate in the programming field but are currently moving towards other software engineering disciplines. Aspects are a promising approach for software product lines in particular, as they can be used to improve modularization of variable and common structures in the origin asset base and the products.

A. Aspect Oriented Programming

Xerox Palo Alto Research Center (Xerox PARC) developed Aspect Oriented Programming (AOP) in the 20th century and is invention of a programming paradigm [1]. Aspect can be thought of a class-like construct. It contains point-cuts and advices related to it. The basic idea of the aspect is to wrapping the intercrossing functionality away from the base program into different well defined modules. AOP is built on OOP and procedural programming. The fundamental way in which AOP is different from OOP is in managing crosscutting concern; in AOP execution of every concern is autonomous to crosscutting behavior inaugurated to it. AOP proposal is partitioned into two halves: the aspect program and the base program. Using OOP, the base program holds the major functionality of the system and can be implemented. The aspect program consists of the intercrossing functionality that has been modularized far away from the base program.

i. Code Tangling

When a particular module or function which covers multiple kinds of features simultaneously implements caused Code tangling. Software may cover multiple aspects like core system logic, authentication, logging, synchronization, performance and so on when particular module is implemented. This may causes presence of simultaneous concern's implementation. This will cause the code tangling. Figure 1.7 illustrates the reason of code tangling in a particular module.

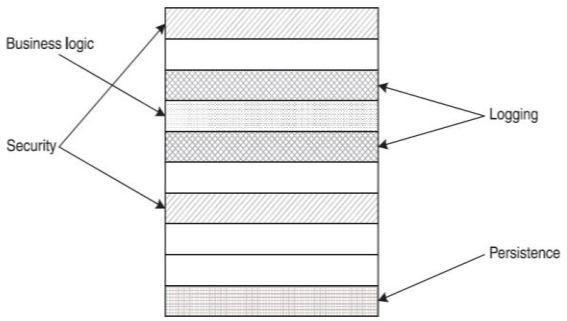

Fig.1. Represents Code Tangling Caused by Simultaneous Concern Implementation

ii. Code scattering

When single task is implemented by multiple modules then it is termed as code scattering. As discussed crosscutting concerns are spread in multiple modules of similar kind may also caused code scattering in all the modules where it gets handled. For example, in database handling for banking task in multiple modules like withdrawal, deposit, fund transfer performance concern may affect all related modules those using the database.

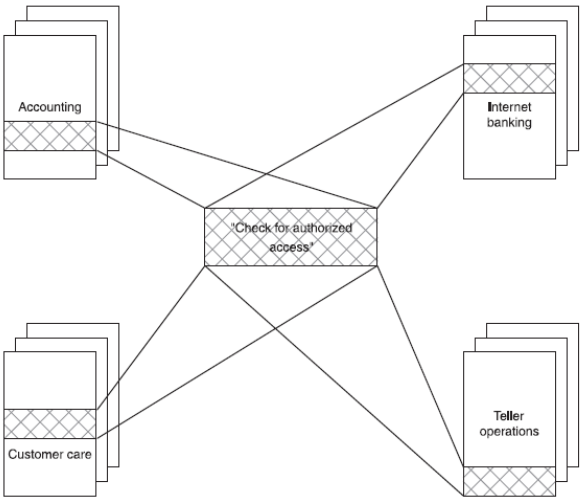

Fig.2. Represents Code Scattering Causes Implementation of Identical Code in Multiple Module

B. Aspect and aspectual components

Aspects: Aspects include all the features of class and add one more feature. With the help of weaving mechanism aspects enhance the behavior of other classes. An aspect illustration is same as an object in various ways. Aspect can be created in same way as object is created: Aspect instances and objects are too much similar to each other;

Table 1. Represents Aspect and Key Aspectual Concerns

Term	Explanation
Aspect	An Abstraction of a crosscutting concern in a program. It contains pointcuts to indicate execution points in the base program and advices to run on those execution points.
Advice	The behavior of a crosscutting concern
Join Point	An execution point where an advice is supposed to execute.
Pointcut	A set of predicates to define related join point
Weaving	A process of incorporating aspect's behavior (an advice) into base program at a specified execution point (join point).

There exist mainly two methods for using aspects. The first is to detach concerns that cut over the functional

constituent and second is to change a present approach in order to put together a new attribute. Aspects can debate both functional and nonfunctional system characteristics. At least elementary shape sustaining the purposes in question previously exist in the system requires implementing a functional property using aspects. Moreover, the insertion of non-operational characteristics must have no effect on the pattern or execution of the current system in the procedure of software expansion or software repairing

C. Aspect Modelling

According to Aram Hovsepyanet.al Aspect-Oriented Modeling (AOM) offers hold up for sorting out concerns at the design stage. It is also possible to maintain the concerns modularized at the implementation stage by targeting an aspect-oriented platform, although most AOM approaches provide ways to carry out the composition of the modularized concerns to gain a collected model. Model-driven approaches have arrived to hold up both alternatives via tools. Authors found that the all-aspectual approach outcomes in smaller, less complex and additional modular implementation.

Klaus Alfert says that in the paper they share their knowledge with modelling needs in software product lines. Non-functional requirements may have aspect-like features. A main result is that feature models provide low-level support for aspects and authors discuss how they can be enlarged to give improved hold up for feature models with aspect characteristics. FODA-based feature models provide a practically usable model for variability of needs within a whole software product line. They even give a little support for aspect characteristics of needs, but only on a low level of abstraction. It depends only on the modeler's discipline how the feature model is prepared.

2. Our Approach

AODML is basically an approach to model Aspect and its related sub components at design phase of software development. AODML is a second part of AOADML. Under AODML proposal of visual models that incorporate Aspects and component in the design model is made. AODML permits modeling of Aspect and related concepts to model with design models. Further AODML models internal components of Aspect like pointcut, join points and advices through visual models or diagrams. Further AODML specify Aspect with classes which are core building block of object oriented development.

A. Aspect Oriented Design Modeling Language (AODML)

As the efforts number of researchers (Clarke and Walker, 2002; Stein et al., 2002a) emphasizing that the need of standard Aspect Oriented Design modeling language. The language should allowed the specialized analytical and design notations in the form of graphical UML model for Aspect Oriented Software Development. As we know Unified Modelling Language (UML) is the best modeling language available towards Object Oriented software development designing. As it becomes very essential to have a de-facto graphical language for proper modeling of analysis and design aspect along with the objects[3][11].

AODML is basically an approach to model Aspect and its related sub components at design phase of software development. AODML is a second part of AORDML. Under AODML proposal of visual models that incorporate Aspects and component in the design model is made. AODML permits modeling of Aspect and related concepts to model with design models. Further AODML models internal components of Aspect like pointcut, join points and advices through visual models or diagrams. Further AODML specify Aspect with classes which are core building block of object oriented development. AODML proposes Aspect-Class Block Diagram and Aspect-Class Detail Diagram that provide integration of Aspect with current development paradigm. AODML is an effort towards Aspect Oriented software development.

AODML has been proposed for identification, definition, specification, representation and designing aspects and its component elements build with base object oriented constructs. AODML helps in modelling of Aspect,

join points, pointcuts and advices with existing object oriented methodology. Major diagrams of AODML are Aspect-class diagrams and join point detection diagram which allows structural and behavioral description aspect oriented components of the system.

AODML could be milestone towards Aspect Oriented Software modeling and designing. This modeling language allows designers to integrate aspect with currently popular UML diagrams for designing of software. This approach is about to identify and specify Aspect and Aspectual components to model with existing UML diagrams. AODML allows modeling Aspect with internal components in the design models of UML.

In AODML following diagrams or models are proposed;

i. **Join Point Detection Diagram** The main goal of Join Point Detection is to recognize the special point or points in the base programs that can be identify within the aspect as pointcut. This detection of join points helps in development of other aspectual components. Join point detection diagram describes join points as an extension to activity diagram of existing UML notation.
ii. **Pointcut-Advice Design Diagram** The primary goal of Pointcut-Advice Diagram is to show the association of a particular pointcut over the respective Advice. It describes the relationship between Individual Aspects pointcut and Advice executed on a particular joinpoint. This diagram is designed using two sections first to design pointcut and then design Advice executed on the pointcut.
iii. **Aspect Design Diagram** The Aspect Design Diagram describes Aspect and its elements diagrammatically. Aspect Diagram of AODML shows aspects and its elements like pointcut and advice at design level. Aspect diagram modeled in the form of rectangle containing three sections. First section depicts the name of Aspect. Second section depicts the pointcuts which gets executed on occurrence of particular join point in base program. Third section indicates the behavior to be executed in the form of advice on the activation of particular pointcut.
iv. **Aspect-Class Modelling Class** diagram is already a core part of object oriented designing. The approach emphasizing towards aspect oriented analysis and designing. Aspect Class modelling shows the relationship of Aspects with core component of object oriented technology, which is Classes.

   a. **Aspect-Class Block Diagram** This diagram shows abstract relationship and association between classes and Aspects. This diagram helps in identification of number of class and aspects and their associations.
   b. **Aspect-Class Detail Diagram** The main objective of Aspect Class Detail diagram is to show the Aspect with all its constituent components, classes with its sub parts and association between different classes and association between classes and Aspects. It depicts the crosscutting behavior and implementation of pointcuts and advices over the associated classes.

3. Application of AODML

Automated Teller Machine (ATM) system is very good example of distributed application having numerous of non functional and functional aspects like authentication, logging, performance and persistence. One of our papers is recognized by IEEE explorer describing ATM case study over AORML. ATM system is kind of banking system allows transaction of different services to the customer like withdrawal, fund transfer, deposit and other banking services. This case study helps us to demonstrate different models under AODML.

A. Functional Aspect

1. **Authentication:** Authentication is treated as functional Aspect. This aspect is responsible for validating customer as well as ATM Technician. It checks the authentication of actors while interacting with the system. The authentication is checked in the form of ATM card and pin number provided to the customer

2. and Technician by the Bank. Authentication of actors is done by validating it with bank database.
3. **Logging:** Logging is also treated as functional Aspect. This aspect is responsible for making log of each a every interaction of customer with ATM system. This log is helpful for ATM technician and Bank.

B. Non Functional Aspect

1. **Performance:** Performance is treated as non-functional Aspect. Performance is one of key issue to maintain for securing ATM system. Whenever customer makes any transaction with ATM system it should be completed within specified period of time. This performance of ATM should be maintained so that customer accounts cannot be hacked.

4. Discussion

Last section described the application of AODML over the ATM system. Our motive is to model all aspectual components in extended UML diagrams as explained in section II. Automated Teller Machine system is very good example of distributed application having numerous of non functional and functional aspects like authentication, logging, performance and persistence.

AODML is able to model all the components starting from Join points for ATM system as 4 use case are identified in ATM system related Jin points are captured in Fig 3- 6. After identifying the join points pointcuts needs to be identified in respective aspects. In ATM system 3 pointcuts with associated Advices are identified described in Fig. 7-9. After identifying all aspectual constructs Aspect needs to be modeled for ATM system which is further described in Fig-10-12. Fig 13 and 14 represents association of aspects with object oriented constructs class at abstract and refined level.

ATM system is modeled with AODML models. Join point detection diagram detects all available join points of ATM system. Further pointcuts and advices are modeled using pointcut-advice diagram. Identified classes and aspects are then modeled into Aspect-Class block diagram and Aspect-Class detailed diagram. Join point detection diagrams identifies execution points in the core modules of ATM system. Further Pointcut, advice and Aspect diagrams are representation of identified aspects and their internal constructs of ATM. Aspect Class diagrams showing incorporation of aspects with classes.

1. **Join point Detection Diagram**

Depositing Cash:

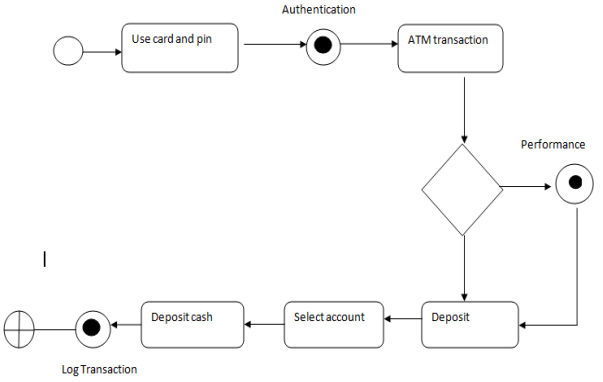

Fig.3. Represents Join Point detection of Deposit case.

Withdrawal Cash:

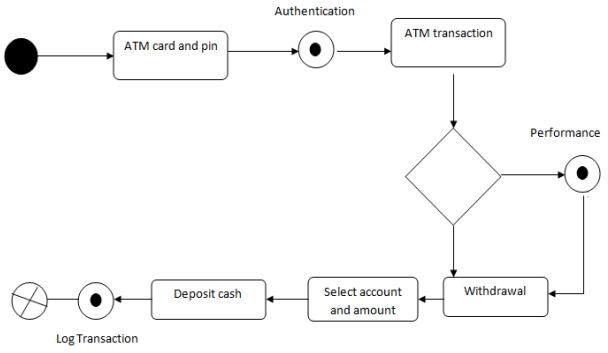

Fig.4. Represents Join Point detection of Withdrawal case.

Fund Transfer:

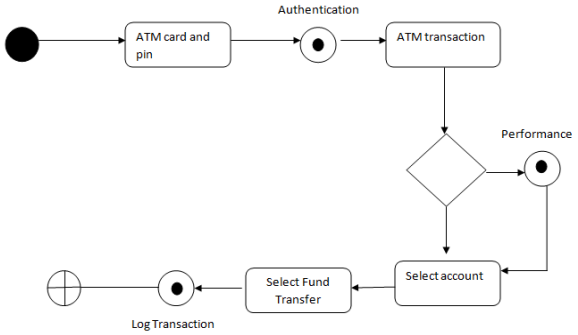

Fig.5. Represents Join Point detection of Fund Transfer case.

Manage Account:

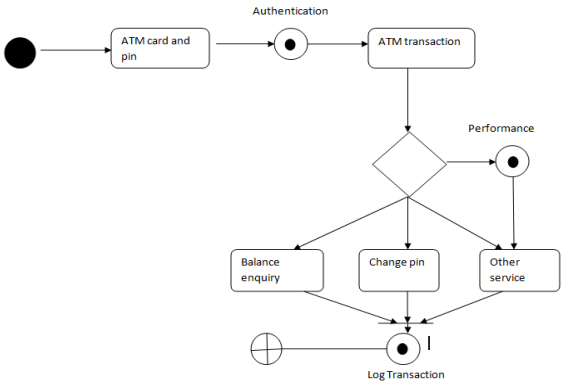

Fig.6. Represents Join Point detection of Manage Account case

2. **Pointcut-Advice Design Diagram**

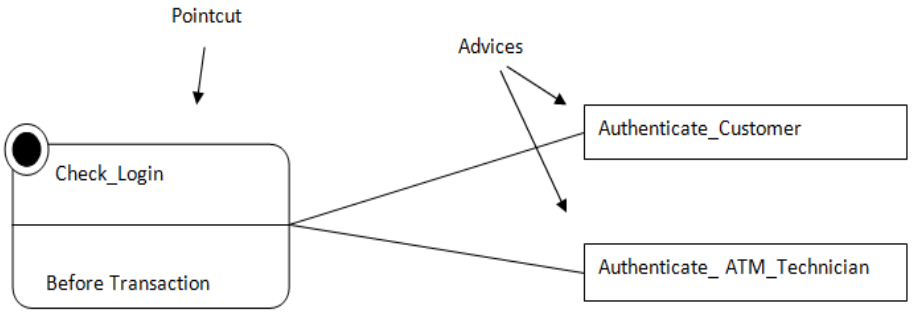

Fig.7. Represents Point Cut Check login

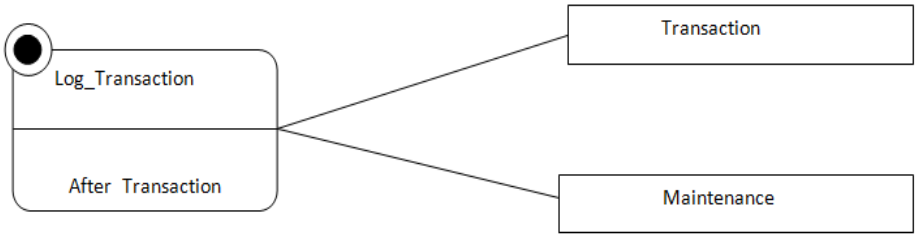

Fig.8. Represents Point Cut Log Transaction

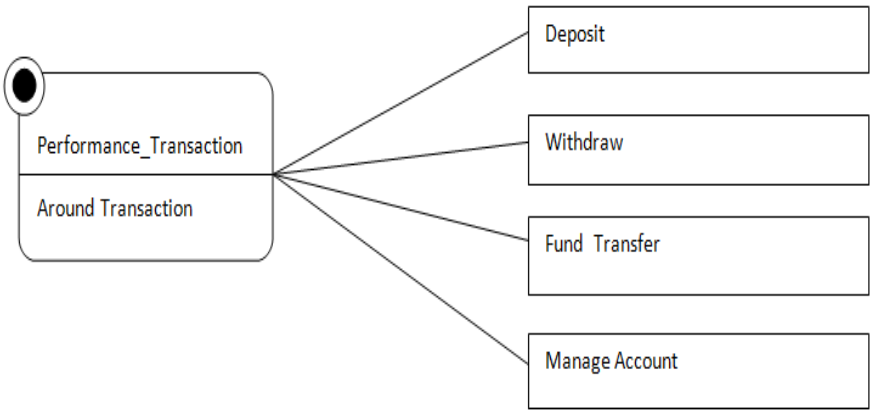

Fig.9. Represents Point Cut Performance Transaction

3. **Aspect Design Diagram**

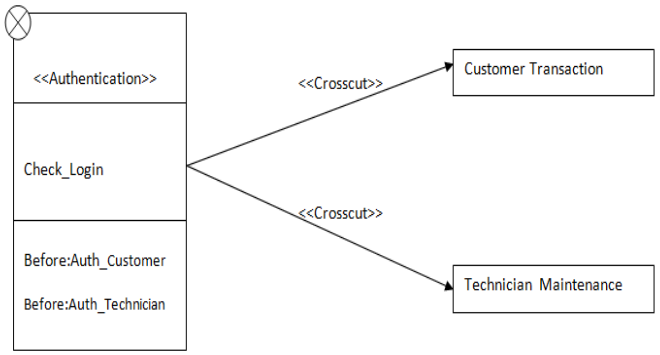

Fig.10. Represents Authentication Aspect in ATM System

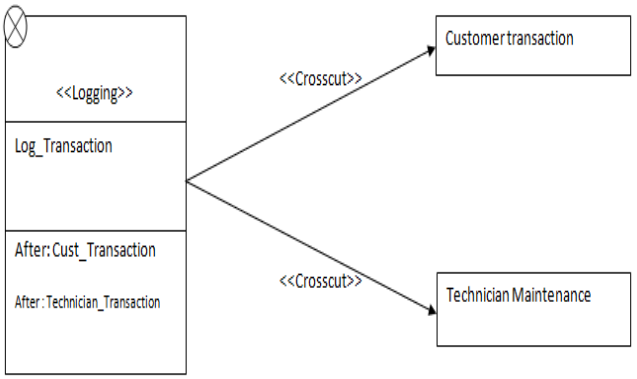

Fig.11. Represents Logging Aspect in ATM System

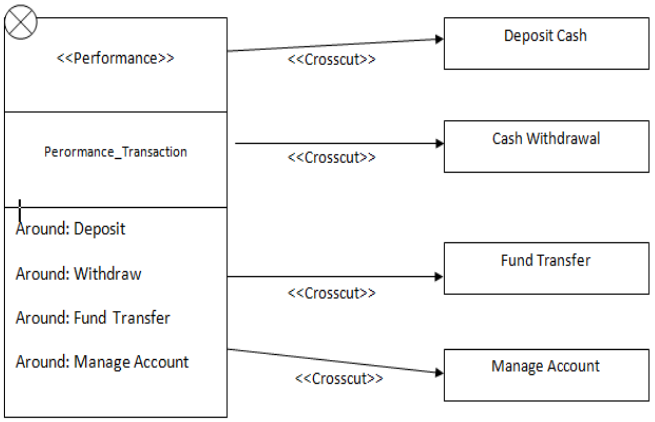

Fig.12. Represents Performance Aspect in ATM System

4. Aspect-Class Block Diagram

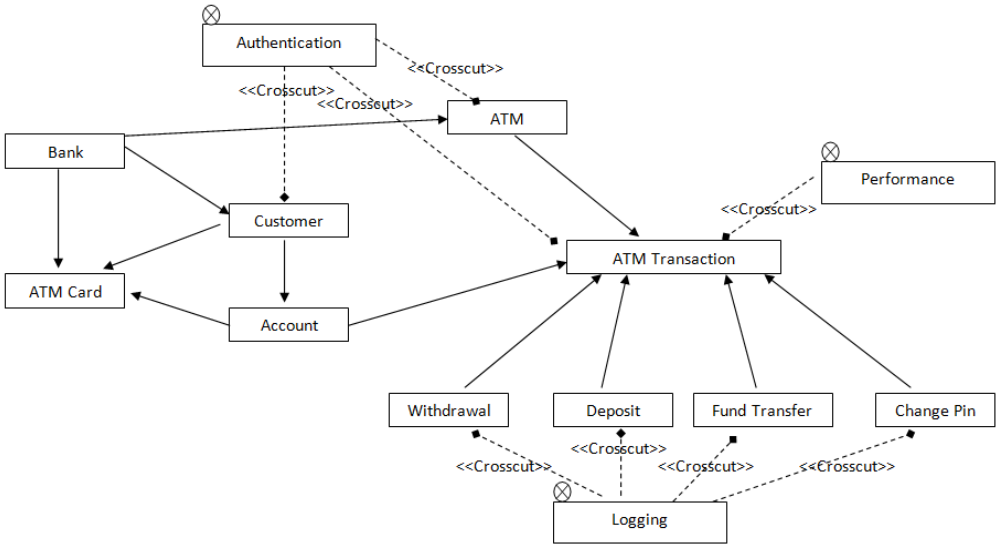

Fig.13. Represents Aspect-Class Block Diagram of ATM System

5. Aspect-Class Detail Diagram

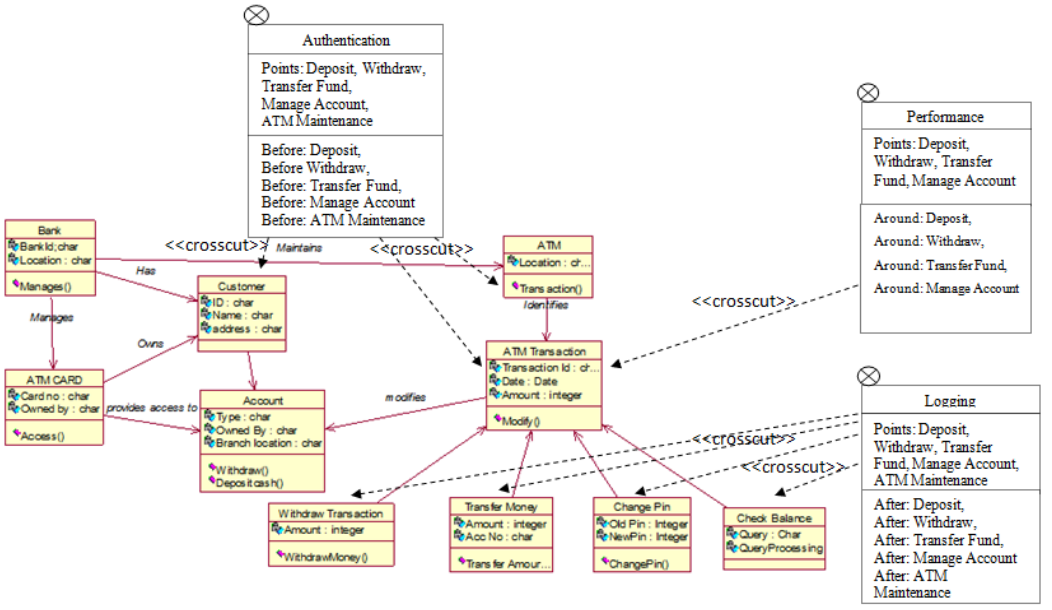

Fig.14. Represents Aspect-Class detail Diagram of ATM System

5. Related work

This section describes contemporary Aspect oriented design methods and their approach for modelling of aspects.

1. **The JAC Design Notation by Pawlak et al** The present model is proposed (Pawlak et al, 2003), (Pawlak et al, 2005) respectively which is used for developing JAC Framework, i.e., which includes an open source framework with a IDE along with modeling purpose and used as a middleware layer for various aspectual components[12]. AODM approach Stein et al., Mimics JAC Design notation and it is an approach implies implementation, and supports an external traceability from design to implementation. The internal tracking is not applicable because no refinement is made in this approach. This approach is based on lightweight UML extensions [9][10].
2. **UML Profile Approach by Aldawud et al** The UML profile approach (Aldawud et al., 2002, 2003); (Elrad et al., 2005) is an analysis and design approach for capturing and designing aspects. It is complemented with an aspect-oriented design language as well which is based on a UML profile. This is a platform-independent approach[8]. AOSD profile approach is based on the UML version 1.x Class diagrams are needed for express the structural dependency model phase machine behavioral addictions problems [1][2].
3. **The Aspect-Oriented Design Model by Stein et al.** This model was the work of Stein et al. (Dominik Stein et al, 2003). The Aspect-Oriented Design Model (AODM) was developed as a design notation for AspectJ one of the most prominent protagonists. It is aligned with the implementation, as well as permits for external traceability of project documentation to implementation [12]. Internal traceability is not apply, as the refinement of models is not expected AODM. Both AspectJ UML and have been studied for this approach, in order to find different portions corresponding AspectJ concepts in UML and also extended to support AspectJ concept if needed [11].

6. Conclusion & future work

Majority of aspect oriented modeling approaches works relatively well in handling the basic modeling tasks at design level. The paper explained new modelling language over aspect oriented modelling, AODML offered modelling notations to depict structural and behavioral crosscutting concerns of the system. AODML were used asymmetrical approach for showing the structural and behavioral aspect of the system. Future work will exploit the results of the coordination model formalization to show the effects of Aspect Oriented Analysis and Design. AODML is an approach in evolving phase and still there is an opportunity for extensions and enhancements in the language. pointcut - pointcut composition and intertype declarations need to be modeled as new notations are required for modelling these extensions. Tool support will also be provided as future work of AODML. Tool support is extremely necessary for acceptability and application of any modelling language.

References

[1] Aldawud O., Bader A., and Elrad T., 2002. "Aspect-Oriented Modelling: Bridging the Gap between Implementation and Design". Presented at Generative Programming and Component Engineering Conference (GPCE), Pittsburgh, PA, USA. pp. 189-201.

[2] Aldawud, O., Elrad, T., and Bader, A. 2003. "UML profile for aspect-oriented software development". In Proceedings of the 3rd International Workshop on Aspect Oriented Modelling.

[3] Clarke, S. and Walker, R. J., 2002. "Towards a Standard Design Language for AOSD". ACM Proceedings

on Aspect Oriented Software Development, pp. 113- 119.
[4] Conde, Patricia and Ortin, Francisco. "JINDY: a Java library to support invokedynamic", Computer Science & Information Systems, 2014.
[5] Elrad, T., Aldawud, O., And Bader, A. 2005. "Expressing aspects using UML behavioural and structural diagrams". In Aspect-Oriented Software Development, R. Filman, T. Elrad, S. Clarke, and M. Aks ¸it, Eds. Addison-Wesley, pp. 459–478.
[6] FRANCISCO ORTIN. "THE DSAW ASPECTORIENTED SOFTWARE DEVELOPMENT PLATFORM", International Journal of Software Engineering and Knowledge Engineering, 2011
[7] García, M., D. Llewellyn-Jones, F. Ortin, and M. Merabti. "Applying dynamic separation of aspects to distributed systems security: a case study", IET Software, 2012.
[8] Iqbal, S. and Allen, G., 2012. "Pointcut Design with AODL". In: The Twenty-Fourth International Conference on Software Engineering and Knowledge Engineering (SEKE 2012), July 1-3, 2012. Redwood City, California, USA. pp. 418-421.
[9] Pawlak, R., Seinturier, L., Duchien, L., Martelli, L., Legond-Aubry, F., and Florin, G., 2005. "Aspect oriented software development with Java aspect components". In Aspect-Oriented Software Development, R. Filman, T. Elrad, S. Clarke, and M. Aks ¸it, Eds. Addison-Wesley, pp. 343–369.
[10] Pawlak, R.,Duchien, L., Florin, G., Legond-Aubry, F., Seinturier, L., And Martelli, L., 2003. "A UML Notation for Aspect-Oriented Software Design". In Proceedings of the 1st Workshop on Aspect-Oriented Modelling with UML (AOSD'03).
[11] Stein, D., Hanenberg, S. and Unland, R., 2003. "Aspect-Oriented Modelling: Issues on Representing Crosscutting Features". Presented at Workshop on Aspect- Oriented Modelling (held with AOSD 2003), Boston, Massachusetts, USA.
[12] Wimmer, Manuel, Andrea Schauerhuber, Gerti Kappel, Werner Retschitzegger, Wieland Schwinger, and Elizabeth Kapsammer. "A survey on UML-based aspect-oriented design modeling", ACM Computing Surveys, 2011.

Authors' Profiles

Vaibhav Vyas, Assistant Professor, Department of Computer, Banasthali University, Rajasthan, India. He has been working in the field of Aspect Oriented Software Engineering, Aspect Oriented Modelling, AORE and AOD. Recently submitted his Ph.D. thesis on the same area. He has published several research papers in national and international conferences and journals. Other area of interest of author is Web Security, Data Analytics and IOT.

How to cite this paper: Vaibhav Vyas, Rajeev G. Vishwakarma, C. K. Jha,"Modeling Aspects with AODML: Extended UML approach for AOD", International Journal of Engineering and Manufacturing(IJEM), Vol.7, No.2, pp.11-22, 2017.DOI: 10.5815/ijem.2017.02.02