Journal of Software Engineering and Applications, 2013, 6, 98-112
http://dx.doi.org/10.4236/jsea.2013.63014 Published Online March 2013 (http://www.scirp.org/journal/jsea)

Using UML Behavioral Model to Support Aspect Oriented Model

Zahid Hussain Qaisar[1], Nauman Anwar[2], Shafiq Ur Rehman[3]

[1]Computer Science Department, Institute of Engineering and Technology, National Fertilizers Corporation (NFC-IET), Multan, Pakistan; [2]Cyber Designs, F-8 Markaz, Islamabad, Pakistan; [3]Center for Software Dependability, Mohammad Ali Jinnah University, Islamabad, Pakistan.
Email: zahidqaisar@yahoo.com, nauman.anwar@ymail.com, shafiq.rehmaan@gmail.com

Received January 15th, 2013; revised February 17th, 2013; accepted February 25th, 2013

ABSTRACT

Aspect oriented software development is an emerging paradigm of software development. The notion of this technique is separation of concerns which means to implement each concern in a single object in object oriented programming but still there are concerns which are distributed on different objects and are called crosscutting concerns while another form is Core concerns are the core functionality provided by the system but crosscutting concerns are the concerns like logging, performance etc. Modeling of aspect oriented software is different from the normal modeling of object-oriented or procedural language software, because aspects don't have the independent identity or existence and they are tightly coupled to their woven context so it is difficult to model them. The one aim of our research paper is to explore the domain of Modeling of the aspect oriented software. The goal of this research paper is to give a UML Behavioral modeling techniques in the domain of aspect oriented software development. This technique of generating UML Behavioral Model for aspects will give better understating of separations concerns.

Keywords: Aspects; Concerns; Cross Cutting Concerns; Cut Points; Join Points; Advices; Meta Data; UML and Meta Model; Aspect Oriented Modeling; Software Modeling

1. Introduction

Aspect oriented development paradigm gives the idea of the separation of concerns. Separation of concerns is not a new idea. Parnas in 1972 gave an idea that a system should be decomposed in modules so that the system should be easy to create, implement, verify and evolve. Each module should hide an aspect of the system that should be evolved independent of the other module. In other words he gave an idea to identify the independent concerns. In the consequence of this research object oriented programming is developed.

These crosscutting concerns are cause of the code tangling and code scattering [1]. In code tangling to implement one concern we have to write the code in different modules. In code scattering one piece of code is written in different classes. To address this issue we use aspect-oriented programming. Core concerns can be implemented in any object-oriented language. To implement cross cutting concerns we use aspects.

Object oriented development paradigm is lacking the idea of separation of concerns [2]. Separation of concerns deals with separating the functional properties of the system from its non-functional properties. Aspect oriented software development is an emerging new programming paradigm. In aspect oriented software development we use the notion of separation of the concerns. There are two major types of concerns when we develop a system; the core concerns and the crosscutting concerns. Core concerns are the functionality provided by the system to be developed. Crosscutting concerns are the concerns like performance, memory management etc.

Within aspect oriented development paradigm, there are several programming and modeling facilities available to the developers. There are several programming languages available supporting aspects, e.g., AspectJ [3], AspectC++, Aspect Oriented C (AspeCt C Development Team, 2007) etc.

There is also extensive support of modeling available in aspect oriented development paradigm. In aspect oriented development paradigm there are pointcuts, join points, advices, introductions and aspect. A pointcut is a collection of join points. Join point is a well defined point to interact with base classes for example method execution, object initialization, field get or set etc. Advice is a piece of code that executes for the join point for it is defined. Three types of advices are before, after and around. The piece of code in the advices is executed according to the type of advice and the join point on which they are defined. These are encapsulated in aspects. Aspect can be abstract or inherited.

When we use modeling it provides us the structure for solving the problem, knowledge to find out the multiple solutions, give abstractions to handle complexity, reduce time-to-market for business problem solutions, reduces the development costs, and handle the risk of errors (UML summery V1.1, 1997, UML syntax and Semantics V1.1, 1997). Benefit of modeling in the aspect oriented development is that aspects are identified at the early stage then they are more reusable and it is also make possible of automated code generation for AOP. Modeling aspect oriented programs provide the better understanding of the system. Check the behavior of the aspect after they are woven into the base class behavioral models is often used [4].

In this research paper, we present an approach for modeling aspect oriented systems. The main contribution of this research paper is to provide a meta-model for aspect oriented modeling which will show the static and behavioral structure of the aspect.

We propose a Meta modeling notation which is for aspect modeling. We also define how the weaving of aspect and base classes will be done. We define how the joinpoints are modeled so that they can be weaved the aspect and base class. This extension for aspects will help in representation of how the aspect will interact with the system.

2. Related Work

In this section, we discuss the literature survey of modeling of aspect oriented constructs. We surveyed different techniques for the modeling of the aspect oriented programs. We will do analysis of the proposed modeling work on different parameters. We focus on the techniques of the Meta model extension that's why our survey has some limitation. We include the literature which is about the extension of the Meta model either it is a profile or an MOF based.

Joerg Evermann presented the Meta level specification and profile for AspectJ in UML. They modeled the aspect as a Meta class because of the characteristics of the aspect. They make an aspect Meta class a stereotype on the UML class construct. They modeled the advice by the extension of Behavioral Feature. They also put constraints that the stereotype "Advice" can only be associated with the behavioral features of the classes stereotype "Aspects". There extension for the static crosscutting features is from behavior that is the superclass for both property and Operation. They also put constraints that the stereotype "Static Cross Cutting Feature" stereotype can only be associated with the behavioral features of the classes stereotype "Aspects". They extend the Meta class for pointcut from the Meta class structural features. They also put constraints that the stereotype "Pointcut" can only be associated with the behavioral features of the classes stereotype "Aspects". In their profile they define the stereotype of join points and the stereotype of pointcut for every type of pointcut. Their profile is very helpful for the modeling of aspect oriented programs because it provides a very detail view for the modeling of the aspects. They use the UML 2.0 for extension.

Mosefaoui and Vachon proposed an approach in which aspects are modeled using the aspect-UML. They also define the pointcuts as class diagrams. They model the aspects oriented programs with two types of models one show the static view and other shows the dynamic view. In static view the technique defines the aspects and base classes in class diagrams. They made a relationship of crosscut between base class and the pointcut class. They relate the pointcut with the aspect with the type of advice which will be executed on matching the required joinpoint which is contained by the pointcut. In dynamic view they use sequence diagrams in which they show the interaction between aspect and base classes after weaving [2].

Heidenreich *et al.* proposed an approach for the modeling of aspect oriented programs using the fragment quires. Their approach applies to aspect-oriented modeling using two UML class diagrams. The first is the core model represents the core, into which the second is the advice model which shall be woven. They define the pointcuts over the core, they apply a fragment query. The anchors in the advice model are then bound to slots and hooks in the core model. Additionally, the bound advice fragments have to be conFig Nouring with core information. Thus, certain anchors in the core model are bound to slots in the advice model [5].

A. Zakaria *et al.* proposed an approach which provides an UML extension for the modeling of the aspect oriented systems. They proposed aspects as class in aspect oriented modeling with the stereotype aspect. They represented the relationship b between the aspects and the class by the UML association relationship. Aspect must be associated to the one or more base class in order to make an impact on the system. They make different types of aspects as the affect the base classes. They proposed different types of tags for the relationship between the classes and aspect these tags are according to the affect of the aspect on the system. They modeled the pointcuts as a stereotype of UML model element class. They associate these pointcuts with the aspect class as a "has pointcut" relationship. They also gave the relationship between aspects. If an aspect has the higher precedence then the other then they give the name of relationship between them as a "dominates". For advices they use bases UML Meta model element operation and they use the stereotype with the same name as the type of advice [6].

Technique proposed by Meier *et al*. presents the model the crosscutting concerns in Adora. Adora is a modeling language. Adora is basically an object oriented modeling language. They provide the extension of Adora for aspect oriented programs. Their aspect oriented extension of ADORA is based on three concepts: Aspect containers represent modules of crosscutting concerns and comprise a description of crosscutting elements, such as behavior and scenarios, join relationships denote explicitly where the crosscutting concerns affect other concerns and view mechanisms provide abstraction for aspect oriented ADORA models. They also define a weaving mechanism for aspects and base classes [7].

An approach by J. Whittle and Jayaraman which presents a Modeling Aspects Using a Transformation Approach (MATA), a UML aspect oriented tool that uses graph transformation to specify and compose aspects. In MATA aspects are graph rules. They use the technique of the critical pair analysis (CPA) to find the dependencies and conflicts between rules. CPA checks the rules in pair if one rule needs model elements which are introduced by another element then there is a dependency and there is a conflict if one rule modifies the base in this condition the other rule is no longer applied [8].

Technique by Klein and Kienzle which presents a purpose for specifying reusable aspect models that defines structure and behavior of aspects. For structural modeling they use the class diagrams and for behavioral modeling they use the sequence diagram. They model the aspect in a package. In this package they model aspect as class diagram. They model pointcut and advice as sequence diagrams [9].

Approach by Mahoney and Ellard presented a technique to model the aspects with sequences charts. They use live sequence charts to model the aspects and base classes. In a live sequence chart the pre chart is for pointcut designator and the main chart is for advice. They show the interaction between base class and the aspect with these sequence diagrams. They model the aspects and base classes as separate sequence diagrams and then combine them. They use live sequence chart to model the interaction. They use play engine to see the impact of the aspects on the base class. They play engine plays the live sequence charts. They modeled the weaving mechanism through state charts [10]. This approach presents a technique to model the aspect with the Aspect Interaction Charts. The idea is same as it was in previous work but in this approach they provide some advance features like what scenario could not happen in the main chart which they called forbidden scenario [10]. In this they also introduce a new message event in aspect interaction chart which they name as before method event which is used to capture the before method call joinpoint [11].

An approach by M. Kende presents how we can model the aspect oriented programs. They made two types of models one is aspect design model and the second is config Nouration model. In first model they identify the connection point form where aspects interact with the objects. In the conFig Nouration model they combine the aspect intense with the interaction component. This model shows the before and after weaving process. After weaving shows additional features introduced by the aspect in the component [12].

Aldawud presented an approach which presents a UML profile for the aspect oriented programs. In their approach they define stereotype for the aspect, pointcuts, joinpoints and advices and for the relationship they define the control stereotype. They present this idea at an abstract level [13].

"A Metamodel for Aspect-Oriented Modeling" technique was proposed by Chavez and Lucena. In this approach there is a UML Meta model for aspect oriented programs. They describe base elements which will take part into the crosscutting relationship. Crosscutting elements are the model elements which take part in crosscutting relationship. Crosscutting relationship describes the structural and behavioral increment which is introduced by the crosscutting elements into the base elements. Aspects are model elements which are combination of local features and the crosscutting interfaces. Crosscutting interfaces are the points at which the aspect crosscut the base class. The crosscutting feature model elements that describe the features that will be combined with the one or more elements of base class [14].

D. Xu, and X. Dianxiang proposed works on aspects are modeled using the state machine. This state machine shows the aspect after the weaving process how an aspect can introduce new states and new transactions in the base class state machine. They use these state machines for testing the aspect oriented programs [15].

"Towards aspect oriented UML executable models" in this paper a technique is proposed to make executable aspect oriented UML models. They first made the executable UML model for the base class. They model the aspects with the aspect oriented executable model profile. To view how the crosscutting concerns must be composed with others concerns they made a joinpoint model. They composed the aspect and base model which they called is a woven model. This model is a common UML executable model. They then use UML tool having executable capabilities to execute the woven model [16].

Cottenier *et al*. presented a technique for modeling the aspect oriented composition. They use Specification and Description language for the modeling and weaving of the aspect oriented programs. They also presented a SDL

Meta model for aspect oriented programs. This Meta model consists for different profiles and Meta models. They define the profile for aspect beans with the name of aspect bean profile which shows how the aspect SDL weaver sees the aspect Bean SDL state charts. They define a profile for the join points this profile identify the elements which can be used as join points in the SDL state charts. They define a connector Meta model which is used to identify the join points in the core model and the advices in the aspect bean. This is used in the first phase of the weaving process. They defined a weaver behavioral Meta model which is used in second phase of weaving for the identification and binding of the join points and the advices [11].

M. Lion proposed technique in which a metamodel for the aspect oriented programming is proposed. They used a tool OpenTool/UML for the extension of the UML. They used they mechanism provided by the OpenTool/UML for the extension of the UML metamodels. They basically discuss how the OpenTool can be used for the extension of the UML. They use aspect oriented programming as their case study for the extension of the UML. They proposed a UML AOP extension Meta-Model [17]. In given below **Table 1** we have mentioned comparison of different techniques on few research parameters and compared them to analyze.

A technique by Y. Han presented a metamodel for the aspect oriented programs. They first introduce the metamodel for the java. They basically present the metamodel for the AspectJ so they use java Meta model for the extension for the aspect. Java Meta model shows the static structure of the java language. They extend that java Meta model for the AspectJ. They also gave graphical notation for the AspectJ and crosscutting elements [18].

3. Proposed Approach

This paper is about using the behavioral model. There is no Meta model presented to use the behavioral model for aspects. So we introduce Meta models for the aspects in this paper so that we can use the behavioral models for the aspects. This is the reason it looks more like a Meta-model paper. We have to make Meta models to show the aspects behavior. The main emphasis in on the behavioral model for the aspects but we have to make the Meta model to verify that we can use behavioral models for aspects.

We proposed an approach for modeling of aspect oriented programs on the basis of the literature review which is about behavioral modeling of the aspect oriented programs. In the literature most of the modeling is language specific. No one has modeled the behavioral structure of the aspect before weaving. We proposed a metamodel for the modeling of the aspects, joinpoints, pointcuts and advices. This proposed Meta model is the extension of the UML Meta model. No specific tool for modeling is needed because it will use all the standard notations of the UML.

3.1. Overview of Our Proposed Approach

We suggested a Meta model for aspect oriented programs in which we proposed a Meta model extension for aspect oriented programs. We used MOF (Meta Object Facility) which has the heavyweight extensibility mechanism in its specification. Because in this mechanism we can make any Meta model the restrictions of profile are not applied on MOF. In this Meta model we modeled the aspect concepts in a Meta model. Our proposed approach not only models the static structure of the aspect but also models the behavioral structure of the model.

In our above Meta model in **Figure 1** we made the static structure of the aspect. Aspect encapsulates the advices, pointcuts and intertype declarations. Aspect can also have inner classes but those classes are abstract. Above Meta model also show the association of the aspect with the pointcut and advices and pointcuts association with the joinpoints as in **Figure 2**.

Three types are before, after and around advice. Before advice execute on meeting the pointcut expression before that piece of code which is crosscut by the pointcut. "After" advice is executed after the piece of code is crosscut by the pointcut. Around advice can be used on meeting the required criteria of the pointcut expression it

Table 1. Comparison of different related technique.

	Joerg Evermann	Mostefaoui. F and Vachon. J	Mahoney. M and Elrad. T	Cotternier. T *et al.*	Chavez. C and Lucena. C	Lion. J *et al.*	Aldawud. O *et al.*	Han. Y *et al.*
Proposed work	UML Profile	UML Profile	Sequence Charts	Framework	MOF Based	MOF Based	UML Profile	MOF Based
UML version used	UML2.0	UML 2.0	NA	UML 2.0	UML 1.4	UML 1.4	NO	UML 2.0
Language specific	Yes	NO	NO	NO	NO	No	NO	Yes
Structural model	Yes	Yes	NO	Yes	Yes	YES	Yes	Yes
Behavioral Model	No	No	Yes	Yes	No	No	NO	Yes

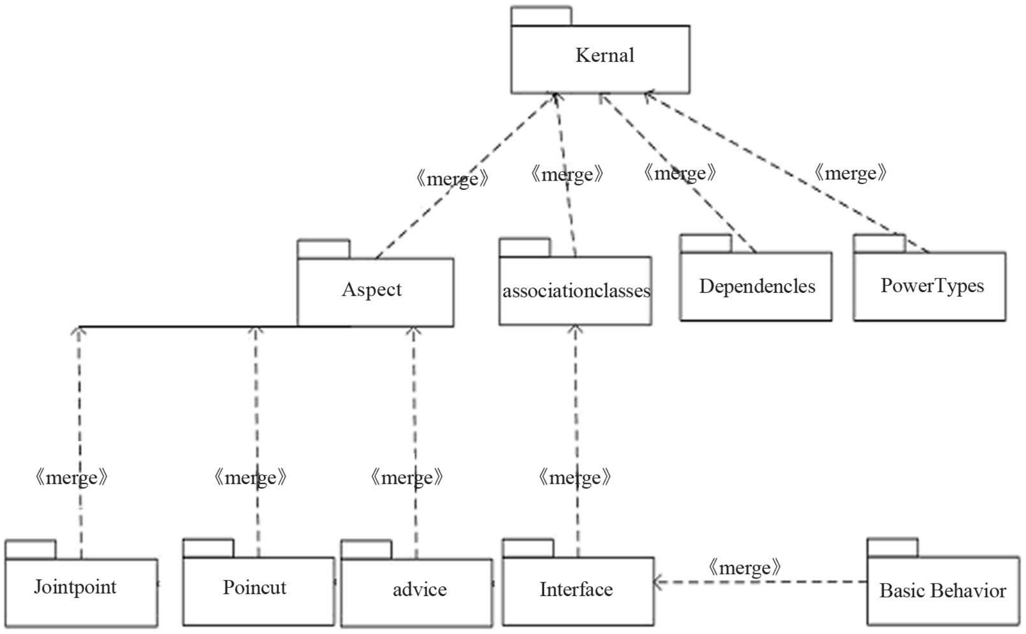

Figure 1. Aspect meta model.

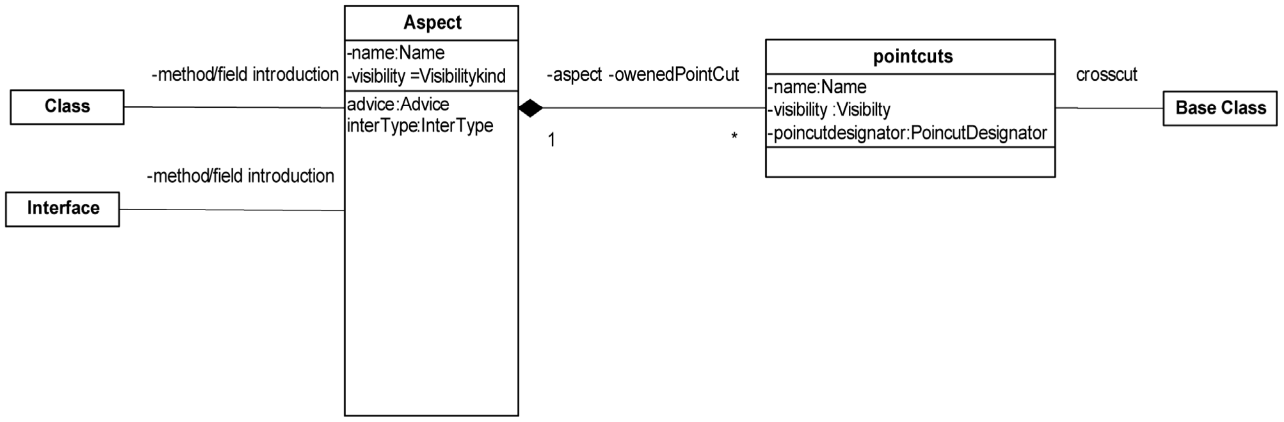

Figure 2. Aspect and base classes Meta model.

can change the flow of the execution of the programs. Intertype declarations are the class variables that can be initialized by the aspects and the method of the classes that can be declared inside the aspects.

In our modeling technique we make a Meta class for the pointcuts. Pointcut contains all the join points. Joinpoints are expressions which define the points where they crosscut the base class. Advices are piece of code that should execute when the join point expression is met. Advices are associated to the pointcuts. Three types of advices are there and they execute according to their type.

3.1.1. Pointcut

A pointcut is a construct designed to identify join points and obtain the context surrounding the join point. The pointcut is more than just a container for join points. It directly shows how a concern will crosscut the base class. Pointcut may also have some parameters. These parameters are used by the advices associated to the particular pointcut [19].

In **Figure 3** we make a Meta class for the pointcut. In a pointcut there may be more than one pointcut designators. Pointcut has a specific name and the visibility. Name attribute of the pointcut is for the name of the pointcut. Pointcut can also have parameter. These parameters are used by the advices which are associated with these pointcuts. It should be the name given to the pointcut any string value.

Figure 4 shows the Meta class for the pointcut designator. These designators are used to capture the joinpoints. The designator may be an execution which matches execution of a method or constructor. It may be a call which matches calls to a method or constructor.

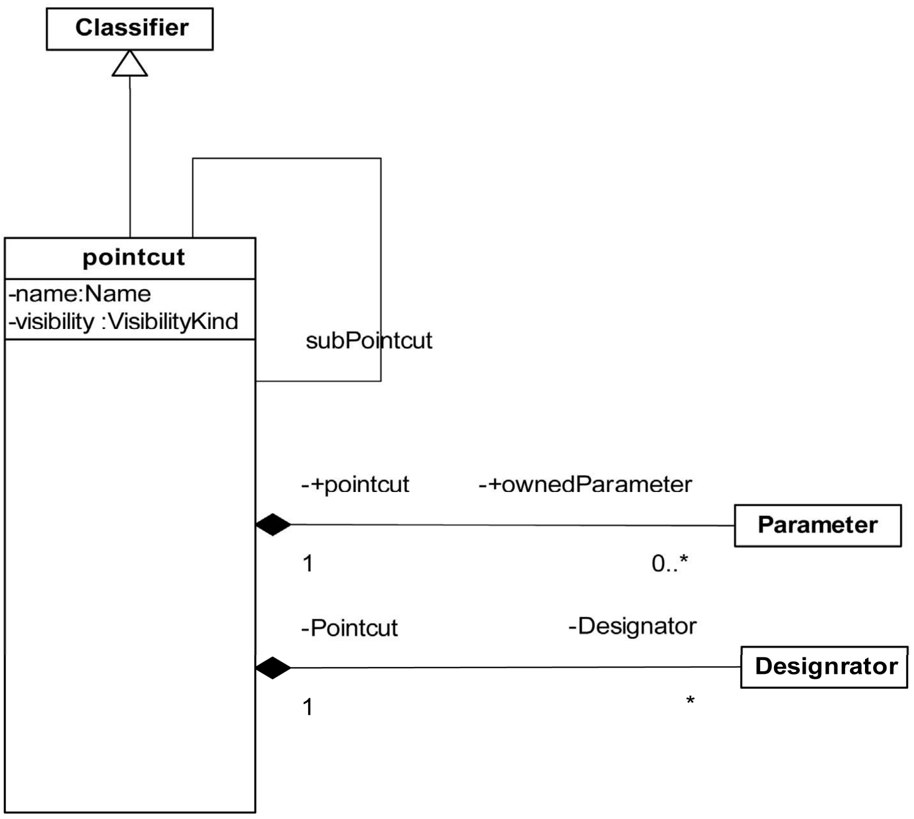

Figure 3. Point cut Meta class.

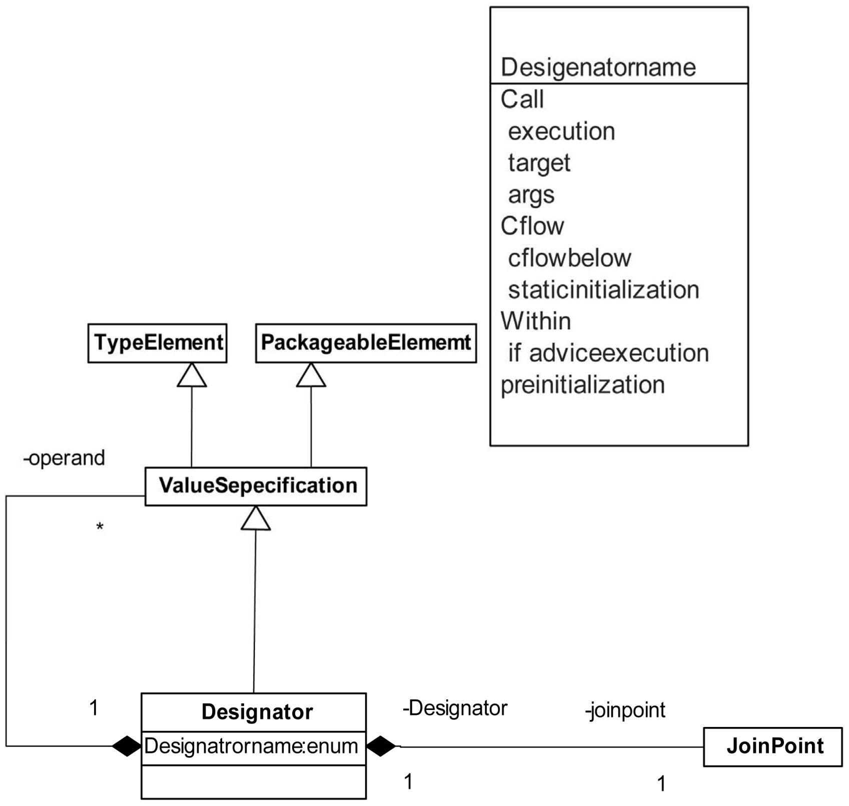

Figure 4. Pointcut designator Meta calss.

It may be an initialization which matches execution of the first constructor to a class. It may be a handler which matches exceptions. It may be a get designator which matches the reference to a class attribute. It may be a set designator which matches the assignment of a class attributes [20].

This designator which returns the object associated with a particular join point or limits the scope of a join point by using a class type. The target designator which returns the target object of a join pointcuts limits the scope of a join point. The args designator which exposes the arguments to a join point or limits the scope of the pointcut. The cflow designator which returns join points in the execution flow of another join point. The cflow below designator which returns join points in the execution flow of another join point but not including the current join point. The static initialization designator which matches the execution of a class's static initialization code. The within code designator which matches join points within a method or constructor. The within designator which matches join points within a specific type. The if designator which allows a dynamic condition to be part of a pointcut. The advice execution designator which matches on advice joins points. The pre initialization designator which matches pre initialization join points [19,20].

3.1.2. Joinpoint

Joinpoints are the points where the aspects crosscut the base classes. Joinpoint is any execution point in the class. Joinpoint may be a method signature or a field signature or an exception type. These joinpoints are picked by the pointcut designator.

Figure 5 shows the Meta class for the join point. As joinpoint may be any method signature, any exception type or any data type signature. So we inherit this class form the Meta classes of operation, type and datatype [20].

3.1.3. Advice

We made a Meta class for advices that is extension for operation diagram (OMG, UML, Superstructure, and V2.1.2). We have modeled the advices through activity diagrams. In which we have show that how advices are executed according the pointcut to which it is associated.

In **Figure 6** given below advice shows the behavior of

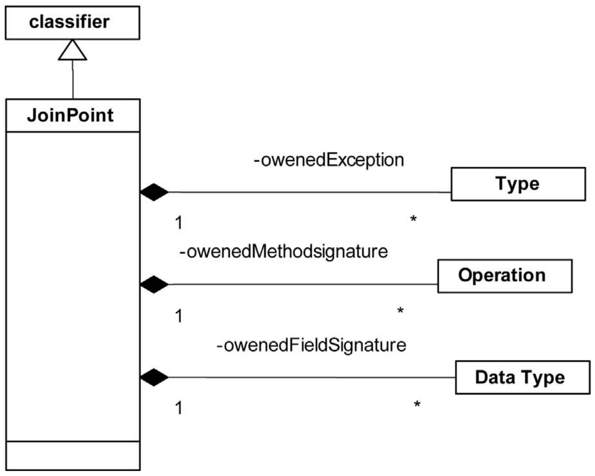

Figure 5. Joinpoint meta class.

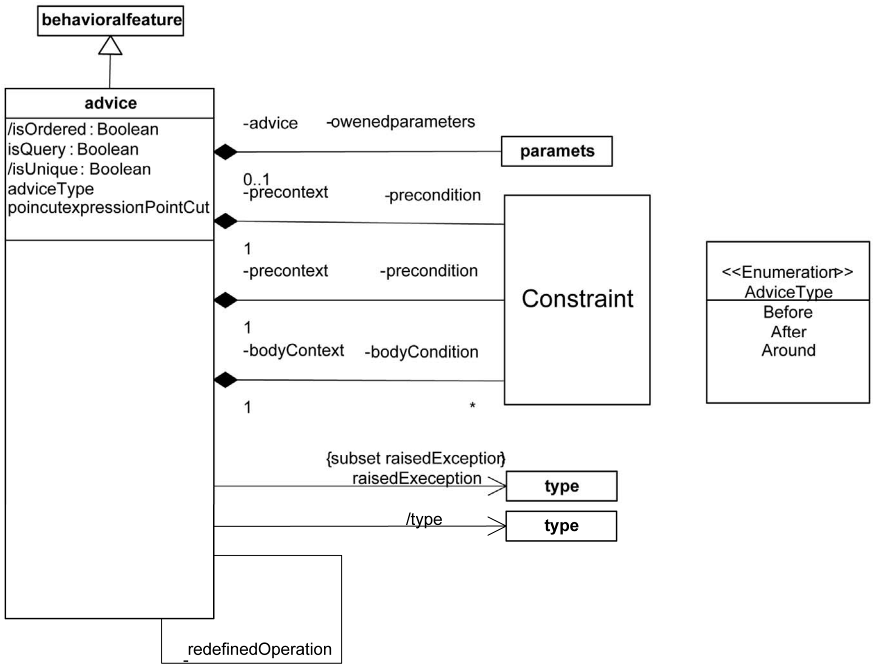

Figure 6. The advices meta class.

the aspect. Advices are like operation in the classes. But like methods in class the advices have no identifier which is logical because we don't have to refer the advice in our code it will execute according to the pointcut, to which it is associated. We extend the metaclass for the operation to make the metaclass for the advice. Advices may have some parameters. They have the pointcut expression which defines that on which pointcut this advice should be executed. Redefined operations are those which are redefined by the advices. The advice must have an advice type which defines that when the advice should be executed according to the pointcut expression before, after or around. Before advice will execute before the joinpoint which is referred by the pointcut [20].

We modeled the advices through the activity diagram. The activity diagram will show how the advice will be executed according to the pointcut which is associated to it. Advices are actions performed against the pointcut to which these are associated so we modeled them through activity diagram. We show that advices are activities performed according to the pointcut associated to them. An activity is a behavior and Basic Behavior is type of Behavior has a association with the behavioral feature. Advice is a behavioral feature so an activity can specify an advice.

3.1.4. Aspect

Aspect encapsulates the pointcuts, advices and intertypes. Aspect like a class can have its own fields and methods so we extend the Meta class of the aspect form the Meta class for class. Aspect can be inherited from the abstract aspect. An aspect is abstract if any pointcut or method in the aspect is abstract [20].We can inherit aspect from another aspect as we can do inheritance in the classes. The sub aspect can use the pointcuts of the super aspect.

The given below **Figure 7** shows the Meta class for the aspect. Aspect encapsulates the advices and the pointcuts. Intertype is a special relationship between base classes and interfaces. Intertype is a method or property of the class or of the interface. Which will be weaved in to the base classes and interface by the aspect [21].

Our package is merged into the Kernal package in **Figure 8** because our all Meta classes are extension of the Meta classes in the kernel package. So this aspect package will be merged into the kernel package [20].

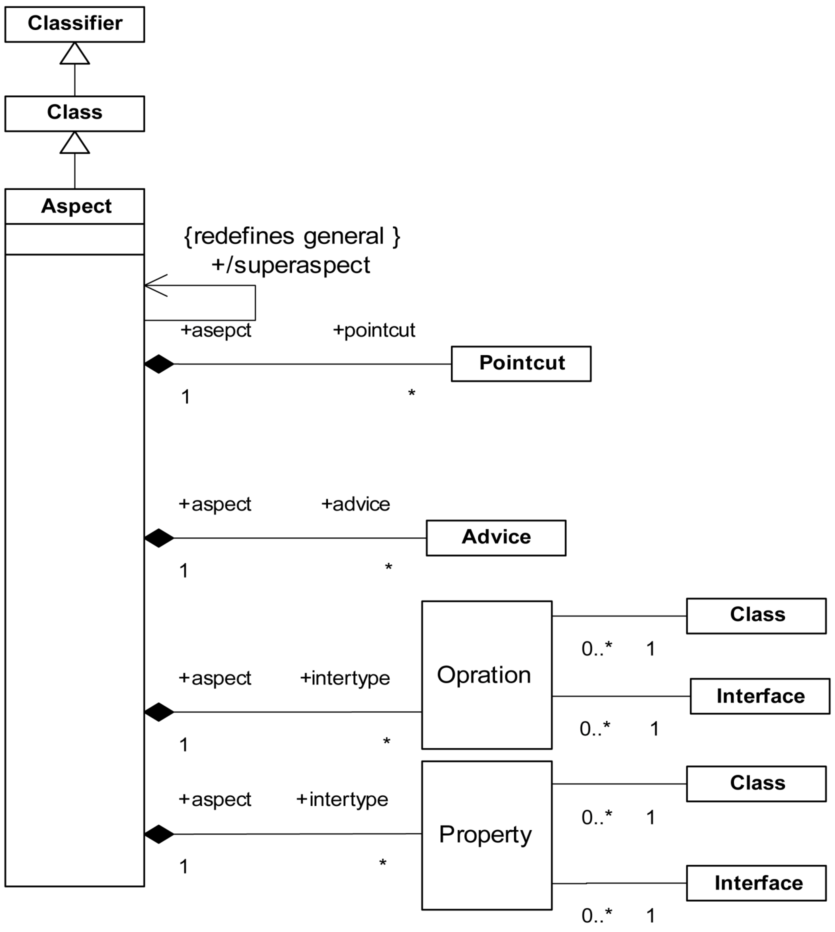

Figure 7. Aspect Meta class.

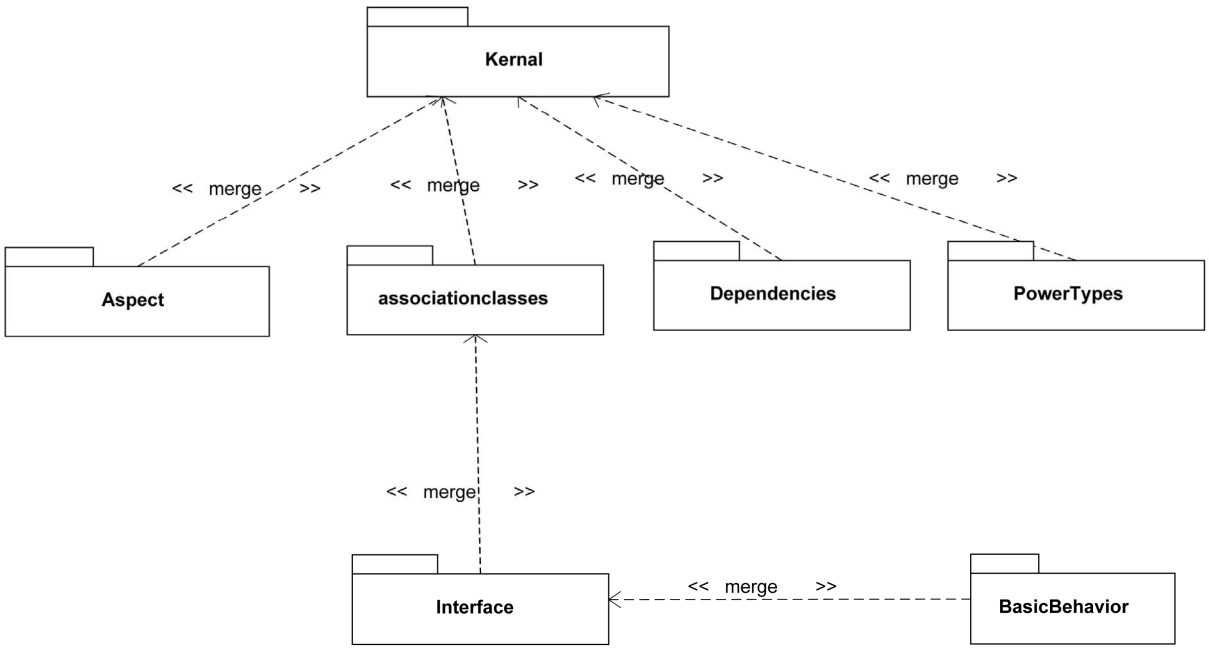

Figure 8. Aspect model.

Figure 9 Shows state diagram for purchasing requisition as mentioned in our case study.

4. Results and Discussion

Different modeling techniques are presented to model the aspects and to show the weaving mechanism. The proposed techniques are either language specific or not the extension of the standard UML Meta models. We extend the UML Meta model to model the aspects. We presented how we can model the static structure and behavioral structure of the aspect. In order to verify the technique discussed above now we model the aspects using those Meta models.

The proposed study shows here how we can model the aspects by using the Meta model extension presented above. This modeling technique is an extension to the latest UML Meta model and based on the UML notation so it is easy to understand and easy to implement for the developers. Because this technique base on the UML so it is also helpful in order to narrow down the gap between the aspects oriented programming and the object oriented programming.

5. Case Study

In this chapter we will use the case study in which we will cover almost every aspect of the aspect oriented programming. We take a real life example case study so that it is easy to understand and can explain the approach proposed. This case study is about banking system in which has the power to create new customers, add accounts to customers, mark certain accounts as overdraft

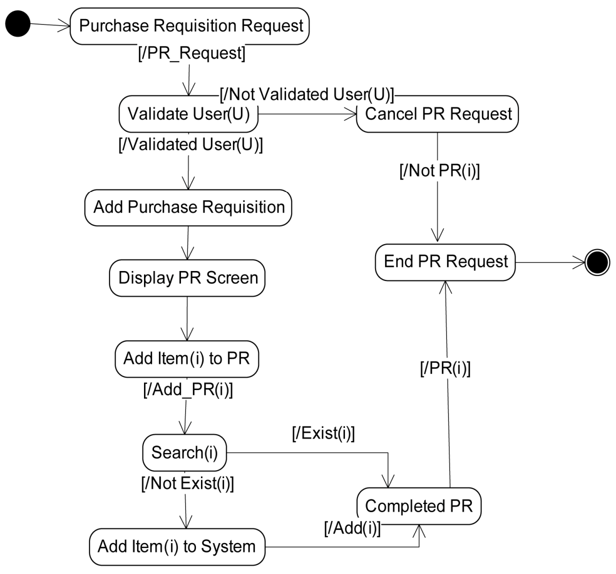

Figure 9. State diagram for purchase requisition.

accounts, and make transactions in the accounts. This also has a facility of a check clearance system. The customer class models a banking customer and is associated with a set of accounts that belong to a customer and a subset of those accounts marked as overdraft accounts. Overdraft accounts are designated to automatically transfer money to the checking account when a check clearance system detects an insufficient balance in that checking account.

In the given below **Figure 10** we modeled the aspects and their relationship among the each other. First we have an abstract aspect with the name of Abstract Debit

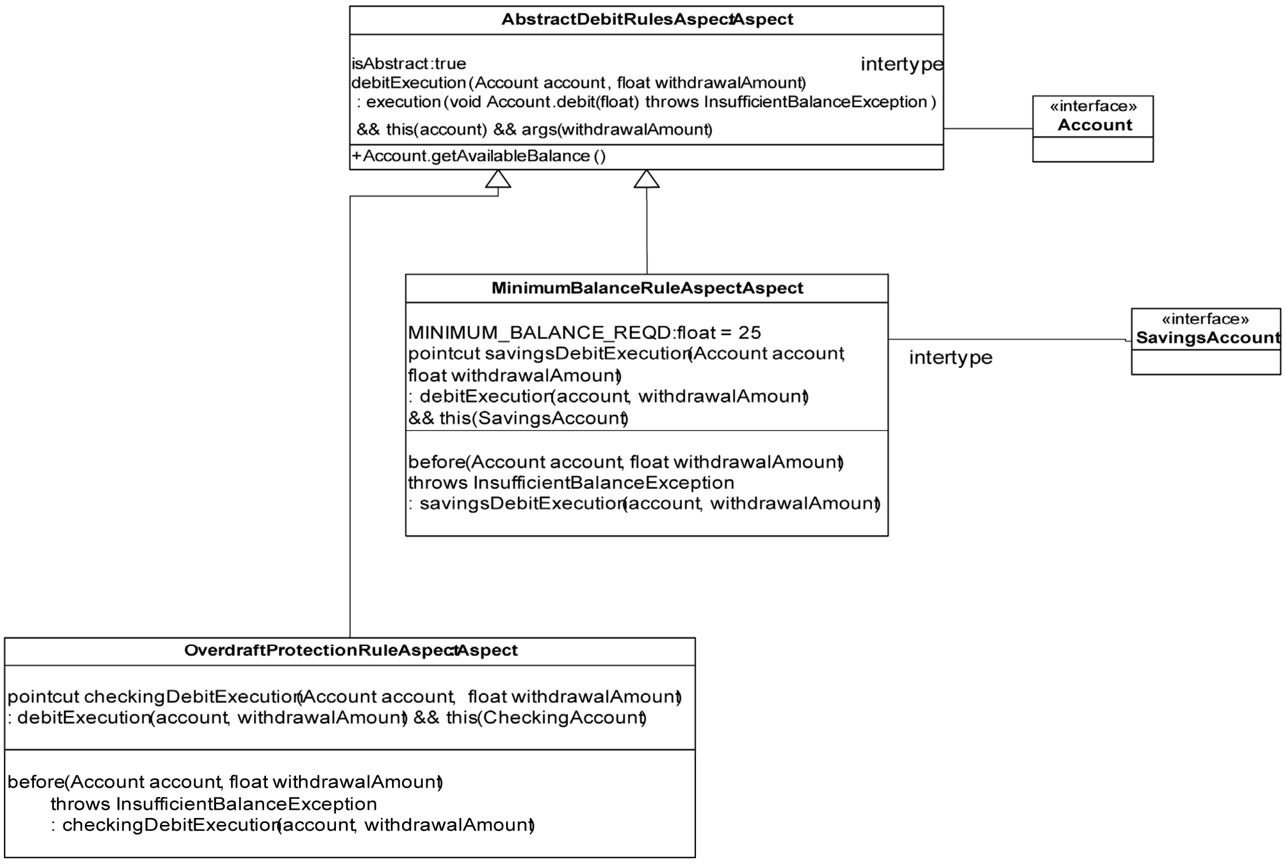
AbstractDebitRulesAspectAspect
isAbstract:true
debitExecution(Account account, float withdrawalAmount)
: execution(void Account.debit(float) throws InsufficientBalanceException)
&& this(account) && args(withdrawalAmount)
+Account.getAvailableBalance()
intertype
«interface» Account
MinimumBalanceRuleAspectAspect
MINIMUM_BALANCE_REQD:float = 25
pointcut savingsDebitExecution(Account account, float withdrawalAmount)
: debitExecution(account, withdrawalAmount)
&& this(SavingsAccount)
before(Account account, float withdrawalAmount)
throws InsufficientBalanceException
: savingsDebitExecution(account, withdrawalAmount)
intertype
«interface» SavingsAccount
OverdraftProtectionRuleAspectAspect
pointcut checkingDebitExecution(Account account, float withdrawalAmount)
: debitExecution(account, withdrawalAmount) && this(CheckingAccount)
before(Account account, float withdrawalAmount)
throws InsufficientBalanceException
: checkingDebitExecution(account, withdrawalAmount)

Figure 10. Aspect static structure.

Rules Aspect that has the pointcut with the name of the debit Execution which is an abstract pointcut because it does not have the advice associated to it. This aspect also has an intertype declaration for the interface Account. It introduces a method with the name of get Available Balance in the interface account. The debit Execution poincut in this aspect has two pointcut designators execution, args and this. The pointcut debit Execution also has two parameters one is account and other is with drawal amount. "Minimum Balance Rule Aspect" aspect is inherited from the "Abstract Debit Rules Aspect" aspect. This aspect has the poincut savings Debi Execution which use the debit Execution poincut of the parent aspect and has one poincut designator this. This aspect has an advice of before type which will be executed before any poincut to whom it is associated. This aspect also has an intertype declaration for the interface saving account. This is an introduction of the field in the interface savingaccount with the name MINIMUM_BALANCE_REQD of data type float.

"Overdraft Protection Rule Aspect" aspect has two pointcuts and one before type advice. Check Clearance Transaction pointcut used the pointcut designator execution. Checking Debit Execution designator uses the debit Execution pointcut of the base class and this poincut designator. This aspect has a local method with the name perform overdraft protection these.

The advices are modeled through activity diagram in **Figure 11** to show the behavior of the aspect. In our Meta class of advice we inherit the adive from the behavioral feature.

The advices are modeled through activity diagram to show the behavior of the aspect. In our Meta class of advice we inherit the advice from the behavioral feature [21]. Which is a type of behavior and we can model a behavior through activity diagram so we modeled the advices through the activity diagram following activity diagrams show the advice how these advices were executed according to the associated pointcuts.

This advice is associated with checking Debit Execution pointcut of the aspect Overdraft Protection Rule Aspect. This activity diagram shows that advice will be executed when the pointcut will match and this advice will check the protection that withdrawal amount will not be more than the actual balance.

Figure 12 shows an advice which is associated with saving Debit Execution pointcut of the aspect Minimum Balance Rule Aspect. This advice will through an exception, if the withdrawal amount in more than the actual amount in the account.

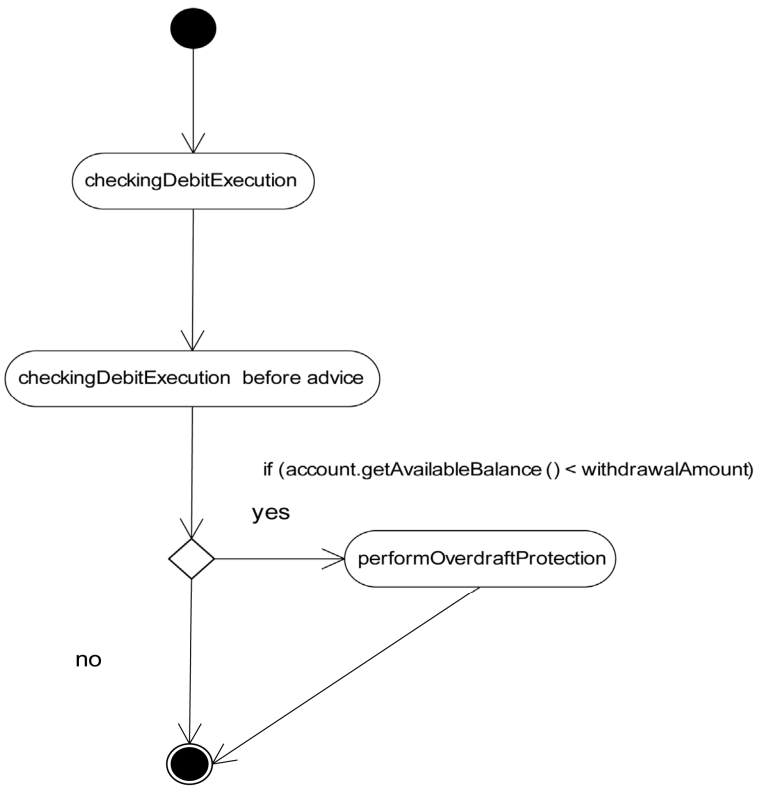
checkingDebitExecution
checkingDebitExecution before advice
if (account.getAvailableBalance () < withdrawalAmount)
yes
performOverdraftProtection
no

Figure 11. Activity diagram of advice.

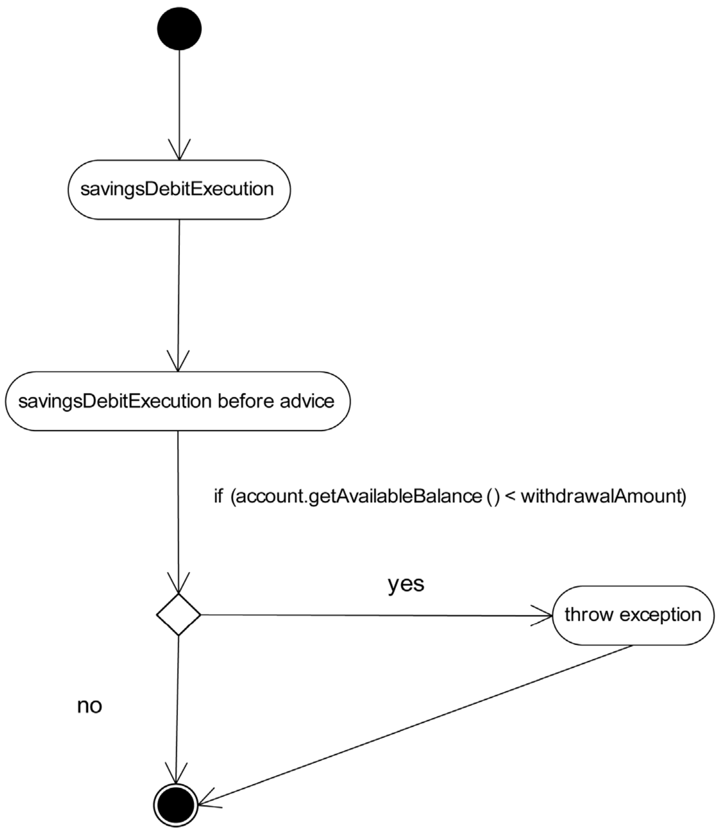
savingsDebitExecution
savingsDebitExecution before advice
if (account.getAvailableBalance () < withdrawalAmount)
yes
throw exception
no

Figure 12. Activity diagram of advice.

The below given **Figure 13** shows the classes diagram of our banking system. In **Figure 13** there are three interface Account, saving Account and Checking Account. Account Simple Impl, Savings Account Simple Impl, Checking Account Simple Impl, Customer and Check Clwerances System are the bases classes used in this banking system example. We made the sequence diagram of the base classes and show the interaction of the advices with them through the interaction overview diagram. Inter action overview diagram is used to show the cooperation between the interaction. Activity and sequence diagram shows the interaction and we combine these two interaction diagrams to show the interaction between advices and the base classes [21].

The above **Figure 14** shows the sequence diagram of the banking system of our case study and **Figure 15** shows the interaction overview diagram of the advices and base classes. The above two nodes represents the activity diagrams of the advices. These two advices will interact to the base classes according to the pointcut designators which contains the join points where these advices will be executed.

We can see that how advices can interact with the base classes. The advices will be executed according to the pointcuts they are associated. Whenever a joinpoint will match the point in the base class the advice will be executed if a before advice as it is in our case it will be executed before that point.

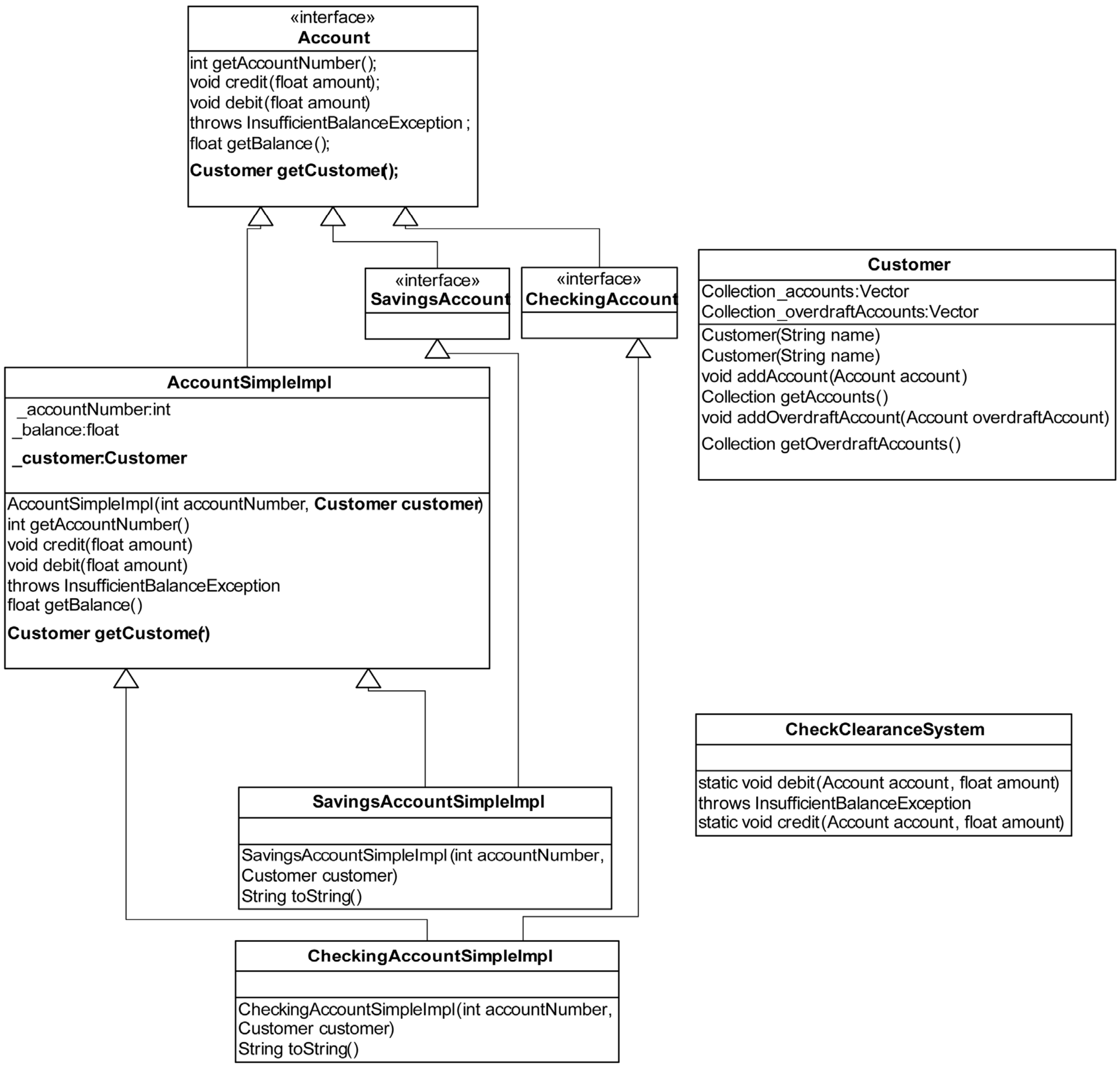
«interface»
Account
int getAccountNumber();
void credit(float amount);
void debit(float amount)
throws InsufficientBalanceException;
float getBalance();
Customer getCustomer();
«interface»
SavingsAccount
«interface»
CheckingAccount
Customer
Collection_accounts:Vector
Collection_overdraftAccounts:Vector
Customer(String name)
Customer(String name)
void addAccount(Account account)
Collection getAccounts()
void addOverdraftAccount(Account overdraftAccount)
Collection getOverdraftAccounts()
AccountSimpleImpl
_accountNumber:int
_balance:float
_customer:Customer
AccountSimpleImpl(int accountNumber, Customer customer)
int getAccountNumber()
void credit(float amount)
void debit(float amount)
throws InsufficientBalanceException
float getBalance()
Customer getCustomer()
CheckClearanceSystem
static void debit(Account account, float amount)
throws InsufficientBalanceException
static void credit(Account account, float amount)
SavingsAccountSimpleImpl
SavingsAccountSimpleImpl(int accountNumber,
Customer customer)
String toString()
CheckingAccountSimpleImpl
CheckingAccountSimpleImpl(int accountNumber,
Customer customer)
String toString()

Figure 13. Base classes.

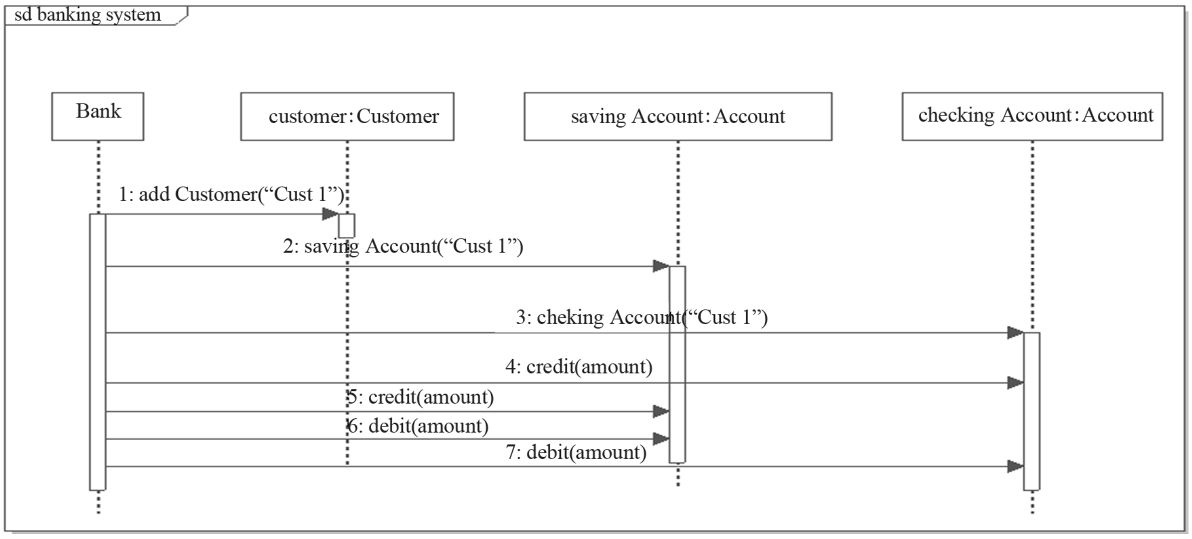

Figure 14. Banking system sequence diagram.

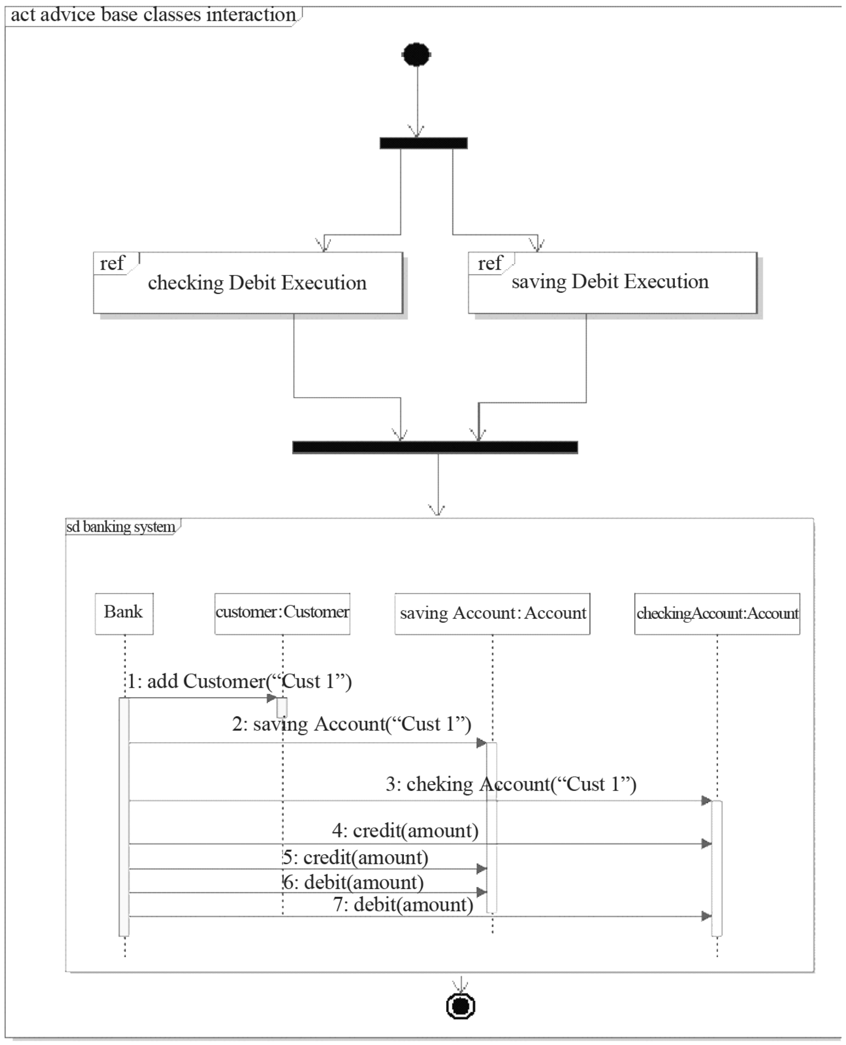

Figure 15. Interaction overview diagram.

5.1. Discussion

In our proposed approach we model the aspect. We show aspects static and behavioral structure. How we can model the aspect before weaving. We can model the behavioral structure of the aspect before weaving which is helpful in testing the operation performed by the advice. To show the complete functionality of the advice it must be weaved with the base class but to show the operation performed by advice we can model it in sequence diagram which will give the better understanding of the system.

5.2. Test Goal Generation through Scenarios

The advantage of applying guards at the scenario enables to generate the test cases also referred to as test goals. These test goals capture the flow of events for the use case scenario. As the test goals are based on contracts so that can be formalized as logical expression.

5.3. Contractual State Chart

State diagrams represent the object behavior with invocation of event “represent operation” and are used to record different states with events that can cause a state transition. A state machine is composed of state representing the behavior of the system on certain input whereas transition may result in an output action, event “an input” and action the output result [17]. State diagram annotation with guards “Guards are associated with pre and post conditions” enables to specify the entry and exit conditions. Optional Guards can be added to states and transition may be annotated with guard, event, and action. If there is no guard or both guards are true then the exit action is performed. Test cases are imposed to verify the behavior of the system when applied on the state chart.

We had implemented a tool that takes XML containing guards of scenario as input and generate test path expressing test cases as logical expression.

5.4. Limitations of Proposed Approach

Our proposed approach helps to get elaboration of aspect details. We have shown the aspect’s static and behavioral structural model. It covers the structural and behavioral features of the aspect. We have not modeled the behavior of the aspect after weaving but we have shown the static structure of weaved model. Our metamodel shows how the aspect will model or behave before weaving. It is an extension of metamodel given by UML metamodel.

6. Conclusions

This approach is helpful in the better understanding of the aspect. In this approach as we have shown how we can model the aspect’s static and behavioral structure. This Meta model is also helpful to make a better aspect because we can clearly show the static structure of the aspect which shows what we can encapsulate in the aspect. We can show the behavioral structure of the aspect which shows what operation advice should perform.

The proposed approach in this paper is a Meta model for the aspect oriented programs. The idea is to give a Meta model for the aspect oriented program which is the extension of the UML Meta model which is a standard language for the modeling. This basic idea is to provide a Meta model for the modeling of the aspect oriented programs. Our modeling approach covered the structural and behavioral features of the aspects. We emphasize on the modeling of aspect not on the weaving mechanism. We cover both the structural and behavioral features of the aspects. Pointcuts and the advices both are modeled. Advices are like operation so we modeled them in sequence diagram which show what operation this advice will perform. This will show the operation of the advice but not impact of the advice on the base class. This is helpful for the testing of the advice operation in separate before weaving. We can easily check either advice is performing the correct operation or not to check it performs operation at the correct time or not it must be weaved into the base class.

Our Meta model shows how the aspect will model before weaving. The static structure of the aspect is modeled as well as the behavioral structure is also modeled which shows the behavior of the aspect. We don’t model the behavior of the aspect after weaving but we show the static structure of weaved model.

REFERENCES

[1] S. Clark and E. Baniassad, “Aspect-Oriented Analysis and Design: The Theme Approach,” Addisn-Wesley Professional, Boston, 2005.

[2] J. Vachon and F. Mostefaoui, “Achieving Supplementary Requirements Using Aspect-Oriented Development,” *In ICEIS*, Vol. 3, No. 4, 2004, pp. 584-587.

[3] J. Ivar and W. G. Pan, “Aspect Oriented Software Development with Use Cases,” 2004.

[4] F. Lidia and P. Sanchez, “Towards Executable Aspect-Oriented UML Models,” *Proceedings of the* 7*th Workshop on Aspect-Oriented Modelling* (*AOM*), Vancouver, 2007, pp. 28-34.

[5] F. Heidenreich, J. Johannes and S. Zschaler, *Proceedings of* 11*th International Workshop on Aspect-Oriented Modeling Co-Located with Models Nashville*, 2007.

[6] A. A. Zakari and H. Hosny, “A UML Extension for Modeling Aspect-Oriented Systems,” 2002.

[7] S. Meier, T. Reinhard, R. Stoiber and M. Glinz, “Zurich in Aspect-Oriented Requirements Engineering and Architecture Design,” *Early Aspects at ICSE*: *Works*, 2007.

[8] J. Whittle and P. K. Jayaraman, “MATA: A Tool for Aspect-Oriented Modeling Based on Graph Transformation Models Workshops,” 2007.

[9] J. Klein and J. Kienzle, “Reusable Aspect Models in Proceeding of the 11th International Workshop on Aspect Oriented Modeling,” 2007.

[10] T. Cottenier, V. B. Den and E. Tazill, “Stateful Aspects: The Case for Aspect Oriented Modeling,” *Proceedings of the* 7*th Workshop on Aspect-Oriented Modelling* (*AOM*), Vancouver, 2007.

[11] M. Kandé, M. Kienzle and J. Strohmeier, “The Second International Workshop on Aspect-Oriented Modeling with UML,” 2002.

[12] O. Aldawud, E. Tazill and A. Bader, “A UML Profile for Aspect Oriented,” *Modeling in Proceedings of OOPSLA*, 2001.

[13] C. Chavez and C. Lucena, “A Metamodel for Aspect-Oriented Modeling Workshop Aspect Oriented Modeling with UML,” *Proceedings of the* 2*nd Workshop on Aspect-Oriented Modelling* (*AOM*), Vancouver, 2002.

[14] D. Xu and W. Xu, “State-Based Incremental Testing of Aspect-Oriented Programs,” *Proceedings of the* 5*th International Conference on Aspect-Oriented Software Development*, Bonn, 2006.

[15] L. Fuentes and P. Sánchez, “Towards Aspect-Oriented UML Executable Models,” *Proceedings of the* 10*th Workshop on Aspect-Oriented Modelling* (*AOM*), 6*th International Conference on Aspect-Oriented Software Development* (*AOSD*), Vancouver, 2007, pp. 28-34.

[16] J. M. Lions, D. Simoneau, G. Pitette and I. Moussa, “Extending OpenTool/UML Using Metamodeling: An Aspect Oriented Programming Case Study,” *Aspect-Oriented Modeling with UML*, 2002.

[17] Y. Han, G. Kniesel and A. Cremers, “Towards Visual AspectJ by a Meta Model and Modeling,” 6*th International Workshop* on *Aspect-Oriented Modeling*, Vancouver, 2006.

[18] L. Ramnivas, “I Want My AOP!” 2002.

[19] J. D. Gradecki and L. Nicholas, “Mastering AspectJ,” 2003.

[20] R. Mužar and M. Grgec, “Role of UML Interaction Overview Diagram in Business Domain Modeling,” *IIS Conference*, 12-14 September 2007.

[21] M. Wimmer, A. Schauerhuber and G. Kappel, “A Survey on UML-Based Aspect-Oriented Design Modeling,” *Journal of ACM Computing Surveys*, Vol. 43, No. 4, 2011.