

MODELLING OF ASPECTS USING ASPECT-ORIENTED DESIGN LANGUAGE

¹SAQIB IQBAL, ²ABDALLA MANSUR, ³GARY ALLEN

^{1,2} Department of Software Engineering, Al Ain University of Science and Technology, Al Ain, UAE

³Department of Informatics, University of Huddersfield, Huddersfield, UK

¹saqib.iqbal@aau.ac.ae, ²abdalla.mansur@aau.ac.ae, ³g.allen@hud.ac.uk

ABSTRACT

The Aspect-composition is a vital step in aspect modelling. Aspects are composed with each other and with base constructs through pointcuts defined in the aspects. Design languages address this composition by providing composition techniques and directives. However, most of the contemporary design languages lack support for inter-aspect and inner-aspect compositions. Another problem is resolving aspect interference which arises as a result of a composition. Although some techniques have been proposed to overcome aspect interference at the implementation level, the problem needs attention at the modelling level. The eradication of interference and conflicts related to aspect composition at the modelling stage could ensure better implementation and fewer conflicts. This paper provides a composition strategy equipped with new design notations and diagrams to provide support for aspect compositions, as well as inner-aspect compositions. The paper also provides a technique to prioritize aspect execution at the modelling stage to reduce aspect interference and aspect conflicts.

Keywords: *Aspect-Oriented Programming, Pointcut Modelling, Aspect Composition, Aspect-Oriented Model*

1. INTRODUCTION

Aspects interact with constructs in the base system as well as with other aspects. The interaction with the base system happens through join points, which are defined as predicates (in pointcuts) in the aspects. The interaction of an aspect with other aspects happens through inheritance relationship and/or via communications such as reference to each other's features. Besides these interactions, there are some inner interactions within an aspect as well. Pointcuts interact with each other by defining join points that include predicates defined on other pointcuts, and they interact with advices through binding relationships, which define which advice is associated with which pointcut. An aspect-oriented design strategy is required to design these interactions so that aspects are modelled comprehensively before being implemented. There have been some strategies proposed over the years for composition of aspects, examples include strategies by Zhang et al., [19], Fleurey et al., [6], and Whittle et al., [18]. These strategies have proposed techniques to compose aspects with the base system and aspects with aspects, but inner-aspect compositions among pointcuts and advices have been overlooked.

During the aspect composition process, two problems arise regarding the priority of execution. One problem is called aspect interference [9], which is related to the alteration in values of variables of aspects or the base system through the execution of advices. The problem has been addressed in several ways. At the modelling level techniques by Driver et al., [3], Zhang et al., (2007) and Reddy et al., [15], and at implementation level techniques by Nagy et al., [11], Durr et al., (2005) and Lagaisse et al. (2004) have been proposed so far. The second problem is called the shared join point problem [11] which arises when multiple aspects try to superimpose their behaviors at one join point simultaneously. Although this problem is usually addressed under the umbrella of the aspect interference problem, we mention it separately due to its importance and impact on the system design. To resolve this problem, the order of aspects is required to be determined before implementation so it does not create problems during the execution of the system. Some solutions have already been proposed by Driver et al., [3], Zhang et al., (2007) and Reddy et al., [15] but they all have their own limitations (discussed in the related work section). The research question this paper will try to answer is that can the shared join point problem and

the aspect interference problem be resolved by a diagrammatical approach?

This paper addresses the problems discussed above by proposing a composition technique which models the composition of aspects with the base system and with other aspects with the help of design notations and design diagrams. The technique also supports inner-aspect composition of pointcuts with each other and with related advices. The paper proposes a precedence mechanism which can be adopted for all types of composition for resolving the issues of aspect interference and shared join point problems at the design level. The paper does not claim that by adopting the proposed composition strategy all kinds of aspect interference (including the shared join point problem) will be removed; rather it suggests how aspectual and inner-aspectual composition can be managed, and how precedence of aspects and advices can be handled at the design level to reduce such interference. The proposed method is based on notations and semantics of AspectJ [22] technology.

The paper applies the design notations of a design language, Aspect-Oriented Design Language (AODL) developed by the authors of this paper [8] on a Tracing program example borrowed from the Eclipse AspectJ Programming Guide [4]. The composition mechanism is built upon the design models proposed by the language, and utilizes semantics and notations of the language while specifying aspect-related and base system related constructs.

The rest of the paper is structured as follows: Section 2 discusses the motivation behind the research, including composition of aspects and aspect interference problem. Section 3 provides an overview of AODL along with its design notations and design diagrams. Section 4 provides a categorization of pointcuts proposed by the authors. Section 5 explains the aspect composition mechanism with the help of an example. Section 6 outlines some of the related research in the same area, and the last section, section 7, provides a detailed discussion on the outcomes of the research and conclusion of the paper.

2. PROBLEMS AND MOTIVATION

There are two primary motivations behind this paper. First, there is a dearth of comprehensive aspect composition strategies. The existing design strategies either lack inner- aspect compositions (such as pointcut with pointcuts and pointcuts with

advices) or techniques to handle aspect interference. The proper composition of aspects provides a design blue print for the system which eventually helps in analysing aspect interactions in the system. The design of interaction can help in identifying possible aspect interference and conflicts among aspects themselves and between aspects and the base constructs. The second motivation is to provide a precedence strategy for aspect composition. The defined priority of execution of aspects on a particular base unit helps in avoiding conflicts and interference. The strategy also provides support to resolve the shared join point problem at the modelling level. The details of the problems are provided below.

2.1. Comprehensive Composition of Aspects

Aspects are tightly coupled with the base system through join points defined in their pointcuts. The aspects' code (known as their advices) also has direct references to the constructs, variables and operations in the base system. Moreover, aspects can make references to the features of other aspects and can have inter-element relationships within their own bodies. If we categorise these relationships, we come up with three different types of aspect relationships:

a) Aspect-to-base system relationships ^[1]_[SEP]

Advices define the behavior of aspects, which are superimposed on well-defined join points in the base system. Join points are control locations in the base system and are defined in terms of direct references to operations, variables and objects of the program. If we define this in simple terms, we can say that an aspect is tightly connected with the base constructs where it inserts its behavior. Any alteration to the connecting join points in the base system can make the related aspect either useless or introduce erroneous behavior at run time. ^[1]_[SEP]

b) Aspect-to-Aspect relationships ^[1]_[SEP]

In AspectJ, aspects can have relationships such as association, inheritance and dependency with other aspects. The inheritance is similarly implemented as in object-oriented programs only with some minor differences. The child aspects can override their parent's pointcuts, but cannot override the parent's advices because advices do not have unique signatures. ^[1]_[SEP]

c) Inner-Aspect relationships ^[1]_[SEP]

Within the body of an aspect, advices are tightly connected with their related pointcuts. Advices in AspectJ are defined for a certain pointcut and they contain direct reference to the pointcut in their signature (in the form of the pointcut's name).

Similarly, pointcuts can define join points on other pointcuts by directly referencing their names in their predicates. These inner-aspect relationships are required to be designed properly to capture the inner structure of an aspect.

A comprehensive aspect composition strategy would cover all types of compositions and would present a design solution to specify associations, relationships and generalization among aspects and related constructs at the modelling level.

2.2. The Aspect Interference Problem

The interaction by an aspect with the base system may create interference. Katz et al. [9] have identified several different forms of interference that may arise when an aspect interacts with the base system. We have categorized this interference into four types:

a) Shared JoinPoints

This type of interference occurs when multiple aspects try to superimpose their behavior at one join point simultaneously [11]. The priority of aspects is required to be designed before such interaction takes place. Several solutions have been proposed to prioritize the execution. Some strategies have handled this problem at the implementation level, such as AspectJ which provides the `<<declare precedence>>` stereotype to order aspects in the source code. Some approaches such as by Driver et al., [3], Zhang et al., (2007) and Reddy et al., [15] have provided order-managing techniques at the design level. ^[1]_{SEP}

b) Altered Join Points

In some instances, an aspect's behavior may change the join points of another aspect. This can happen when an aspect modifies or adds to the join point which defines the location of interaction of the second aspect. This type of interference may cause an aspect not to insert its behavior when it is supposed to do so. ^[1]_{SEP}

c) Altered Variables

An aspect can modify, add or delete a variable which is used by another aspect later in the execution. Although aspects are

designed not to share common variables, a modification to a variable which in return modifies another variable or a join point may cause this type of interference. ^[1]_{SEP}

d) Altered Aspect Behavior

In complex systems, designers are tempted to introduce a number of aspects to increase the modularity and understandability of the system. This situation may result in aspects being created which contradict the specification of other aspects. This type of interference can only be handled during the specification of aspects. Such interference can induce undesired outcomes from an aspect due to the execution of a conflicting aspect. ^[1]_{SEP}

This paper proposes a precedence mechanism which when utilized properly can help in tackling aspect interference which is the root cause of all the above-mentioned problems, and can provide a way of identifying and ordering aspects at the modelling level.





3. OVERVIEW OF ASPECT-ORIENTED DESIGN LANGUAGE

Aspect-Oriented Design Language (AODL) [8] is a design language developed by the authors of this paper, which introduces some new design notations and design diagrams for aspects and their elements. AODL is based on AspectJ technology and provides a set of extensions to UML 2.4.1 [12]. It introduces design notations for the main constructs of AspectJ, such as aspects, join points, pointcuts and advices. Design notations are used in the AODL models to describe structural and behavioural properties of an aspect and its constituent elements.

The design notations used in AODL are shown in Table 1. They have been proposed to specify and represent aspectual constructs during the software development life cycle. Each notation depicts the behaviour and characteristics of the corresponding element. The details about these notations can be found in [8].

The structural and behavioural characteristics of aspect-oriented elements are captured using design diagrams. An AODL diagrammatic Model (shown in Figure 1) shows all the diagrams used in AODL.

Table 1: AODL Design Notations

Join Point		Weaving Association	
Aspect	 <div style="border: 1px solid black; padding: 5px; width: fit-content;"> Aspect name Attributes Operations <u>Pointcuts</u> Advices </div>	<u>Pointcut</u>	 <div style="border: 1px solid black; padding: 5px; width: fit-content;"> <u>Pointcut name</u> Join points </div>

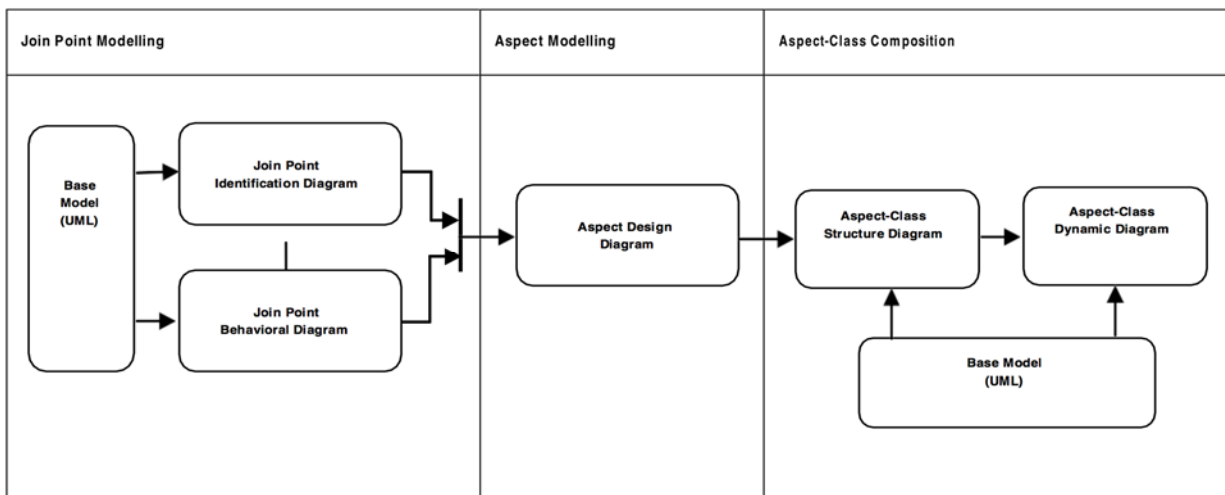


Figure 1: AODL Diagrammatic Model

4. CASE STUDY

The proposed techniques will be evaluated by implementing them on a case study. The reason to choose the evaluation by implementation is that the diagrammatic techniques such as the proposed ones could very well be tested while implemented to a real-life problem.

The proposed techniques and models in this paper are applied to the Tracing example borrowed from the Eclipse [4]. The reason to choose this case study is that it is a well-known case study by AspectJ and has been discussed in several similar papers. The other reason is the presence of identified problems in this case study, which are shared join point problem and the interference problem. There is no discussion about the case study other than its application on the proposed methodology in this paper. To know more about it, consult [4].

5. COMPOSITION OF ASPECTS

Before composing aspects together and with related constructs in the base system, their constituent elements need to be composed. We will adopt this bottom up approach for the proposed technique of dynamic composition.

Aspects can contain elements such as attributes, operations, pointcuts, inter-type declarations and advices. Attributes and operations describe static features of aspects so they are specified statically in the structural model of aspects in AODL. Intertype declarations and pointcuts, on the other hand, represent structural crosscutting of aspects and are needed to be designed in such a manner that their interactions and relationships are modelled structurally and behaviourally. In other words, these constructs should be modelled using a design model which represents their relationships with the base constructs explicitly. AODL composes them (as shown in Figure 2) along with the base constructs in a composition model which uses

structural notations to depict relationships and associations and uses dynamic models to represent the behavioural binding among aspects and the base constructs. With regard to advices, they are the action part of aspects and are connected with their corresponding pointcuts, hence they are represented along with their pointcuts in the structural as well as the behavioural models.

5.1. Static Pointcut Composition

AODL proposes a static composition of pointcuts where pointcuts are composed with each other and with their corresponding advices. Pointcuts and their features are represented in the AODL notation in the form of containers. The containers are distinguished from those of other constructs with a join point symbol on top. The list of join points is shown within the container along with the pointcut name. The relationship between a pointcut and its related advices is represented with the help of an association bearing an occurrence type (before, after or around). Advices are represented in a container with the stereotype <<advice>> which contains the advice's Id and an explanation about the implementation contained in the advice. It is to be noted here that in AODL advices are assigned ids to identify them contrary to AspectJ semantics where advices do not have a signature. Figure 2 shows an Aspect Design Model for a *Trace* aspect designed in AODL which contains a structural representation of pointcuts. The example system is the Tracing program borrowed from [4]. The objects of the base system have also been shown in the model, with association <<crosscuts>>.

The static composition model helps in identifying static features of a pointcut. It shows all the pointcuts of a specific aspect along with their related advices in a static diagrammatic way. This diagram is a part of high-level design where join points are not modelled dynamically but rather are represented as static features of their respective pointcuts.

5.2. Dynamic Pointcut Composition

Pointcuts are composed dynamically when aspects are woven into the system. AODL designs each join point included in a pointcut with the help of a behavioural diagram. The diagrams are based on the UML communication diagram. Communication diagrams (previously known as collaboration diagrams) help in designing the dynamic collaboration of objects with each other in UML. The interaction is shown in the form of message passing among objects. AODL exploits this diagram for the join points' selection during the composition of aspects. Before explaining the pointcut composition, we introduce categories of pointcuts. We have categorized Pointcuts used in AspectJ into four types:

Scope Pointcuts: The pointcuts that define a scope of selection of join points in the base system are included in this category. Examples of such pointcuts are: *cflow()*, *within()*, *withincode()*, *cflowbelow()*, *this()*, *target()* and *args()*.

Method Pointcuts: The pointcuts that are defined on methods and constructors of classes of the base system are part of this category. Examples of pointcuts defined in this category are *call()*, *execution()*, *get()*, *set()*, *call(const)*, *execution(const)*, *handler()*, *adviceexecution()*, *initialization()*, *preinitialization()*, and *staticinitialization()*.

Peer Pointcuts: Peer pointcuts select other pointcuts defined in the same aspect or a related aspect. These pointcuts are defined on already defined pointcuts. Some of the examples in this category are *pointcutID()*, *!pointcut()*, *pointcut0 & & pointcut1*, *pointcut0 || pointcut1* and *(pointcut)*.

Conditional Pointcuts: Conditional pointcuts are defined on join points satisfying a Boolean condition. These pointcuts may be defined on a type of Boolean operator such as AND, OR, NOT etc. The *if(Boolean)* expression is also part of this category.

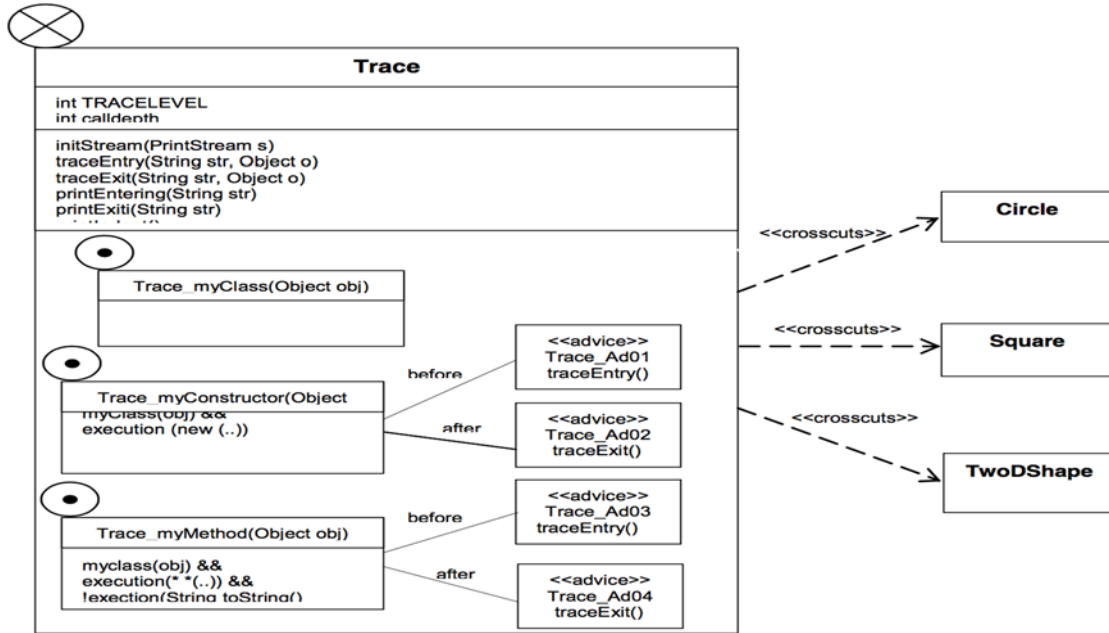


Figure 2: AODL Aspect Design Model for Trace Aspect

5.3. Design of pointcuts

AODL models Method Pointcuts and Peer Pointcuts using extended versions of UML’s Communication Diagram, whereas Scope Pointcuts and Conditional Pointcuts are statically represented in the pointcut container. The reason to select the Communication Diagram to depict the behavioural characteristics of a join point is its

usage in UML where it provides means to show the behavioural interactions among objects (UML 2.4.1). The join points included in Method Pointcuts are usually defined on the call or execution of methods, constructors or exception handlers. The communication diagram can depict all these types of join point as shown in Figure 3.

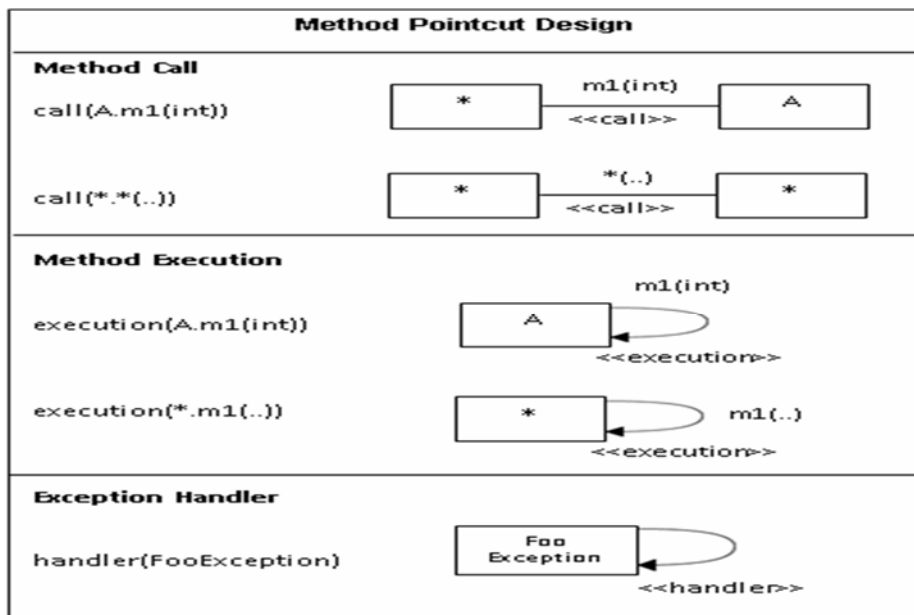


Figure 3: Design of Method Pointcuts

AODL adds stereotypes such as `<< call >>` for calling of a method, `<< execution >>` for execution of a method, and

`<< handler >>` for call of an exception handler. The join point defined on execution of an internal method of a class is denoted by a self-call with `<< execution >>` stereotype.

5.4. Pointcut Composition

AODL uses collaboration classifiers of UML to contain the dynamic behavior of a join point of a pointcut. Collaboration is used to show the structure of collaborating model elements in UML where each element performs a unique function. The combined interaction of collaborating elements forms the desired functionality (UML 2.4.1). The reason behind selecting collaboration classifier to contain a join point is threefold; (i) a container element is required to contain dynamic behavior of a join point, (ii) an element is needed which could be combined with other related elements to depict the joining of join points in a pointcut, and (iii) a UML element is required to contain the join point as the joinpoint's internal behavior is being depicted by UML's communication diagram. Each collaboration included in the pointcut nests a communication diagram to show the behavior of a join point. The collaborations are then combined with the help of Boolean operators such as AND, OR and NOT to form a complete pointcut model.

Pointcuts are then composed with their respective advices through an association labelled with the occurrence type (before, around and after) which decides when the advice is supposed to execute. Advices are in return composed with the parent aspect through composition associations. Pointcuts, sometimes, may contain direct reference to other pointcuts. There are two such examples. In first case, a pointcut contains another pointcut's reference in the form of a join point. AODL shows this relationship with the stereotype `<< includes >>`. In the second situation, a pointcut may define a join point that uses a reference to another join point. AODL shows this relationship with the `<< uses >>` stereotype.

Figure 4 shows a Dynamic Pointcut Composition Model of the aspects *Trace* and *TraceMyClasses* of the Tracing case study. The aspect *TraceMyClasses* contains only one pointcut, *myClass*, which is an overridden method of an abstract pointcut, *myClass*, in the *Trace* aspect. The definition of this pointcut is:

```
pointcut myClass(Object obj): this(obj) &&
(within(TwoDShape) // within(Circle) //
within(Square));
```

The pointcut is depicted in a pointcut container model where the full definition has been shown statically. Since there is no method pointcut involved in the definition, so no collaboration element has been included in the model.

The *Trace* aspect has three pointcuts, *myClass*, *myConstructor*, and *myMethod*, where *myClass* is an abstract pointcut. AODL designs both pointcuts using communication diagrams and collaboration elements. The *myClass* pointcut has the following definition.

```
abstract pointcut myClass(Object obj);
```

This has been shown in a pointcut container model. The body of the aspect remains empty since the aspect has no definition. All the aspects implementing this pointcut will have an `<< implements >>` relationship directed to this pointcut, as shown in Figure 4.

The *myConstructor* pointcut has the following definition:

```
pointcut myConstructor(Object obj):
myClass(obj) && execution(new(..));
```

There are two predicates in this pointcut. The first predicate contains all the join points which are captured by the *myClass* pointcut, and the second predicate contains a join point on the initiation of any object in the program.

The diagram shows both the predicates in their collaboration containers, and the containers are combined with a Boolean operator, AND. The pointcut is also related with the *myClass* pointcut of the *TraceMyClasses* aspect which has been shown with the help of an `<< includes >>` relationship.

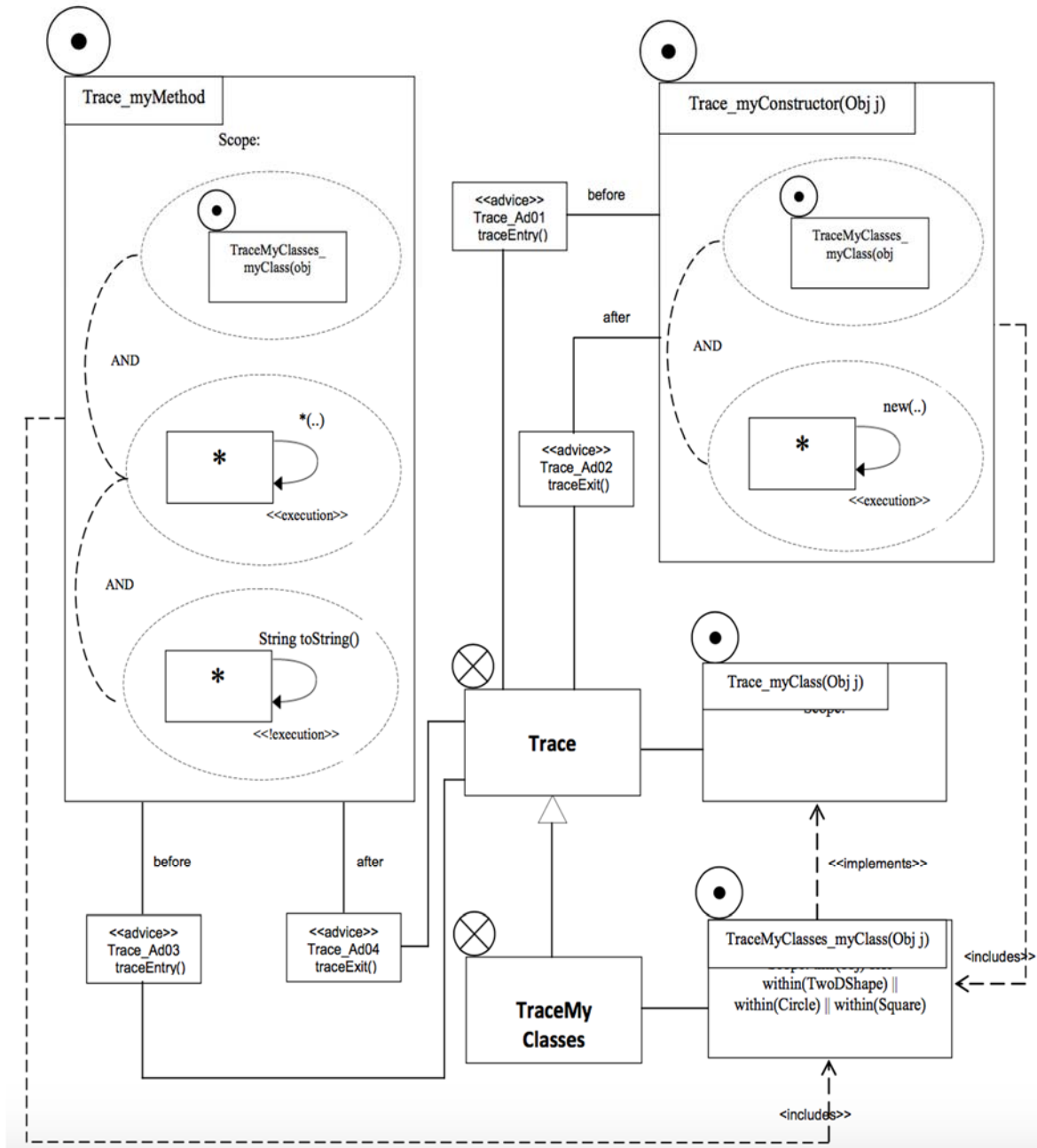


Figure 4: Dynamic Composition of Pointcuts for the Tracing Example

The third pointcut of the *Trace* aspect has the following

definition:

```
pointcut myMethod(Object obj):
    myClass(obj) && execution(* *(..)) &&
    !execution(String toString());
```

The pointcut has three different predicates joined together with the AND operator. The first predicate indicates all join points captured by the

myClass pointcut. The second predicate indicates join points on the execution of all types of methods in the program and the third pointcut ensures that these methods must not include *toString()* methods. The model for this pointcut in Figure 4 contains three different collaboration elements containing behavioural representation of each join point. The collaborations are combined with the help of a joiner, AND. Since the pointcut contains a direct reference to the myClass pointcut in *TraceMyClasses* so it has been shown related to the

pointcut with the help of an `<<includes>>` relationship.

5.5. Aspect Composition

AODL composes aspects with base constructs in two models. The first model designs the dynamic composition of aspects with the base objects. The purpose is to capture composition of both the constructs at run time. The model is based

on UML's communication diagram, which is used to capture dynamic behaviour of objects in UML. Aspects have been added to this diagram with new notations and associations to depict composition of aspectual behaviour (advices) at run time. Figure 5 shows an Aspect-Class Dynamic Model for the Tracing example, where the process of creation of a TwoDShape has been depicted.

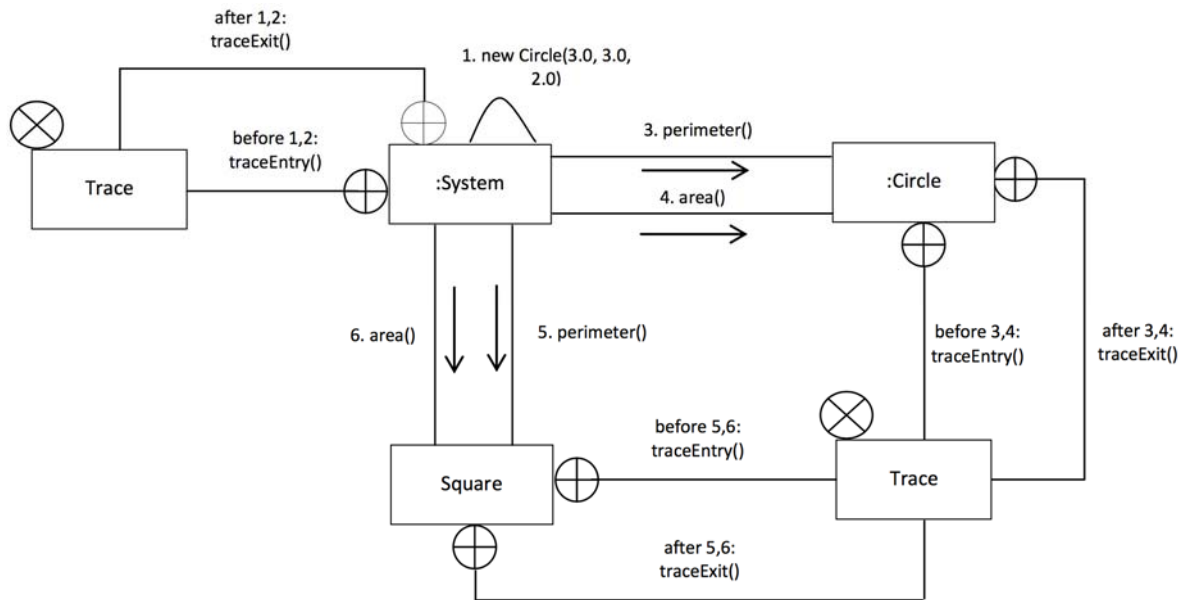


Figure 5: Aspect-Class Structural Composition

The aspect *Trace* weaves its behaviour at different points during the message passing between objects. The association for weaving is shown by an arrow with a plus signed head to depict the appending process of an advice. The second model is a structural model, called the Aspect-Class Structural model, which captures structural composition of aspects with classes. This model shows interaction between an aspect and a base class with a stereotype `<<crosscuts>>`. For example, in Figure 6, all three classes, TwoDShape, Square and Circle, are associated with both the aspects, *Trace* and *TraceMyClasses*, through `<<crosscuts>>` relationship.

The model provides support to depicts other relationships such generalization, dependency and association as well. The model also provides support for precedence allocation. The stereotype `<<precedes>>` depicts a relationship between two aspects where *A* `<<precedes>>` *B* indicates that aspect *A* has priority regarding the advice execution over aspect *B*. This relationship is also illustrated in Figure 6 where the *TraceMyClasses* aspect is shown as having priority over the *Trace* aspect.

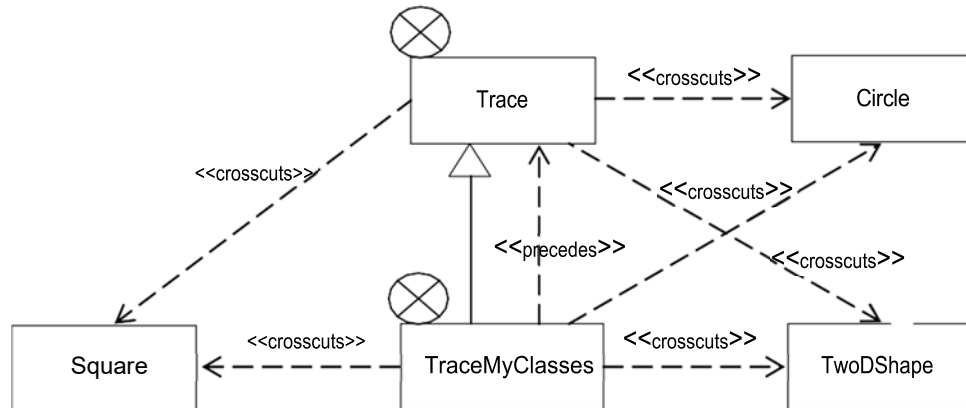


Figure 6: Aspect-Class Structural Composition

Depicting aspects along with base classes in a structural model as shown in Figure 6 enhances the expressiveness of the system and helps in designing and understanding the structural composition of aspects. The order implemented by the `<<precedes>>` stereotype provides a means to control or even eradicate some types of aspect interference. The proposed approach, however, does not claim that all issues related to all kinds of interference can be resolved by adopting this mechanism. The method only suggests that it can help in achieving consistency in aspect composition and can help in ordering aspects' execution at the modelling stage.

6. RELATED WORK

Aspect composition has been offered by almost all aspect-oriented software design and modelling strategies. The strategies are categorized into symmetric and asymmetric modelling techniques. The symmetric techniques treat aspects and base classes similarly and propose design strategies for both the constructs. Examples of the approaches using this technique include: Theme/UML approach [1,3], goal-oriented approach [10], subject-oriented approach [3], and multi-dimensional separation of concerns approach [7]. The design strategies using asymmetric technique only support modelling of aspect-oriented concerns and compose them with the base models without providing any modelling support to the base constructs. Examples of these strategies are: AODM approach [14,15], AOSD profiles [5] and JAC Design Notations of Pawlak et al. [14]. They all follow asymmetric notations and semantics of AspectJ technology. There are some hybrid approaches as well which do not fully fall in either

technique but rather employ a mixture of both the techniques; a noted example of this type of approach is a methodology by Reddy et al. [15]. The problems mentioned in this paper regarding aspect composition and aspect interference have been handled by only a few approaches before. For example, an aspect composition approach for Motorola Weavr has been proposed by Zhang et al. [19] in which they suggest three ordering tags for aspect models, `<<follows>>`, `<<dependent on>>` and `<<hidden by>>`. These tags are used to provide a precedence strategy to avoid aspect interference. Similarly, AspectJ [22] provides a declare precedence keyword to prioritize aspects in order to reduce aspect interference. Similar types of techniques have been provided by Reddy et al., [15], which adopts a follows keyword to resolve precedence issue, and by Clarke and Baniassad [1], which offers the *prec* tag to order the priority. Our approach is different from these as it does not only provide an aspect composition mechanism but also composes aspectual elements at the modelling level, thus providing support for locating and rectifying aspect interference problems early in the development cycle.

A lot of research into resolution of aspect interference has been conducted at the implementation level. Most of the approaches consider this problem an implementation level problem. For instance, Lagaisse et al. [23] proposed integration contracts to avoid semantic interference, Nagy et al., [11] proposed constraints for managing orders and structural constraints, and Durr et al., [24] proposed a technique to define advices in terms of operations on the aspect model in order to resolve aspect interference.

Although aspect composition and aspect interference have been addressed at both the modelling and the implementation levels, the expressivity in terms of composition of inner-aspect interactions and aspectual elements such as pointcuts and advices is lacking in most of the discussed strategies. This paper provides a composition approach for all levels of binding of aspects with base models and with each other. The paper provides static and dynamic models to address all issues of aspect composition, hence aiding in resolving conflicts and aspect interference at each level. The paper also proposes a stereotype << *precedes*>> to order aspect execution during the weaving process.

7. DISCUSSION AND FUTURE WORK

AODL has proposed composition strategies for aspects and their constituent elements at the design and modelling level. The strategies provide means to depict structural composition as well as behavioral composition of aspects. Pointcut composition is carried out by first composing pointcuts with the related advices and with the base constructs statically using a Static Composition Diagram. This diagram helps in specifying a certain pointcut within the container of its parent aspect. The features of the pointcut are included in its definition, and its associations with related advices are depicted with the help of design notations and design diagrams. The diagrams also show the base constructs which interact with pointcuts. This diagrammatic approach to specify a pointcut improves comprehensibility of the model.

The behavioral composition of pointcuts is carried out using Dynamic Composition Models. These models provide a means to depict the behavioral characteristics of join points and the relationship among pointcuts and their related advices. The Dynamic Composition Model helps in understanding the weaving process of aspects at the join point and object-interaction level. For making the dynamic composition more expressive, AODL categorizes pointcuts into four types, Method Pointcuts, Scope Pointcuts, Peer Pointcuts and Conditional Pointcuts. The categories help in understanding characteristics and behavior of a pointcut. The categories also help in the dynamic composition process where pointcuts are designed according to their type. AODL provides a Dynamic Composition Model for pointcuts. The structural composition of aspects has been represented with the help of an

Aspect-Class Structural Model which shows the structure of aspect composition. It provides support for depicting all types of relationships among aspects including associations, generalization and dependencies. The model also introduces a priority mechanism where precedence of an aspect is shown with the help of the << *precedes*>> stereotype. The approach is still primitive and requires more investigation before claims can be made of resolving all types of aspect interference.

This paper introduces an on-going research work which is still progressing. The future investigations will focus on addressing issues related to pointcut composition, such as the fragile pointcut problem and shared join point problem. The idea is to propose general notations to depict relationships between aspects and base classes. The future work also includes development of tool-support for the proposed aspect composition techniques, and implementation of at least one large case study to fully test the efficacy and scalability of the approach.

REFERENCES:

- [1] Clarke, S. and Baniassad, E. “Aspect-Oriented Analysis and Design: The Theme Approach”. Addison Wesley, 2005
- [2] Clarke S., Harrison W., Ossher H., and Tarr P. “Subject-Oriented Design: Towards Improved Alignment of Requirements, Design and Code”. Presented at *Proc. Object-Oriented Programming, Systems, Languages and Applications* (OOPSLA 1999), 1999 Denver, Colorado, USA.
- [3] Driver, C., Cahill, V., and Clarke, S. “Separation of distributed real-time embedded concerns with Theme/UML”. In *Proceedings of the 5th International Workshop on Model-Based Methodologies for Pervasive and Embedded Software*. IEEE Computer Society, 27–33. Eclipse, 2012.
- [4] Aspect-J Programming Guide, <http://aspectj.org/doc/dist/progguide/index.html>, Accessed on 5th of June. 2017.
- [5] Elrad, T., Aldawud, O., and Bader, A. “Expressing Aspects Using UML Behavioral and Structural Diagrams”. In R.E. Filman, T. Elrad, S. Clarke, and M. Aksit, editors, *Aspect-Oriented Software Development*, 2005, pages 459-478. Addison-Wesley, Boston.
- [6] Fleurey, F., Baudry, B., France, R., and Ghosh, S. “A Generic Approach for Automatic Model Composition”. In *AOM@MoDELS'07*, 2007, 11th Int. Workshop on Aspect-Oriented

- Modelling, Nashville TN USA.
- [7] France R., Georg G., and Ray I. “Supporting Multi- Dimensional Separation of Design Concerns”. Presented at *Workshop on Aspect-oriented Modelling with UML* (held with AOSD 2003), 2003, Boston, Massachusetts, USA
- [8] Iqbal, S. and Allen, G. “Designing Aspects with AODL”. *International Journal of Software Engineering*, 2011, ISSN 1687-6954 (In Press)
- [9] Katz, E. et al.” Detecting Interference among Aspects”. *AOSD Europe Deliverable*, 2008, D116
- [10] Lamsweerde A. “Goal-Oriented Requirements Engineering: A Guided Tour”. Presented at *5th IEEE International Symposium on Requirements Engineering*, 2001.
- [11] Nagy, I., Lodewijk, B., and Mehmet, A. “Composing aspects at shared join points”. In Proceedings of *International Conference NetObjectDays (NODE)*, 2005, volume P-69 of Lecture Notes
- [12] OMG, 2017, Unified Modelling Language, <http://www.uml.org/>. Accessed on June 2, 2017.
- [13] Pascal, D., Tom, S., Lodewijk, B., and Mehmet, A. “Reasoning About Semantic Conflicts Between Aspects”. In *the 2nd European Interactive Workshop on Aspects in Software (EIWAS)*, 2005, Brussels, Belgium.
- [14] Pawlak, R., Seinturier, L., Duchien, L., Martelli, L., Legond-Aubry, F., and Florin G. “Aspect-Oriented Software Development with Java Aspect Components”. In *Aspect-Oriented Software Development*, 2005, pages 343-369. Addison-Wesley, Boston.
- [15] Reddy, R., Ghosh, S., France, R., Straw G., Bieman, J. M., McEachen, N., Song, E., and Georg, G. “Directives for Composing Aspect-Oriented Design Class Models”. *Transactions on Aspect-Oriented Software Development*, 2006, LNCS 3880, Springer-Verlag, pp. 75-105.
- [16] Stein, D., Hanenberg, S., and Unland, R.. “Expressing Different Conceptual Models of Join Point Selections in Aspect-Oriented Design”. In Proc.of the *5th International Conference on Aspect-Oriented Software Development (AOSD'06)*, 2006, Bonn, Germany, pages 15-26. ACM Press.
- [17] Stricker, V., Hanenberg, S., Stein, D. “Designing design constraints in the UML using join point designation diagrams”. In Proceedings of the *47th International Conference on Objects, Components, Models and Patterns (TOOLS'09)* 2009.
- [18] Whittle, J., Jayaraman, P., Elkhodary, A., Moreira, A., and Araújo, J. “MATA: A Unified Approach for Composing UML Aspect Models Based on Graph Transformation”. *Aspect-Oriented Software Development*. 2009. VI, 6:191-237.
- [19] Zhang, J., Cottenier, T., Berg, A. V. D., and Gray, J.. “Aspect Composition in the Motorola Aspect- Oriented Modelling Weaver”. In *Journal of Object Technology*, vol. 6, no. 7, Special Issue: Aspect-Oriented Modelling, 2007, pp. 89-108.
- [20] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold. An overview of AspectJ. In Proceedings of the European Conference on Object-Oriented Programming, 2001.
- [21] Lagaisse, B., Joosen, W., and De Win, B. “Managing semantic interference with aspect integration contracts”. In *SPLAT: Software Engineering Properties of Languages for Aspect Technologies*. 2004.
- [22] Durr, P., Staijen, T., Bergmans, L., and Mehmet A., 2005. Reasoning About Semantic Conflicts Between Aspects. Presented in 2nd European Interactive Workshop on Aspects in Software (EIWAS), Brussels, Belgium.