Towards a UML Meta Model Extension for Aspect-Oriented Modeling

Meriem Chibani
Department of Mathematics and Computer Science
University of Oum El Bouaghi
Oum El Bouaghi, Algeria
e-mail: c.meriem@univ-oeb.dz

Brahim Belattar
Department of Computer Science
University of Batna
Batna, Algeria
e-mail: brahim.belattar@univ-batna.dz

Abdelhabib Bourouis
Department of Mathematics and Computer Science
University of Oum El Bouaghi
Oum El Bouaghi, Algeria
e-mail: a.bourouis@univ-oeb.dz

Abstract— **The aspect-oriented programming paradigm (AOP) as a way of improving the separation of concerns principle has emerged initially at the programming level using strong languages like AspectJ. Currently, it becomes mature to stretch at premature stages of the software development process namely, the Aspect-Oriented Software Development (AOSD) which is a popular topic of software engineering research that leads to more dependable, reusable and maintainable artifacts. In this paper, we propose a UML profile for modeling crosscutting concerns where the separation of concerns is maintained to the level of code and the weaving is done by an AspectJ compiler.**

Keywords-Aspect-Oriented Programming (AOP); UML profile; AspectJ; Aspect-Oriented Software Development.

I. INTRODUCTION

Besides functional concerns, software system development requires other concerns, namely crosscutting concerns as logging, distribution, error handling and security. These concerns cross cut the system functional modules, which produces a scattered and tangled design and decreases software's maintainability and modularity. The object-oriented paradigm does not satisfy the separation of concerns principle. It provides a powerful way to separate core concerns but it could not modularize crosscutting concerns in separate units. The aspect-orientation has originally emerged at the programming level with the well-known AspectJ language [1], in the late 1990s. Its use is no longer restricted to the programming level but more and more stretches over early phases of the software development life cycle such as requirements engineering, analysis and design. This new field is called the Aspect-Oriented Software Development (AOSD).

Aspect-oriented programming has emerged as a solution paradigm to overcome modularization problem. It distinguishes between the different categories of concerns, decreases coupling between concerns and more generally, it increases reuse. An AOP system may include many constructs where the central one is the aspect unit, which consists of two parts: dynamic crosscutting constructs and static ones. Dynamic crosscutting constructs provide a way to affect the behavior of a system. Join points are the points in the execution flow of an application; and pointcuts, a mechanism for selecting join points. The aspects have advices that are attached to one or more join points. When an advice is attached to join points, it will be executed, guided by its modifier which may specify the execution time relative to the join points: before, after, around, after exception or even after return value. These advices have an additional instance variable named thisJoinPoint that encapsulates the contextual information captured from the current junction. On the other hand, static crosscutting constructs alter static structure of the system. For example, when implementing tracing crosscutting concern, the introduction of a logger field into each traced class could be needed and inter-type declaration constructs make such modifications possible. In some situations, the need to detect certain conditions could arise, typically the existence of particular join points, before the execution of the system for which weave-time declaration constructs are suitable [2]. Furthermore, one of the main elements of AOP is the "weaving" mechanism which composes classes and aspects to produce a system with a new semantics. It could be performed before or after compilation and is known as static weaving. On the other hand, dynamic weaving is performed at load-time or run-time [3].

For an Aspect-Oriented Modeling (AOM) notation that provides a foundation for achieving better concern separation and integration, there is a need for several requirements. A general purpose, UML-based visual modeling language has several advantages over textual and domain specific alternatives. The notation should be complete, which means having a supporting abstraction for each of the commonly accepted AOSD concepts (aspect, component, pointcut, advice, static and dynamic crosscutting, Aspect-component relation and aspect-aspect relation). Furthermore, different concepts should be implicitly or explicitly mapped to different existing or new first-class UML elements. The notation should be independent from implementation language until the lowest level of detail is provided. In this way, the resulting aspect-oriented architectural models could be easily translated into elements of distinct aspect-oriented programming languages/frameworks and detailed design notations. Finally, the integrated UML-based notation should promote simplicity and avoid unnecessary extensions [4].

The Unified Modeling Language (UML) is a standard object oriented modeling language for specifying, visualizing, constructing, and documenting the artifacts of a system process. To enable it to represent the AOSD concepts at the design level, two alternatives are available. The general extension alternative aims at modifying the meta model of UML to include concepts related to the paradigm and is currently impractical due to a lack of tools support. The second alternative aims at building a UML profile which provides extension mechanisms [5]. UML extension mechanisms are based on “Stereotypes”, “Tagged Values”, and “Constraints” concepts. Briefly said, stereotypes are means of extending the UML metamodel classes, while tagged values are properties for stereotypes and constraints are used to restrict the stereotype vocabulary.

In this paper, we propose a UML v2.4 profile for modeling crosscutting concerns at the design level. The separation of concerns is maintained to the level of code and the weaving is done by an AspectJ compiler. We have used only UML class diagrams where the system behavior is not specified in UML behavioral diagrams.

The rest of the paper is organized as follows. Section 2 describes briefly the related work. Section 3 presents the proposed profile, while Section 4 discusses an application example. Finally, a conclusion is given in Section 5.

II. RELATED WORK

An aspect-oriented UML profile is one of the most challenges in closing the gap between AOP and aspect-oriented modeling phases. Initial discussion on UML profile was presented in [6], which proposed the specification of aspects as stereotypes on classes and aspects behavior as association relationship using collaboration diagram. The profile was specific for synchronization aspect and without addressing joinpoints, advice and pointcut concepts. It was later extended to include advice and pointcut specification in [7]. Similarly, in [8][9], initial aspect-oriented extensions using UML metamodels were described with a lack in graphical representation of most aspect-oriented constructs such as static crosscutting, join point and pointcuts.

In contrast to previous works, a complete AspectJ profile without textual specification was discussed by Evermann [10]. It was developed using the commercial tool MagicDraw with XMI (XML Metadata Interchange) format which allows easy code generation. However, it has inconsistencies compared to what is required by the paradigm and the proof was provided by a process for aspect-oriented profile checking in [11]. In [12], Evermann profile was extended to support aspect-oriented frameworks taking into consideration some AspectJ idioms, patterns and also stereotypes from a profile for object-oriented frameworks called UML-F.

In the terminology of Model Driven Architecture (MDA), unlike the previous works, which allow modeling only of Platform Specific Models (PSM), a Platform Independent Modeling (PIM) profile was developed in [13], after the identification of commonalities and differences between two representative AOSD implementations. As shown in Table 1, the significant differences between the implementation languages, i.e., AspectJ and AspectS, make the resulting profile complex to apply to models. Thus, a profile dedicated to a platform-specific technology is the candidate solution for reducing complexity [14].

TABLE I COMPARAISON OF AOP APPROACHES [14]

	AspectJ	AspectS	AspectML
Aspects can be instantiated	×	√	AspectML does not have an aspect construct.
Aspect inheritance	×	√	
Nested aspects	√	×	
Privileged aspects	√	×	
Polymorphic pointcuts	×	×	√
Polymorphic advices	×	×	√
Advice on field access	√	×	NA

Recently, Gowri [15] modeled joinpoint as sequence diagrams and it adopted XMI to deploy the profile in available CASE tools. It is a generic profile that captures only few of the AspectJ extensions.

The present proposal is an extension of the Evermann profile with several improvements. It represents a complete AspectJ imitation with two main contributions:

- Extending Evermann profile to comprise static crosscutting representation as shown in Figure 1, with highlighted stereotypes, e.g., the weave-time error and warning declarations constructs.
- Doing a considerable number of changes, for instance at the level of the used metaclasses and relations between stereotyped profile elements in order to eliminate Evermann profile complexity and improving efficiency, e.g., the metaclass Property is sufficient to represent the pointcut instead of the metaclass StructuralFeature, add the conditionalPointcut stereotype, etc.

III. THE PROPOSED PROFILE

Our profile is developed using the UML commercial tool MagicDraw [16]. It provides an efficient graphical UML2 editor for modeling and profiling with OCL verification engine for constraints checking.

A. Aspect

Aspect represents the modular unit in AOP paradigm that includes all crosscutting constructs such as advice and pointcut. The aspect is like a class, which may have both attributes and operations, access modifiers (public, private, protected or package), the ability to extend other classes, realize interfaces in addition to the fact that they may be abstract. Thereby, aspects are modeled by means of a stereotype <<aspect>> of Class, as shown in the Figure 1. Despite their similarities, aspects are different from classes and in order to overcome this, additional attributes and

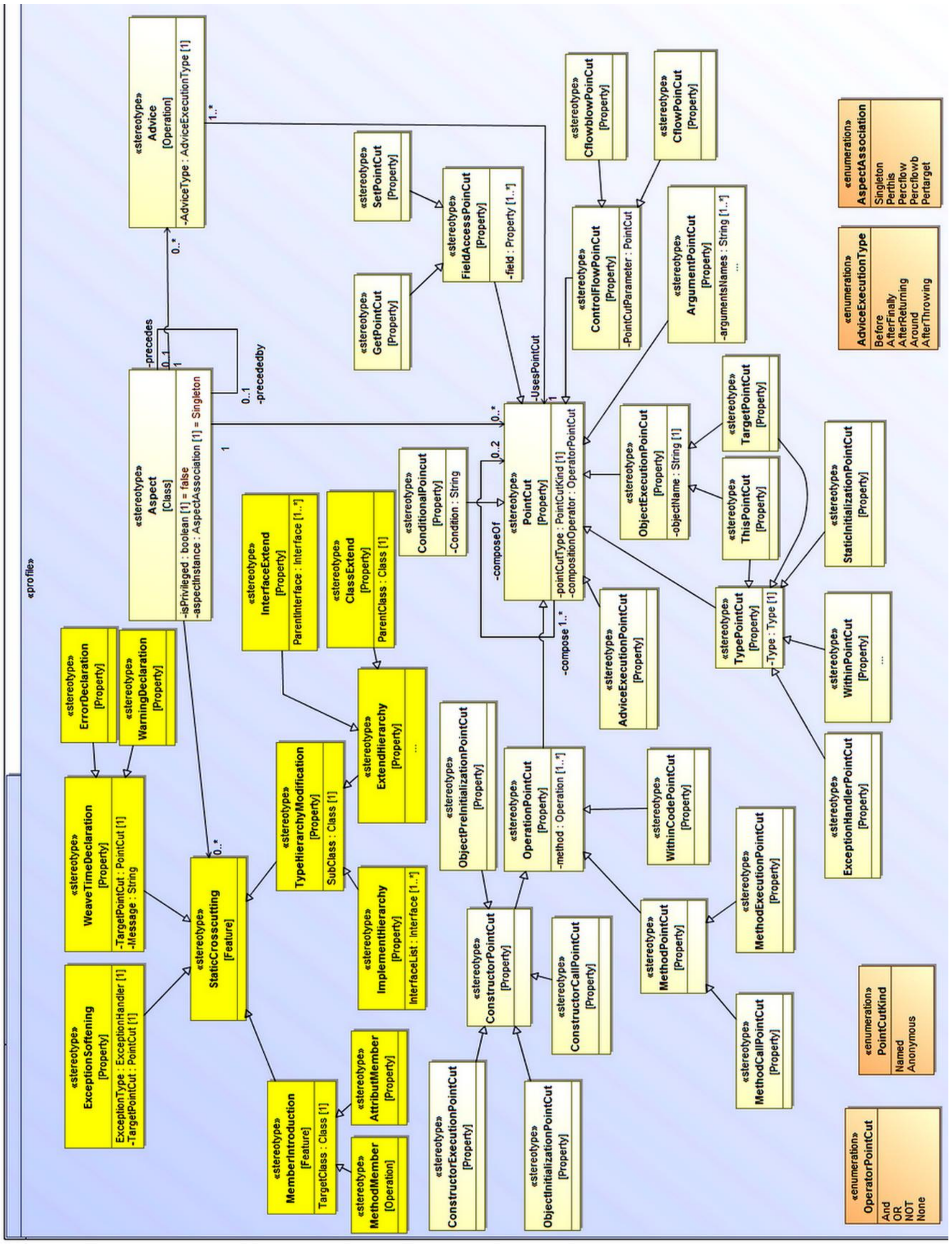
«profile»
«stereotype» Advice [Operation]
-AdviceType : AdviceExecutionType [1]
«stereotype» Aspect [Class]
-isPrivileged : boolean [1] = false
-aspectInstance : AspectAssociation [1] = Singleton
-precedes
-precededby
0..1
0..*
1..*
1
«stereotype» PointCut [Property]
-pointCutType : PointCutKind [1]
-compositionOperator : OperatorPointCut
-UsesPointCut
-composeOf
-compose 1..*
0..2
«stereotype» SetPointCut [Property]
«stereotype» GetPointCut [Property]
«stereotype» FieldAccessPoinCut [Property]
-field : Property [1..*]
«stereotype» CflowblowPoinCut [Property]
«stereotype» CflowPoinCut [Property]
«stereotype» ControlFlowPoinCut [Property]
-PointCutParameter : PointCut
«stereotype» ArgumentPointCut [Property]
-argumentsNames : String [1..*]
«stereotype» ConditionalPoincut [Property]
-Condition : String
«stereotype» ObjectExecutionPoinCut [Property]
-objectName : String [1]
«stereotype» TargetPointCut [Property]
«stereotype» ThisPointCut [Property]
«stereotype» StaticInitializationPointCut [Property]
«stereotype» TypePointCut [Property]
-Type : Type [1]
«stereotype» WithinPointCut [Property]
«stereotype» ExceptionHandlerPointCut [Property]
«stereotype» AdviceExecutionPointCut [Property]
«stereotype» InterfaceExtend [Property]
ParentInterface : Interface [1..*]
«stereotype» ClassExtend [Property]
ParentClass : Class [1]
«stereotype» ExtendHierarchy [Property]
«stereotype» TypeHierarchyModification [Property]
SubClass : Class [1]
«stereotype» ImplementHierarchy [Property]
InterfaceList : Interface [1..*]
«stereotype» ErrorDeclaration [Property]
«stereotype» WarningDeclaration [Property]
«stereotype» WeaveTimeDeclaration [Property]
-TargetPointCut : PointCut [1]
-Message : String
«stereotype» StaticCrosscutting [Feature]
«stereotype» ExceptionSoftening [Property]
ExceptionType : ExceptionHandler [1]
-TargetPointCut : PointCut [1]
«stereotype» MemberIntroduction [Feature]
TargetClass : Class [1]
«stereotype» AttributMember [Property]
«stereotype» MethodMember [Operation]
«stereotype» ObjectPreInitializationPointCut [Property]
«stereotype» OperationPointCut [Property]
-method : Operation [1..*]
«stereotype» WithinCodePointCut [Property]
«stereotype» ConstructorPointCut [Property]
«stereotype» ConstructorCallPointCut [Property]
«stereotype» MethodPointCut [Property]
«stereotype» MethodExecutionPointCut [Property]
«stereotype» MethodCallPointCut [Property]
«stereotype» ConstructorExecutionPointCut [Property]
«stereotype» ObjectInitializationPointCut [Property]
«enumeration» AspectAssociation
Singleton
Perthis
Percflow
Percflowb
Pertarget
«enumeration» AdviceExecutionType
Before
AfterFinally
AfterReturning
Around
AfterThrowing
«enumeration» PointCutKind
Named
Anonymous
«enumeration» OperatorPointCut
And
OR
NOT
None

Figure 1. AspectJ profile.

constraints on the metaclass Aspect are used.

1) Attributes

- isPriviliged: a Boolean which indicates if the aspect has a special privileged access specifier. If true, the aspect may access to private members of the classes which are crosscutting.
- aspectInstance: specifies the aspect instantiation model. Its possible values are: perthis, pertarget, percflow, singleton, or percflowb. Its default value is singleton, which means that the aspect has a unique instance.
- Precedence: it is modeled using a recursive (reflexive) association and determines the execution order of aspects with the same join point.

2) Constraints

In contrast to the class, concrete aspect could not declare generic parameters. Further, concrete aspect is not available for inheritance.

B. Advice

Advice is a dynamic construct in AspectJ, whereby it alters the behavior of the system at joinpoints selected by pointcuts. Because both advice and method express the behavior, have name, have arguments, could throw exceptions and have a body, we model advices using the metaclass Advice which extends the metaclass Operation.

1) Attributes

AdviceExecutionType: enumeration attribute that determines the type of the advice, i.e., before, after or around.

2) Constraints

In contrast to the method, which applies through an explicit call, the advice applies automatically in crosscutting manner. This is why an advice doesn't have an access specifier and only the "around" advice includes return type.

C. Pointcut

Pointcut selects the joinpoints with a structural description and has no relation with the dynamic behavior. This is why we model it using the metaclass Property and we add the constraint that the pointcut stereotype may be only applied to classes that are stereotyped Aspect. Furthermore, the metaclass Pointcut has additional attributes as follows:

- pointcutType: determines if the pointcut has a name or is anonymous.
- A pointcut may be composite, including other pointcuts using the OperatorPointcut enumeration. This mechanism is specified using a recursive association.

D. Static Crosscutting

Although advice alters the behavior of the system, static crosscutting alters its static structure in a crosscutting manner with structural specification. It is modeled using the metaclass feature. It may be of different types, exception softening, weave-time and warning declaration, or member introduction. A constraint is added to ensure that the static crosscutting stereotype is applied only to classes that are stereotyped Aspect.

IV. CASE STUDY

In order to validate the applicability and efficiency of the proposed profile, we have chosen a simple application that is used frequently in the literature as a motivation example [17]. The Line, Point and FigureElement classes as shown in Figure 2, include the display.update() method as a crosscutting behavior. AspectJ proposes a solution to localize and separate this crosscutting concern by means of an anonymous pointcut and an "after" execution advice as follows:

after(): call(void FigureElement+.set (..))*
|| call(void FigureElement.moveBy(int, int)) {
Display.update();

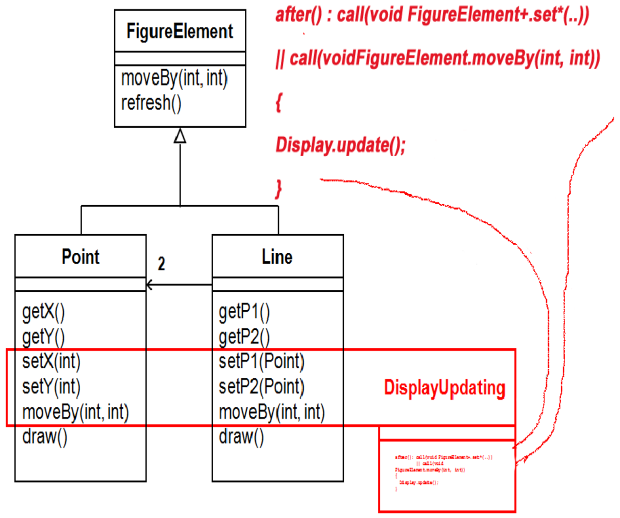

Figure 2. The AspectJ solution for the crosscutting Display.update()method.

In order to use the aspect-oriented paradigm at the design level, we apply our profile to the model. The profile metaclasses became stereotypes and their attributes became tags values with the DisplayUpdating aspect, as shown in Figure 3.

V. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a UML profile as an aspect-oriented modeling contribution based on AspectJ language. Our proposal has several strength points:

- It is a complete specification of the AspectJ language (aspect, advice, pointcut, static crosscutting constructs) in terms of the UML metamodel.
- Compliant with the XMI format, which means that it is possible to manipulate and exchange the profile between UML case tools.

Nevertheless, it remains open to future improvements, namely:

- Generating AspectJ code automatically from the UML model, which is compliant with the XMI standard and fully specified in terms of the metamodel. This could be accomplished by applying MDE/MDA tools and languages, which are already available and mature.
- Demonstrate the applicability and benefits of this profile in various areas. We intend to apply it shortly in the Modeling and Simulation domain.

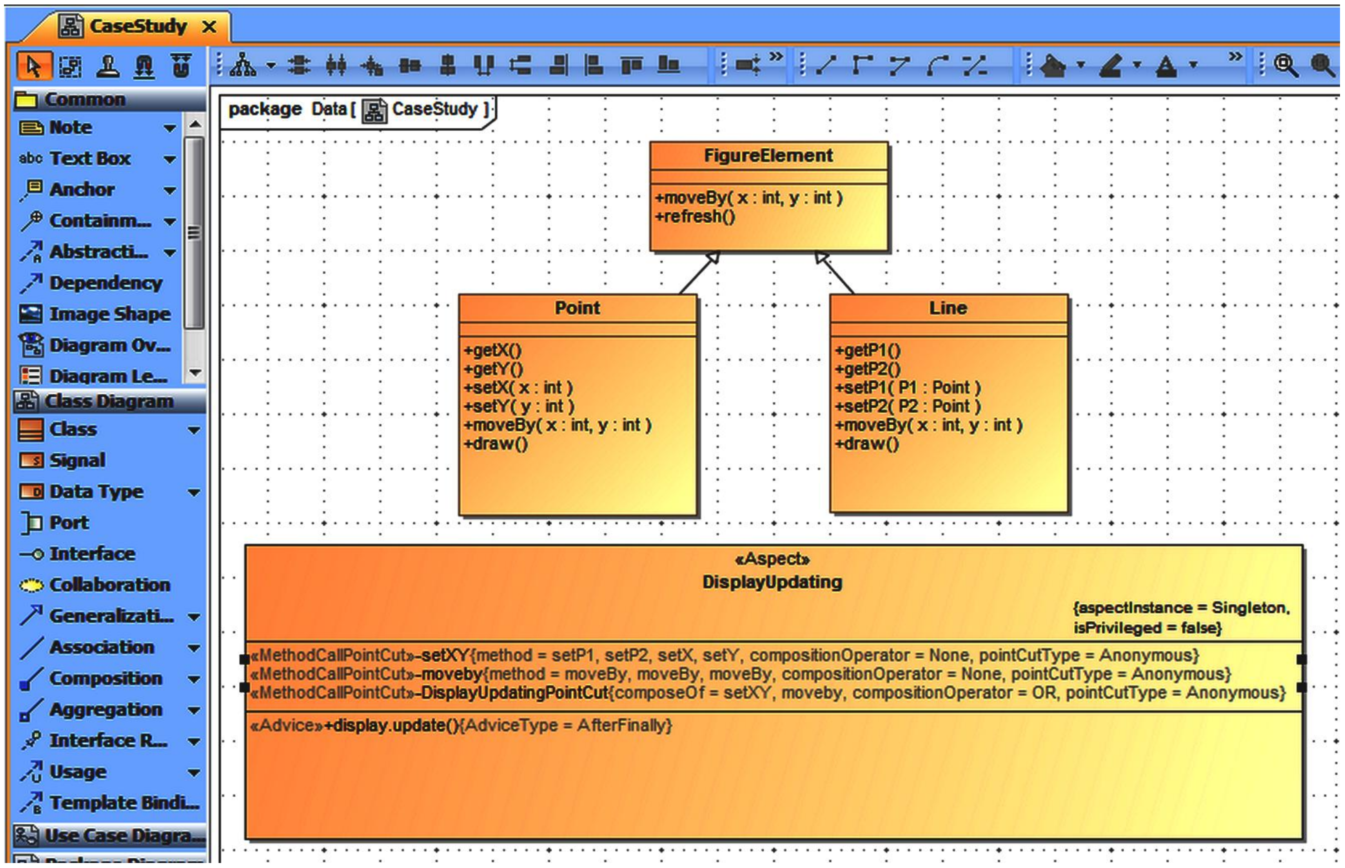
CaseStudy
Common
Note
Text Box
Anchor
Containm...
Abstracti...
Dependency
Image Shape
Diagram Ov...
Diagram Le...
Class Diagram
Class
Signal
Data Type
Port
Interface
Collaboration
Generalizati...
Association
Composition
Aggregation
Interface R...
Usage
Template Bindi...
Use Case Diagra...
package Data [CaseStudy]
FigureElement
+moveBy(x : int, y : int)
+refresh()
Point
+getX()
+getY()
+setX(x : int)
+setY(y : int)
+moveBy(x : int, y : int)
+draw()
Line
+getP1()
+getP2()
+setP1(P1 : Point)
+setP2(P2 : Point)
+moveBy(x : int, y : int)
+draw()
«Aspect»
DisplayUpdating
{aspectInstance = Singleton, isPrivileged = false}
«MethodCallPointCut»-setXY{method = setP1, setP2, setX, setY, compositionOperator = None, pointCutType = Anonymous}
«MethodCallPointCut»-moveby{method = moveBy, moveBy, moveBy, compositionOperator = None, pointCutType = Anonymous}
«MethodCallPointCut»-DisplayUpdatingPointCut{composeOf = setXY, moveby, compositionOperator = OR, pointCutType = Anonymous}
«Advice»+display.update(){AdviceType = AfterFinally}

Figure 3. The UML model after the application of the AspectJ profile.

REFERENCES

[1] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and G. W. Griswold, “An Overview of AspectJ,” Proc. Of ECOOP'01 the 15th European Conference on Object-Oriented Programming, Springer-Verlag London, UK, pp. 327-353.

[2] R. Laddad, “AspectJ in Action”, Enterprise AOP with Spring Applications. Manning Publications, Second Edition, 2009.

[3] M. Forgáč and J. Kollár, “Static and Dynamic Approaches to Weaving,” Proc. Of the 5th Slovakian-Hungarian Joint Symposium on Applied Machine Intelligence and Informatics, Poprad, Slovakia, January 25-26, 2007, pp. 201-210.

[4] Unified Modeling Language (UML), V2.4. http://www.omg.org/spec/UML/2.4/. [retrieved: September, 2013].

[5] I. Groher and T. Baumgarth, “Aspect-Orientation from Design to Code,” Proc. Of the Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design Workshop In conjunction with 3rd International Conference on Aspect-Oriented Software Development, Lancaster, UK, March 22-26, 2004, pp. 63-68.

[6] O. Aldawud, T. Elrad, and A. Bader, “A UML profile for aspect oriented modeling,” Workshop on Advanced Separation of Concerns in Object-Oriented Systems at OOPSLA2001, 2001, available at http://www.cs.ubc.ca/~kdvolder/Workshops/OOPSLA2001/submissions/26-aldawud.pdf.

[7] O. Aldawud, T. Elrad, and A. Bader, “UML Profile for Aspect-Oriented Software Development,” 3rd International Workshop on Aspect Oriented Modeling at AOSD 2003, Boston, Massachusetts, March 2003.

[8] S. Dominik, S. Hanenberg, and R. Unland, “A UML-based aspect-oriented design notation for AspectJ,” Proc. Of the 1st International Conference on Aspect-Oriented Software Development, AOSD 2002, University of Twente, Enschede, The Netherlands. ACM, April 22-26, 2002, pp. 106-112.

[9] M. A. Basch, “Incorporating Aspects into the Software Development Process in Context of Aspect-Oriented Programming”. UNF Theses and Dissertations, paper 112. University of North Florida, December 2012. Available at: http://digitalcommons.unf.edu/etd/112. [retrieved: September, 2013].

[10] J. Evermann, “A Meta-Level Specification and Profile for AspectJ in UML,” Journal of Object Technology,

Volume 6, no. 7, 2007, pp. 27-49, doi: 10.5381/jot.2007.6.7.a2.

[11] T. Gottardi, R. Aparecida, and V. V. Camargo, "A Process for Aspect-Oriented Platform-Specific Profile Checking," Proc. Of the 2011 international workshop on Early aspects (EA'11), Porto de Galinhas, Brazil, March 21-25, 2011, pp. 1-5, doi: 10.1145/1960502.1960504.

[12] J. U. Júnior, V. V. Camargo, and C. V. Flach, "UML-AOF, A Profile for Modeling Aspect-Oriented Frameworks," Proc. Of the 13th workshop on Aspect-Oriented Modeling (AOM'09), Charlottesville, Virginia, USA, 2009, pp. 1-6, doi: 10.1145/1509297.1509299.

[13] J. Evermann, A. Fiech, and F. E. Alam, "A Platform-Independent UML Profile for Aspect-Oriented Development," Proc. Of the Fourth International Conference on Computer Science and Software Engineering (C3S2E'11), Montreal, Canada, May 16-18, 2011, pp. 25-34.

[14] F. E. Alam, J. Evermann, and A. Fiech, "Modeling for Dynamic Aspect-Oriented Development," Proc. Of the 2nd Canadian Conference on Computer Science and Software Engineering (C3S2E-09), Montreal, Canada, May 19-21, 2009, pp. 109-113.

[15] V. Gowri, "Extending the UML metamodel to grant prop up for crosscutting concerns", International Journal of Advanced Research in Computer Engineering & Technology (IJARCET), Vol. 1, Issue 7, September 2012, pp. 193-198.

[16] MagicDraw Software: http://www.nomagic.com/products/magicdraw.html. [retrieved: September, 2013].

[17] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold, "Getting started with ASPECTJ," Communications of the ACM 44 (10), New York, NY, USA, 2001, pp. 59-65.