Ingeniería e Investigación
ISSN: 0120-5609
revii_bog@unal.edu.co
Universidad Nacional de Colombia
Colombia

Vidal Silva, C.; Saens, R.; Del Río, C.; Villarroel, R.
OOAspectZ and aspect-oriented UML class diagrams for Aspect-oriented software modelling (AOSM)
Ingeniería e Investigación, vol. 33, núm. 3, septiembre-diciembre, 2013, pp. 66-71
Universidad Nacional de Colombia
Bogotá, Colombia

OOAspectZ and aspect-oriented UML class diagrams for Aspect-oriented software modelling (AOSM)

OOAspectZ y diagramas de clase orientados a los aspectos para la Modelación Orientada a Aspectos (MSOA)

C. Vidal Silva[1], R. Saens[2], C. Del Río[3] and R. Villarroel[4]

ABSTRACT aspect-oriented

Regarding modularised software development, Aspect-oriented programming (AOP) identifies and represents individually crosscutting concerns during the software development cycle's programming stage. This article proposes and applies OOAspectZ to formal Aspect-oriented requirement specifications for prior stages of the software development cycle. It particularly concerns requirement specification and the structural design of data and behaviour, along with describing and applying Aspect-oriented UML class diagrams to designing classes, aspects and associations among classes and aspects during Aspect-oriented software development (AOSD).

OOAspectZ is a language integrating both Object-Z and AspectZ formal languages whereas Aspect-oriented UML class diagrams represent AOP code, object class and crosscutting concern class structure by means of stereotypes. This article shows and applies the main OOAspectZ and AO UML class diagram characteristics to Aspect-oriented software modelling (AOSM) using a classic example of AOP. Ideas for future work concerning an actual AOP version are also indicated.

Keywords: Aspects, OOAspectZ, UML class diagram, crosscutting concern.

RESUMEN

En la búsqueda de desarrollo del software modularizado, la Programación Orientada a Aspectos (POA) identifica y representa de manera separada funcionalidades cruzadas en la etapa de programación del ciclo de desarrollo del software. Para las etapas previas del ciclo de desarrollo del software, particularmente, en la especificación de requerimientos y el diseño estructural de los datos y comportamientos, este trabajo propone y aplica OOAspectZ para la especificación formal de requerimientos orientados a aspectos, además, describe y aplica diagramas de clases UML orientados en el diseño y la asociación entre clases y aspectos, para el proceso de Desarrollo del Software Orientado a Aspectos (DSOA), respectivamente.

Particularmente, OOAspectZ es un lenguaje que integra los lenguajes formales Object-Z y AspectZ, mientras que, los diagramas de clases UML orientados a aspectos representan la estructura del código de POA, clases de objetos y clases de funcionalidades cruzadas con el uso de estereotipos. Este artículo muestra y aplica las principales características de los lenguajes OOAspectZ y diagramas de clase UML orientados a aspectos, para la modelación del software orientado a aspectos (MSOA) que se aplican a un ejemplo clásico de POA, además, se entregan ideas de trabajo futuro respecto a una actual versión de POA.

Palabras clave: aspectos, OOAspectZ, diagramas de clase UML, e incumbencias cruzadas.

Received: February 26th 2013
Accepted: October 2th 2013

Introduction

According to Kiczales *et al*., (1997) "in system implementation, an aspect is a property that cannot be clearly encapsulated in a general procedure." Also, aspects cannot be modularised by means of traditional procedural or object-orientated (OO) methods (Kiczales, & Mezini, 2005). Aspects thus interleave a system's encapsulated methods, producing cross-cutting concerns. To solve these issues, AOP allows modularising aspects to enable modular reasoning in cross-cutting concerns. Typical examples of cross-

[1] Cristian Vidal Silva. PhD Computer Science student, Michigan State University, USA. Affiliation: Professor of Business Informatics Administration at the University of Talca, Chile. E-mail: cvidal@utalca.cl

[2] Rodrigo Saens. PhD in Economics from the University of Connecticut, USA. Affiliation: Professor of Business Administration at the University of Talca, Chile. E-mail: rsaens@utalca.cl

[3] Carolina Del Río. MSc in Organizational and Behavioral Development from Diego Portales University, Chile. Affiliation: Professor of Business Administration at the University of Talca, Chile. E-mail: cdelrio@utalca.cl

[4] Rodolfo Villarroel. PhD in Computer Science from the Universidad de Castilla-La Mancha at Ciudad Real, Spain. MSc in Computer Science from the Universidad Técnica Federico Santa María, Chile. Affiliation: Professor at the Escuela de Ingeniería Informática from the Pontificia Universidad Católica de Valparaíso, Chile. E-mail: rodolfo.villarroel@ucv.cl

How to cite: Vidal, C., Saens, R., Del Río, C., Villaroel, R., OOAspectZ y diagramas de clase orientados a los aspectos para la Modelación Orientada a Aspectos (MSOA)., Ingeniería e Investigación, Vol. 33, No. 3, December 2013, pp. 66 – 71.

cutting in a software system would be logging and security issues.

Aspects know which encapsulated elements should be advised, since (in classical AOP) each aspect explicitly declares which routines and classes must be advised and in which conditions, i.e. aspects know when and where they have to advise a system's encapsulated elements by means of a pointcut (PC) rule definition. A PC is a predicate defining join points (JP) between a system's elements and aspects (Kiczales, & Mezini, 2005). For a given JP, aspects can use three kinds of advice: before, after and around. Aspect behaviour (advice) is added to or replaces encapsulated behaviour (Kiczale *et al*., 1997; Kiczales, & Mezini, 2005). Regarding software applications, aspects can change their base code behaviour.

Even though AOP enables modularising elements which could not be modularised previously by other software development methodology, as indicated by Bodden *et al*., (2013) a few issues concern modularising software using classic AOP and join point interfaces (JPI) has been proposed (Bodden *et al*., 2013) for solving such issues.

This article looks at using classic Aspect-oriented (AO) principles in previous phases of software development. Its main goals were to propose a formal specification language, OOAspectZ, and model a classic AOP system by using OOAspectZ to describe the pros and cons of this formal language proposal. It adapts a previous proposal for AO UML class diagrams, attempting a clearer definition of PC units.

```
interface Shape {
        public moveBy(int dx, int dy);
}

class Point implements Shape {
 int x, y; //intentionally package public

 public int getX() { return x; }

 public int getY() { return y; }

 public void setX(int x) { this.x = x; }

 public void setY(int y) { this.y = y; }

 public void moveBy(int dx, int dy) {
  x += dx;
  y += dy; }
 }
}

class Line implements Shape {
 private Point p1, p2;

 public Point getP1() { return p1; }

 public Point getP2() { return p2; }

 public void moveBy(int dx, int dy) {
 p1.x += dx; p1.y += dy;
 p2.x += dx; p2.y += dy; }
}

aspect UpdateSignaling {
  pointcut change(): execution(void Point.setX(int))
        || execution(void Point.setY(int))
        || execution(void Shape+.moveBy(int, int));

  after() returning: change() {
                Display.update();
  }
}
```

Figure 1. Painting system AspectJ source code

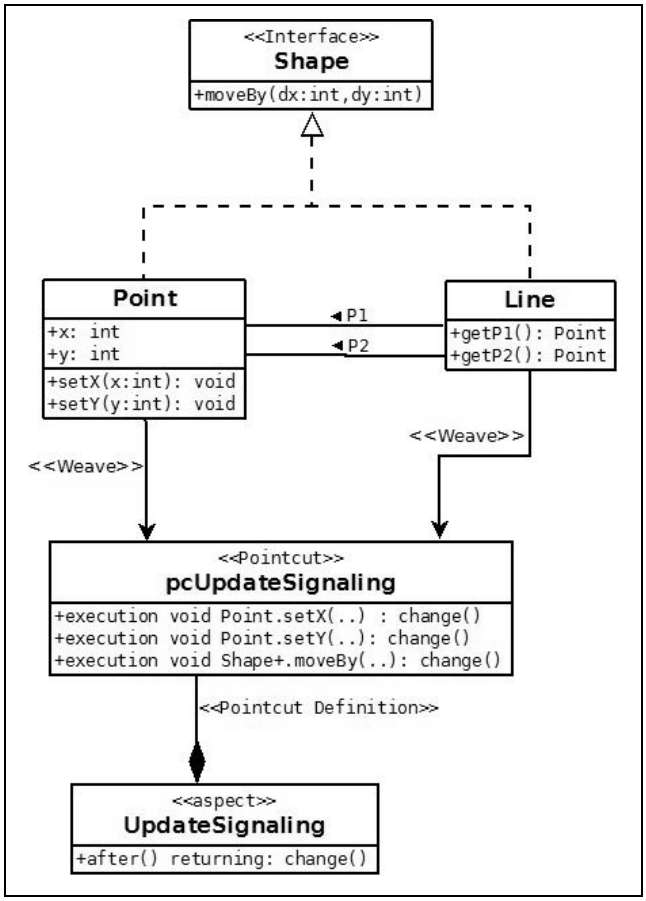

Figure 2. Painting system AO UML class diagram and AspectJ source code

Aspect-oriented modelling

Several articles have dealt with UML extensions to support AO modelling (AOM). For example, Bustos, & Eterovic (2007) and Liu, & Chuang-Wen (2008) have applied UML class diagrams to AO modelling and Wimmer *et al*., (2011) have surveyed UML and AOM. Using an updated version of the modelling form presented by Liu, & Chuang-Wen (2008), Figure 1 shows a painting system AspectJ source code (Kiczales & Mezini, 2005), a classic example of AOP, whereas Figure 2 shows a painting system AO UML class diagram.

The AOM form presented by Liu and Chuang-Wen (2008) allows modelling the main AOP elements, such as advised classes by aspects, and aspects. However, elements for defining PC units are not considered by such AOM form. Figure 2 shows that aspects should consist of PC units and there are directed associations stereotyped by <<Weave>> from classes to PC units, even though a PC defines a criterion for advising its referenced instances of classes, i.e., association between classes and PC units should be opposed. However, using weaving, means that association direction would be correct since final waived classes incorporate code of aspects.

Figure 2 gives an extended version of Liu and Chuang-Wen (2008)'s proposal for modelling the AOP code in Figure 1. The original proposal's ideas are preserved and this extended version only defines PC using ideas concerning AspectJ such as a method's execution and call. These extensions look for a detailed PC definition to enable a high degree of transparency between the code's structural elements and the structural model of the code, i.e., easy

translation of structural models to the main elements of AOP programme code (aspects and classes) and vice-versa, as shown in Figures 1 and 2.

Painting system key objects are instances of shape, point and line, and display. This scenario concerns an abstract shape class, and concrete point and line classes, shape subclasses. There is a single display class and, for simplicity, just a single system-wide display operation: Display.update(), performed by aspect UpdateSignaling.

Figure 2 shows a class having the <<Pointcut>> stereotype to identify PC elements of classes identified by the <<aspect>> stereotype. These classes are named <<Pointcut>> and <<aspect>> classes. Therefore, given a set of classes attended by aspects, <<Pointcut>> classes relate advised classes to <<aspect>> classes. Thus, there can be more than one <<Pointcut>> and <<aspect>> classes for a model, even though a <<Pointcut>> class allows defining different pointcut elements for the same <<aspect>> class.

A behavioural view of the system highlights an important consideration; <<Pointcut>> and <<aspect>> classes represent global instances which are active when join points occur.

Khatchadourian and Soundarajan (2007) have argued that there are similarities between classic AOP and concurrent / distributive programming.

AO formal modelling

Regarding AO formal modelling, a few applications of formal methods would include Alloy (Mostefaoui & Vachon, 2007) and AspectZ (Yu, Liu, Yang & He, 2005) (Vidal, Saens, Del Río & Villarroel, 2013) and an extension of the formal language Z (Woodcock, & Davies, 1996) for OOSD ideas.

Since AOSD tries to reduce cross-cutting concerns in object-orientated software development (OOSD) and Object-Z (Duke, & Rose, 2000) (Smith, 1999), there should be integration between Object-Z and AspectZ. This article thus proposes OOAspectZ which integrates Object-Z and AspectZ. Figure 3 presents the main elements of OOAspectZ while Figures 4, 5, 6, 7, 8 and 9 present associated OOAspectZ diagrams for the painting system.

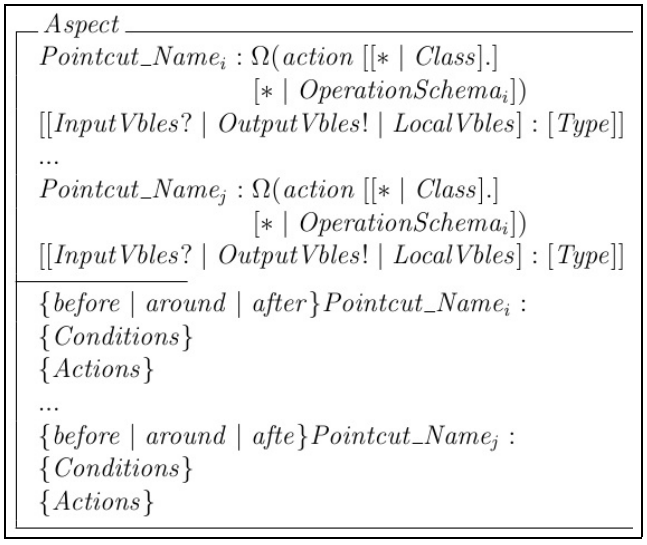

Figure 3. OOAspectZ aspect-schema

A traditional class diagram is used as in UML class diagrams, for representing an interface in Object-Z. Thus, classes implementing an interface define public attributes and rules for interface methods. Figure 4 gives an OOAspectZ model of the interface shape for a painting system, while Figures 5 and 6 show point and line painting system classes involving the shape properties interface.

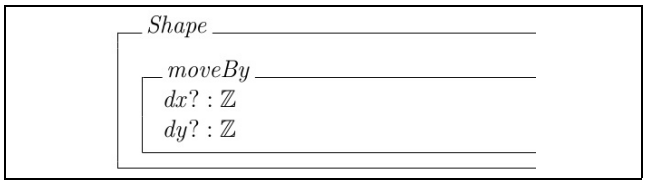

Figure 4. Painting system interface shape schema

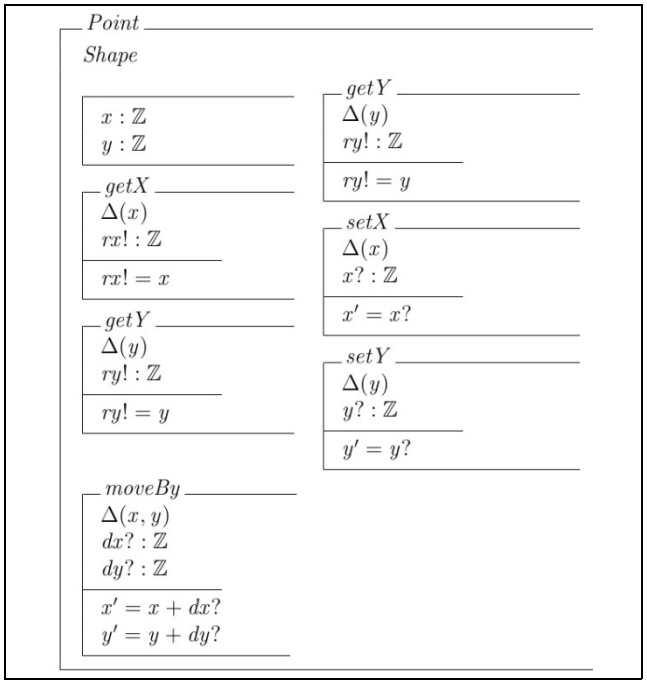

Figure 5. Painting system object-Z class point schema

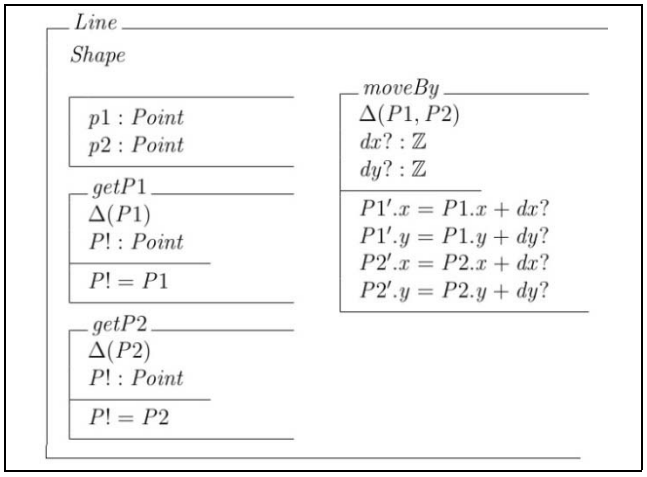

Figure 6. Painting system object-Z class line schema

Figure 5 gives an Object-Z diagram for the point class implementing properties, i.e., methods and attributes, from the shape class. The first element in class is the state, defining class attributes x and y. Invariant rules can be defined in such schema for instances of class. This figure gives schema for modelled class operation or methods. As shown, method moveBy updates attributes x and y for that object for a point class.

Figure 6 shows an Object-Z schema for line class inheriting properties from shape class. Line consists of two point class objects (p1 and p2) and a method obtains p1 and p2 line values This Figure gives a complete definition of moveBy inherited from the shape class which is different from the point class moveBy method. moveBy updates values for fields x and y of p1 and p2 concerning input values. Considering that Object-Z allows substituting predicate and schema elements, Figure 7 shows the schema for moveBy using substitutions.

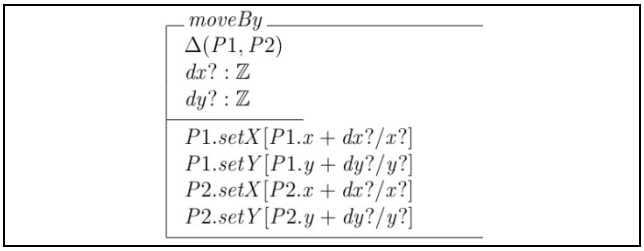

Figure 7. Painting system Object-Z MoveBy method for line class with substitutions

Figure 8 shows the OOAspectZ aspect-schema for UpdatingSignaling. Clearly, this shows the elements present in the AspectJ source code in Figure 1. A direct translation from OOAspectZ specification to AspectJ code is thereby completely supported by this example. Such compatibility between domain and application models is a fundamental principle for model-driven engineering (MDE), a particular software development methodology (Mellor & Balcer, 2002).

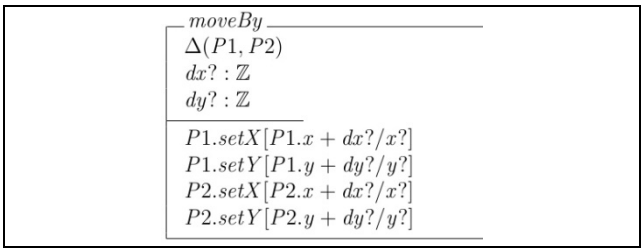

Figure 8. Painting system OOAspecZ aspect-schema update signal

The implicit announcement and invocation of aspects are basic elements of classic AOSD. A formal representation of AOSD elements is not a direct task for current modelling languages. AO solutions are abstractions of OO solutions in which a 'weaver' controls transforming AO code into the associated OO code. For a formal representation of AO models, a potential solution would be to apply the AO 'weaver' in previous phases of the software lifecycle to obtain OO models which are formally representable by first-order logic predicates.

Since elements of a Z specification are translatable into first-order logic predicates (Zave & Jackson, 1993) and taking into account previous use of AOSD for weaving, woven OOAspectZ / Object-Z schemas are thus representable by means of first-order predicate logic. Figure 9 shows the woven schema setX. for the point class.

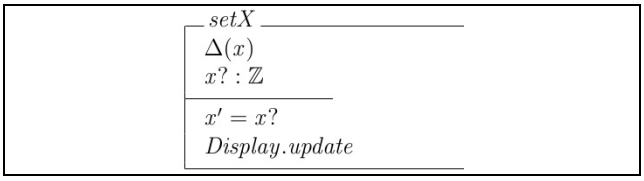

Figure 9. Painting system woven OOAspectZ/Object-Z SetX schema

According to Zave & Jackson (1993), Z operation schemas are representable as event types in a state-transition model. Primed and unprimed elements for an operation schema represent the object state before and after a current event occurrs. The following predicates would be established by the method schema setX:

- Since a Z schema is translated into an individual type predicate, a predicate setX(e) represents the occurrence of event e as an instance of operation setX;
- Because each argument in an operation schema is represented by a predicate, there is predicate x?(num, e) for setX which means that num is the integer argument of event e for affected attribute x;
- Regarding the non-temporal properties of Z language, there are unary predicates for each basic type (Zave & Jackson, 1993), Z in this case for integer numbers. Due to such predicate, there is the following uniqueness restriction:

$$\forall e(setX(e) \Rightarrow \exists! num(x?(num,e) \wedge Z(num)),$$

- Since setX schema contains notation Δ(x), indicating that operation setX changes the value of attribute x of the current instance, a setX operation would have predicate Δx(x, e) meaning that x is the object point coordinate applied to the event; and
- According to the point class state schema, x is an attribute of this class. Therefore, because operation setX represents event e, when such event occurs there would be a new state of the current point class instance. As previously shown, there are predicates Δx(x, e) and x?(num, e). Thus, to know the effects of the setX method on the current point class object, the relationship between the values for attribute x of two consecutive states must be shown. According to Zave & Jackson (1993), a predicate x(y, v) represents value y for x in state v which must be equal to the set value, num in this case:

$$\forall e \forall v \forall x \forall num \forall y(begin(e,v) \wedge setX(e) \wedge$$
$$\Delta(x,e) \wedge x?(num,e) \wedge x(y,v)) \Rightarrow num = y$$

Since Zave & Jackson (1993) have indicated rules for representing Z schemas as first-order logic predicates, this article tried to adjust such rules to OOAspectZ. These new rules should lead to defining operations and methods affecting class object attributes. Since formal modelling involves reasoning about software requirements more profoundly than modelling software requirements informally, a complete review and adaption of Zave & Jackson rules for OOAspectZ, along with their application for a complete OOAspectZ model, form part of the authors' future research.

Classic AOSD distinguishes between primary (base) and crosscutting concerns. In an AOSD scenario, instead of generating a woven OO solution and defining rules for woven elements, adapted Zave & Jackson rules can be defined for each base module and aspects of a system, such as adapted Object-Z specification rules. The authors will formalise integrating base modules and aspects in OOAspectZ specifications without weaving by means of first-order logic predicates in their future work. Preconditions should be defined for the associated first-order logic predicates for each OOAspectZ specification schema and aspect-schema, because Zave, & Jackson have stated that as schemas are types, predicates for schemas are able to include references to previous and after schema. Therefore, for each schema advised by an aspect-schema, guards for correct interaction order must be included. For example, if an operation schema has previous aspect-schema

A, the operation schema must indicate A as part of its preconditions; aspect-schema A is seen as a previous event. Regarding a formal definition of aspect-schemas, each presents the pointcut as its precondition. These ideas can be extended to after aspect-schema. Around aspect-schemas behave like before aspect-schemas whose next state is the attended schemas for an action to proceed, or the following schema for the attended one, when there is no action to proceed.

Regarding a complete aspect-schema definition, considering aspect-schemas for setting and obtaining the value of attributes of a class instance, integrating aspect-schemas with traditional Object-Z schemas and defining associated first-order logic predicates for this kind of aspect-schema constitute future research work into OOAspectZ.

Even though AOSD allows separating concerns, there are situations in which applying the obliviousness principle generates complex situations thereby restricting complete modularisation. Even though Sullivan *et al*., (2005) defined obliviousness for designing base elements without worrying about aspect functionality, base elements avoid cross-cutting concerns; Sullivan, Griswold, Song & Cai (2005) have mentioned a few common complex situations for AO programmers:

- 'Private join points' for a tight link between base and aspects code (in such situation, changes in base code can imply no more join points for the aspect's action);
- 'State-point separation' for aspects in a defined module reacting for the initialisation of defined variables when those variables are initialised in different modules;
- 'Inaccessible join points' for switching and nested conditional statements which do not allow a join point to be accessible; and
- 'Quantification failure' for a non-updated pointcut, although there are changes in the base code.

As Sullivan, Griswold, Song & Cai (2005) indicated, since obliviousness is an AOP principle, there are situations in which more attention should be paid to the specific abstract state and behaviour of the application and not just on concrete event execution. Thus, more attention should be paid to software lifecycle design phases.

The authors of this article faced an additional AOSD issue regarding aspect definitions for instances of aggregated classes, instances of classes in an aggregation or composition association. Aspects advise aggregated instances having access to their attributes in such situations and attributes of the class to which instances are the whole. For example, if class A consists of sets of class B and C objects, one set for each class, and aspects advise instances of B, thus aspects can also update a set of elements in C. The authors called such situation 'total access for aggregated instances'.

Another AOP issue concerns aspect composition, or aspects advising other aspects. Defining this association among aspects is neither simple nor direct in classic AOP. JPI has been used for resolving aspect composition (Bodden *et al*., 2013).

Since aggregation is an essential OOSD element, above all in OOP, the previously mentioned situations represent an overall AOSD composition issue. One solution to the 1st composition issue is to define aspects for the composed class instead of for the classes forming part of the whole class. A formal review of the 'total access for aggregated instances' problem along with an analysis of solutions for that problem form part of the authors' future research.

Undoubtedly, a formal AO method, such as OOAspectZ, allows better modularisation since base class schema are defined by complete separation of concerns. Even though OOAspectZ specification class schemas include only their base concerns, OOAspectZ specification aspect-schemas can include complex issues such as the 'total access for aggregated instances'.

Conclusions

AOM allows complete isolation of software system cross-cutting functionalities into separate and independent entities called aspects;

AOM for requirements and design allows more complete AOSD since OOAspectZ enables defining AO formal requirement specifications and UML class diagrams facilitate AO design;

Given the simplicity of Z languages, such as Z and Object-Z, the proposed OOAspectZ formal language allows capturing the essence of those languages along with modularisation benefits regarding classic AOP to identify and isolate (i.e., modularise) software cross-cutting concerns; and

Using UML class diagrams and OOAspectZ, this paper has presented the main AOM idea as well as illustrating the advantages of these languages for modularising cross-cutting concerns. Regarding UML class diagrams, this article gives a new version of Liu, & Chuang-Wen (2008) using AspecJ notation to facilitate a design for coding translation.

Related and future work

Future work will present first-order rules for representing OOAspectZ schemas and applying such rules to model a complete case study as well as reviewing the integration of traditional Object-Z and OOAspectZ schemas for defining woven schemas.

The authors wish to tackle the 'total access for aggregated instances' issue in detail as well as ways of resolving it.

The authors will continue working towards a full definition of OOAspectZ for applying this Aspect-oriented formal language to specify a case study such as those analysed by Steimann et al., (2010).

Even though classic AOP allows modularising cross-cutting concerns which have not been able to become modularised by other software development methodology, such as object-orientated and structure programming, Bodden et al., (2013) have indicated that classic AOP introduces implicit dependency between aspects and advised routines. This would mean that if an advised routine should change its signature, regardless of the magnitude of signature change, aspects advising that routine could not advise more without changing their PC. A routine is usually advised without expecting change in its behaviour. Aspects can access, including private elements of an advised routine associated class. Bodden et al., (2013) proposed JPI for establishing an interface among aspects and advised classes' routines. Extending our current proposal regarding OOAspectZ formal specification language to support JPI modularisation ideas, JPI-AspectZ, forms part of our future work.

Following JPI ideas, the authors' current work is aimed at modelling JPI programme structure and behaviour for transformation transparency among models and code and high-level understanding of the JPI code.

References

Bodden, E., Tanter, E., Inostroza, M., A Brief Tour of Join Point Interfaces., In: proceedings of the 12th Annual International Conference Companion on Aspect-Oriented Software Development, AOSD'13 Companion, Fukuoka, Japan, 2013, pp. 19-22.

Bustos, A., Eterovic, Y., Modeling Aspects with UML's Class, Sequence and State Diagrams in an Industrial Setting., In: proceedings of the 11th IASTED International Conference on Software Engineering and Applications, SEA '07, 2007, pp. 403-410.

Duke, R., Rose, D., Formal Object-Oriented Specification Using Object-Z., 1st edition, London, MacMillan Press Limited, 2000.

Khatchadourian, R., Soundarajan, N., Rely-Guarantee Approach to Reasoning about Aspect-Oriented Programs., In: proceedings of the 5th Workshop on Software Engineering Properties of Languages and Aspect Technologies, SPLAT '07, ACM, New York, NY, USA, 2007.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J. C., Irwin, J., Aspect-oriented programming., In: proceedings of European Conference of Object-Oriented Programing, ECOOP97, Springer Verlag, 1997, pp. 220-242.

Kiczales, G., Mezini, M., Aspect-Oriented Programming and Modlar Reasoning., In: proceedings of the 27th International Conference on Software Engineering, ICSE '05, ACM, New York, NY, USA, 2005, pp. 49-58.

Liu, C., Chuang-Wen, C., A State-Based Testing Approach for Aspect-Oriented Programming., Journal of Information Science and Engineering, Vol. 24, 2008, pp. 11-31.

Mellor, S. J., Balcer, M., Executable UML: A Foundation for Model-Driven Architectures., Boston, MA, USA, Addison-Wesley Longman Publishing Co. Inc., 2002.

Mostefaoui, F., Vachon, J., Verification of Aspect-UML Models Using Alloy., In: proceedings of the 10th International Workshop on Aspect-Oriented Modeling, AOM '07, ACM, New York, NY, USA, 2007, pp. 41-48.

Smith, G., The Object-Z Specification Language., Vol. 1 of Advances in Formal Methods., 5th Ed., Springer US, Software, 1999.

Steimann, F., Pawlitzki, T., Apel, S., Kästner, C., Types and Modularity for Implicit Invocation with Implicit Announcement, ACM Transaction on Software Engineering an Methodology., TOSEM, Vol. 20, No. 1, July, 2010, pp. 1-43.

Sullivan, K., Griswold, W. G., Song, Y., Cai, Y., Information Hiding Interfaces for Aspect-Oriented Design., In: ESEC/FSE-13: Proceedings of the 10th European software Engineering Conference, 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2005, pp. 166-175.

Vidal Silva, C., Saens, R., Del Río, C., Villarroel, R., Aspect-Oriented Modeling: Applying Aspect-Oriented UML Use Cases and Extending AspectZ., Computing and Informatics Journal, Bratislava, Slovak, 2013, pp. 573-593.

Wimmer, M., Schauerhuber, A., Kappel, G., Retschitzegger, W., Schwinger, W., Kapsammer, E., A Survey on UML-Based Aspect-Oriented Design Modeling., ACM Computing Survey, Vol. 43, No. 4, Oct., 2011, pp. 1-28.

Woodcock, J., Davies, J., Using Z: Specification, Refinement, and Proof., Upper Saddle River, NJ, USA, Prentice-Hall, Inc., 1996.

Yu, H., Liu, D., Yang, L., He, X., Formal Aspect-Oriented Modeling and Analysis by AspectZ: An Aspect-Oriented Modeling., 17th International Conference on Software Engineering and Knowledge Engineering (SEKE05), Taipei, Taiwan, 2005.

Zave, P., Jackson, M., Conjunction as Composition, ACM Transactions on Software Engineering and Methodology., Vol. 2, 1993, pp. 379-411.