Formal Specification Language Jpiaspectz: Looking For A Complete JPI Software Development Process

Cristian Vidal-Silva, Claudia Jiménez, Erika Madariaga, Luis Urzúa

Abstract: Aspect-Oriented Software Development AOSD solves modularity issues in the Object-Oriented Software Development OOSD approach. AOSD adds a few more details concerning the dependency between related modules. Join Point Interface JPI represents an AOSD methodology to solve those AOSD issues by the definition of interfaces in the middle of advisable artifacts and aspects. JPI permits developing software modules without crosscutting concerns. Looking for a JPI software development approach, this article proposes and exemplifies the use of JPIAspectZ, an extension of the formal aspect-oriented language AspectZ for the requirement specification of JPI solutions. Mainly, JPIAspectZ looks for a consistent JPI software development process. Defining join point interfaces represents a primary JPI component for explicitly associating aspects and advised modules. Classes are no longer oblivious of possible interaction with aspects, and effectiveness of aspects no more depend on signatures of advisable modules components for the use of JPI instances. JPIAspectZ fully supports these JPI principles. As JPI application examples, this article shows the formal requirements specification, structural model, and JPI code for a typical aspect-oriented application.

Index Terms: Aspects, Concerns, Formal Modeling, Join Point Interface, JPI, JPIAspectZ, Modularity.

1. INTRODUCTION

Aspect-Oriented Software Development (AOSD) permits modularizing crosscutting concerns in Object-Oriented Software Development (OOSD) stages [1]. Because AOSD was born at the Object-Oriented (OO) programming stage, to reach a complete transparency of concepts and design in the AOSD process seems a complicated task. Looking that transparency in the AOSD process, different proposals of modeling language extensions already exist to support AOSD such as aspect-oriented UML use case diagrams [2] and aspect-oriented UML class diagram [3]. Specifically, Wimmer et al. [4] present a survey of aspect-oriented UML languages. Nevertheless, only a few articles about formal aspect-oriented languages for requirement specification proposals exist so far; for example, Yu et al. [5]; Vidal et al. [6] describe and apply AspectZ, the works of Vidal et al. [7-8] describe OOAspectZ, and Mostefaoui and Vachon [9] illustrates the use of an AO Alloy version.Besides, Bodden et al. [10] indicate that, in traditional AOSD solutions, a double-dependency between base modules and aspects exists. To solve this issue, the works of [10-12] propose the use of Join Point Interface (JPI) instances between classes and aspects. Thus, with the purpose of obtaining JPI solutions and getting transparency of concepts in stages of the AOSD-JPI process, this article proposes and applies JPIAspectZ, an extension of OOAspectZ [7-8] for requirements specification of JPI software applications.

- *Cristian Vidal-Silva, professor at Departamento de Administración, Facultad de Economía y Negocios, Universidad Católica del Norte, Antofagasta, Chile. E-mail: cristian.vidal@ucn.cl*
- *Claudia Jiménez, director of Ingeniería Civil Informática, Facultad de Ingeniería y Negocios, Universidad Viña del Mar, Viña del Mar, Chile. E-mail: cjimenez@uvm.cl*
- *Erika Madariaga, director of Ingeniería Informática, Facultad de Ingeniería, Ciencia y Tecnología, Universidad Bernardo O'Higgins, Santiago, Chile. E-mail: erika.madariaga@ubo,cl*
- *Luis Urzúz, profesor at Escuela de Kinesiología, Facultad de Salud, Universidad Santo Tomás, Talca, Chile. E-mail: lurzua@santotomas.c*

This paper structures is as follows: Section 2 signals main properties of Aspect-Oriented Programming (AOP) paradigm and its JPI extension. Section 3 describes the main properties of Z, Object-Z, and Aspect-Z formal languages to introduce and describe the main properties of JPIAspectZ. Section 4 presents a few applications of the JPIAspectZ formal requirement specification language over a classic aspect-oriented case study and a JPI case study. Section 5 evaluates JPIAspectZ consistency. Conclusions finally concludes and presents future work ideas.

2 ASPECT-ORIENTED PROGRAMMING AND JPI

Kiczales et al. [1] proposed Aspect-Oriented Programming (AOP) to modularize crosscutting concerns as aspects in OOP. Aspects advise classes like events, that is, aspects introduce behavior and structural elements such as methods and attributes into classes. Nevertheless, as Bodden et al. [10] indicate, AOP presents implicit dependencies between advised classes and aspects. First, aspects define Pointcut Rules (PCs) for advisable classes' behavior; and, as a result, instances of those classes are entirely oblivious of possible changes in their components, methods, and attributes. Second, aspects can be ineffective or spurious for signature changes on advised methods of target classes. Such as [10-11] mention, the last issue is known as the fragile pointcut problem. Likewise, Bodden et al. [10] also indicate that traditional AOP like Aspect-J solutions compromise the independent development of base code and aspect modules since developers of base code, and aspects must obtain a global knowledge about all program components and their associations, that is, they must know all the details about aspects, classes, and their relations.To isolate crosscutting concerns and get modular AOP programs without the mentioned implicit dependencies, the work of Bodden et al. [10] describe the JPI programming methodology. JPI introduces the idea of join point interface on classic AOP. Like classic AOP [10-11], for JPI applications, aspects represent crosscutting functionalities, but without PCs. Aspects in JPI only present their implementation of join point interfaces. Besides, in JPI, non-oblivious advised classes exhibit explicit join point interfaces, that is, classes know about potential

change s on their methods. Figures 1 and 2 [10] illustrate dependencies between aspects and classes in classic AOP and JPI applications, respectively.

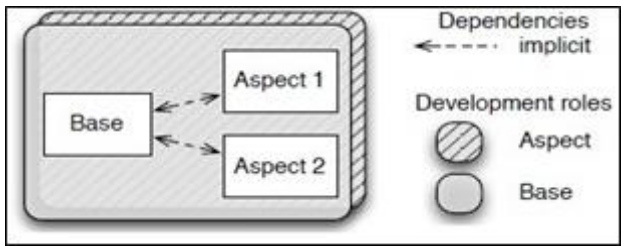


***Fig. 1**. Associations of base and aspects modules in classic AOP.*

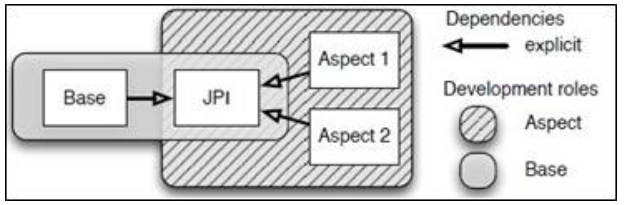
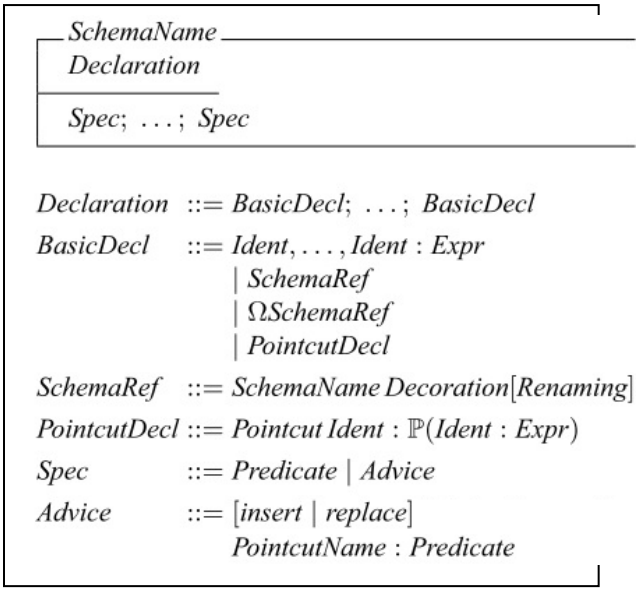


***Fig. 2**. Associations of base and aspect modules with a join point interface in JPI.*

3 JPIASPECTZ FUNDAMENT

Z [13] and Object-Z [14] are formal languages for software requirements specification. Specifically, Z is the classic formal specification language without the direct support of object-oriented abstractions like classes and inheritance, and Object-Z is an extension of the famous Z to support OOSD principles. Likewise, AspectZ [5-6] and OOAspectZ [7-8] represent Z extensions for requirements specification of AOP applications and their integration with Z and Object-Z, respectively. Figures 3 and 4 show the schemas specification for AspectZ and OOAspectZ. Considering JPI ideas, this article describes JPIAspectZ, an OOAspectZ extension to model JPI applications and its integration with Object-Z. The next lines present the main elements of a JPIAspectZ formal specification.

- Base Modules: Unlike AspectZ and OOAspectZ which present oblivious base modules, JPIAspectZ base modules are specified as Object-Z class modules which include an exhibits rule con-cerning advisable operations of advised class instances. Figure 5 shows the structure of a JPIAspectZ class schema, JPI schema, and Aspect schema.



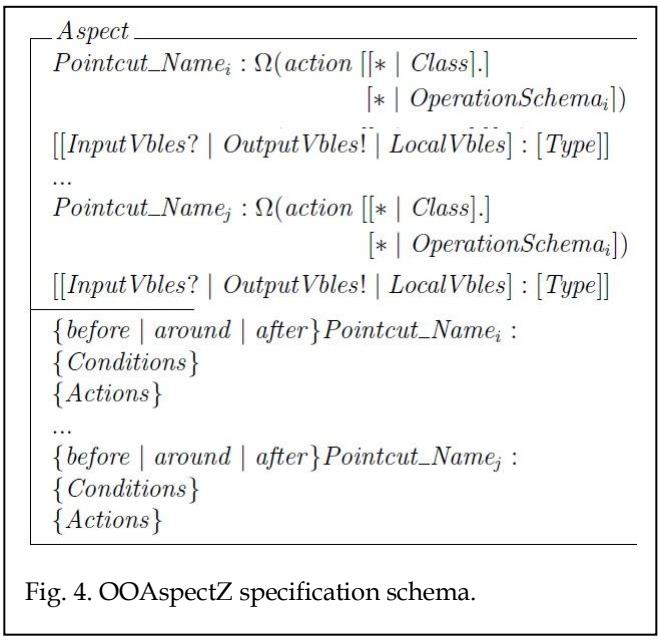


Fig. 4. OOAspectZ specification schema.

- Since the declaration part of an Object-Z operation schema permits defining operation parameters, when looking for a transparency of concepts and design for JPI applications, an exhibits rule is definable in two sections: first, exhibits JPI for the join point interface instances which the class exhibits, and second, a set of conditions for the join point event. So far, JPIAspectZ considers basic AOP and JPI conditions for dynamic and static crosscuts, that is, call operation; execution operation; logic connectors &&, ||, !; args(arguments list) to identify the arguments of catchable methods; this(object) to determine the object on which the advisable method operates; and target(object) to identify the object owner of the advisable method.

- Join Point Interface: In JPIAspectZ, operation schemas starting with the JPI initials represent join point interfaces (JPI schemas) for a system specification. For example,

Figure 8 shows JPIUpdateX and JPIUpdateY. Furthermore, JPI schemas only present a declaration section to indicate their list of parameters.

- Aspects: JPIAspectZ Aspects-schemas are like Object-Z class diagrams labeled with the phrase aspect. Aspect-schemas include state schemas to define attributes and invariants and operation schemas for the schema advice operations. As a distinction regarding class schemas, Aspect-schemas can indicate the occurrence time for operations (before, after, and around) to specify kind of advice. Semantically, aspect-schemas advise operation schemas, usually for inserting new methods in the advised classes, for adding behavior at the beginning, around, and end on advised operations schemas.

- From advised method schemas and associated aspect-schemas, JPIAspectZ permits obtaining woven schemas. It is relevant to highlight the modular evolution from AspectZ (Figure 3 [5]), OOAspectZ (Figure 4 [7-8]), and JPIAspectZ (Figure 5) schemas respectively. Note that for the first two, base schema, Z operation schema, and Object-Z class schema, an aspect operates over oblivious advised elements. Nevertheless, for the JPI philosophy, in JPIAspectZ, aspects, and classes know about interfaces to implement and exhibit, respectively.

3 Application Examples

The Painting System [7], is a classic AO example that presents classes Point and Line which are Shapes, and each Line instance is composed of a few Point instances. The main idea is to illustrate the updating screen process as an external behavior. Figure 7 illustrates a JPI UML class diagram that includes main JPI elements for the Painting System. Clearly, for exhibits and implements rules, classes are not more oblivious and aspect not directly refer to classes: classes exhibit JPI and aspects implement those interfaces. We recommend to review [15] for more details about JPI.

As a JPI example, [10] [12] show a Shopping session 'running example' of an e-commerce system (ShoppingSession system). That example presents a join point interface checkingOut, a class ShoppingSession that exhibits checkingOut and an aspect Discount that implements checkingOut for around kind of advice.

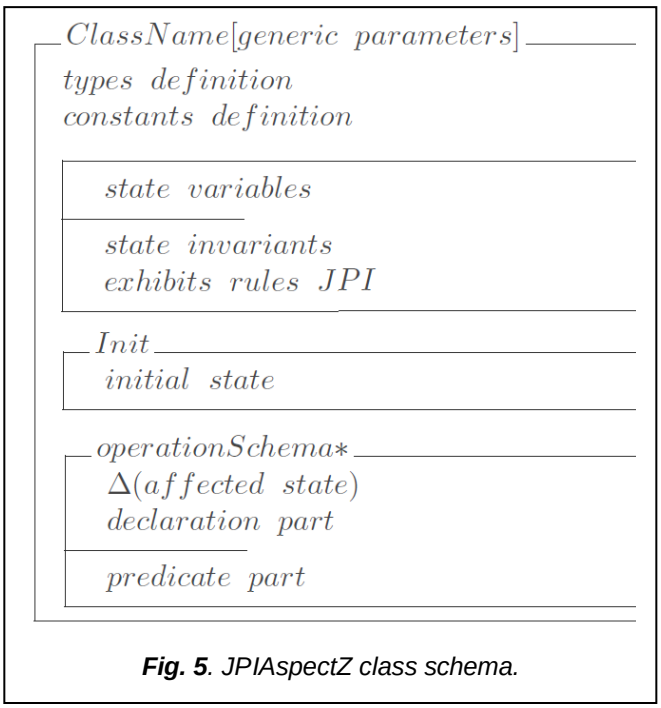


***Fig. 5**. JPIAspectZ class schema.*

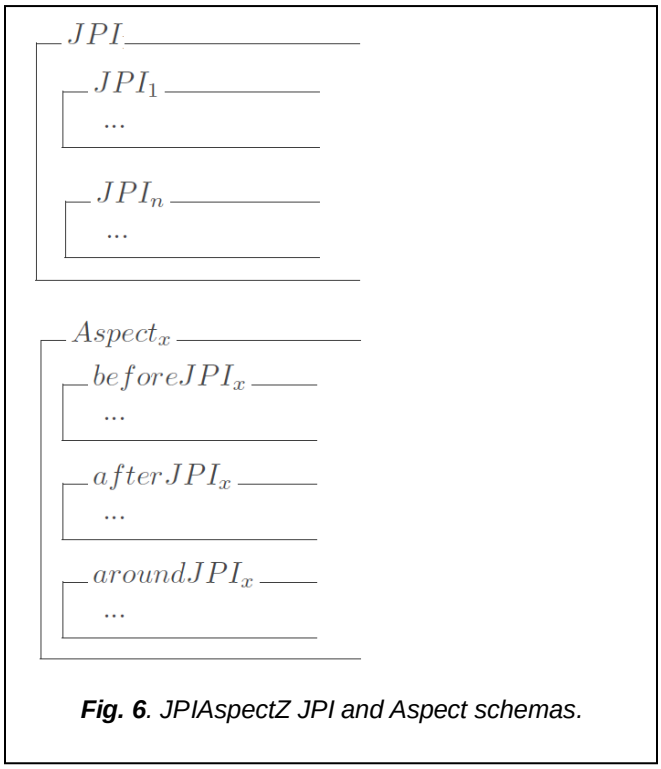


***Fig. 6**. JPIAspectZ JPI and Aspect schemas.*

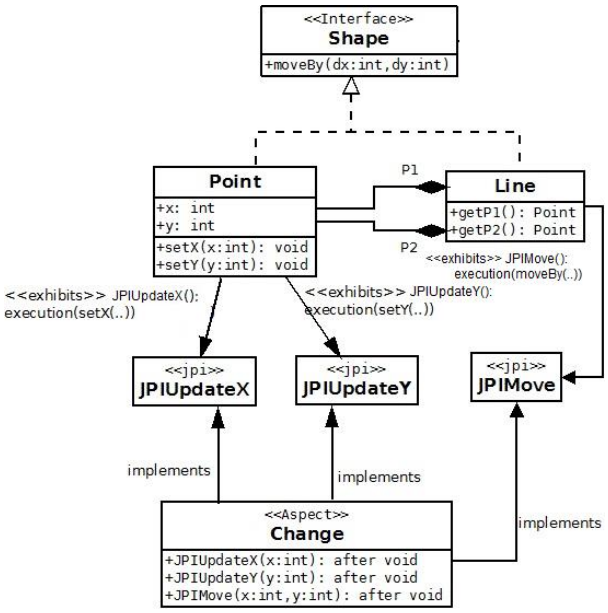


***Fig. 7**. Painting system JPI UML class diagram.*

4 APPLICATION RESULTS

Figures 8, 9, 10, and 11 present the JPIAspectZ specification for the Painting system: a Shape interface; Point and Line classes; the JPI instances JPIUpdateX, JPIUpdateY, and JPIMove; and aspect Aspect1Painting. Point and Line classes exhibit JPI instances, class Point exhibit JPIUpdateX and JPIUpdateY, and class Line exhibit JPIMove; whereas Aspect1Painting implements these JPI instances. A consistency exists among Figures 7 (JPI class diagram) and Figures 8, 9, 10, and 11 (JPIAspectZ specification) for the modeling of the Pinting system.

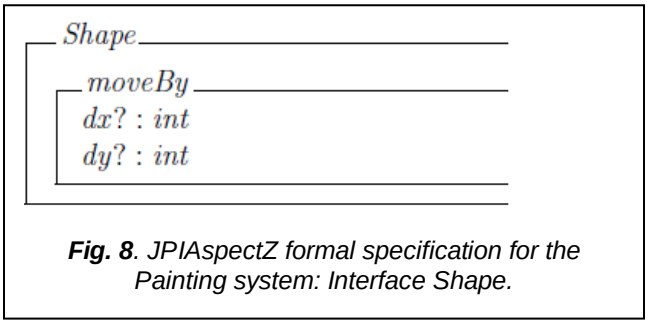


***Fig. 8**. JPIAspectZ formal specification for the Painting system: Interface Shape.*

Figure 9 presents a JPI programming code and Figure 11 shows a JPIAspectZ formal specification for the ShoppingSession system. Again, a clear consistency exists between these two figures. Figure 12 presents the JPI code for the same application to demonstrate a full consistency between the JPI model and implementation code.

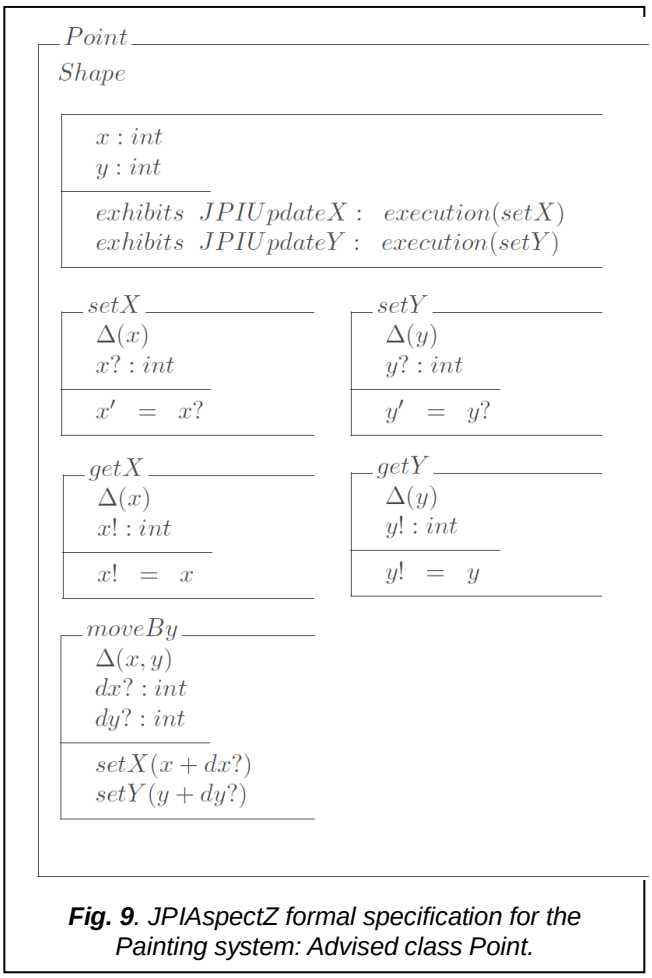


***Fig. 9**. JPIAspectZ formal specification for the Painting system: Advised class Point.*

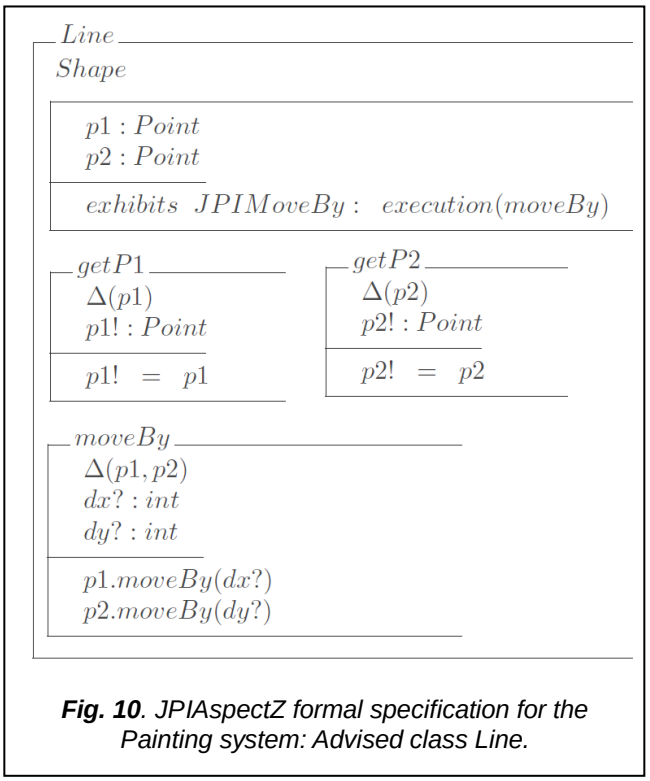


***Fig. 10**. JPIAspectZ formal specification for the Painting system: Advised class Line.*

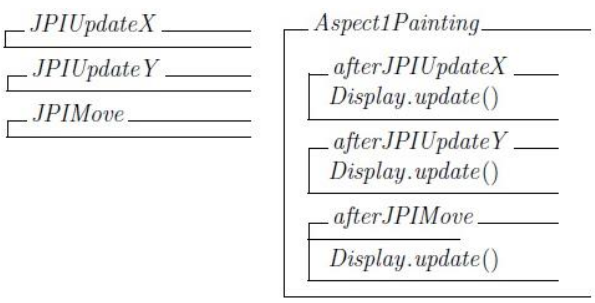


***Fig. 11**. JPIAspectZ formal specification for the Painting system: Join point interfaces and Aspects.*

```
jpi void JPIUpdateX();
jpi void JPIUpdateY();
jpi void JPIMove();
interface Shape{
        void MoveBy(int dx, int dy);
}
class Point implements Shape{
        exhibits void JPIUpdateX():
                execution(* setX(..);
        exhibits void JPIUpdateY():
                execution(* setY(..));
        int x, y;

        void setX(int x){
                this.x = x;
        }
        void setY(int y){
                this.y = y;
        }
        void moveBy(int x, int y){
                this.x += x;
                this.y += y;
        }
}
class Line implements Shape{
        exhibits void JPIMove():
                execution(* moveBy(..));
        Point p1, p2;

        Point getP1(){
                return P1;
        }
        Point getP2(){
                return P1;
        }
        void moveBy(int x, int y){
                p1.moveBy(x, y);
                p2.moveBy(x, y);
        }
}
```

***Fig. 12.** JPI Java code for the Painiting system.*

REFERENCES

[1] G. Kiczales, “Aspect-oriented programming,” ACM Comput. Surv., vol. 28, no. 4es, Dec. 1996. [Online]. Available: http://doi.acm.org/10.1145/242224.242420

[2] I. Jacobson and P.-W. Ng, Aspect-Oriented Software Development with Use Cases (Addison-Wesley Object Technology Series). AddisonWesley Professional, 2004.

[3] F. Wedyan, S. Ghosh, and L. R. Vijayasarathy, “An approach and tool for measurement of state variable based data-flow test coverage for aspect-oriented programs,” Inf. Softw. Technol., vol. 59, no. C, pp. 233–254, Mar. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2014.11.008

[4] M. Wimmer, A. Schauerhuber, G. Kappel, W. Retschitzegger, W. Schwinger, and E. Kapsammer, “A survey on umlbased aspect-oriented design modeling,” ACM Comput. Surv., vol. 43, no. 4, pp. 28:1–28:33, Oct. 2011. [Online]. Available: http://doi.acm.org/10.1145/1978802.1978807

[5] Y. Huiqun, L. Dongmei, Y. Li, and H. Xudong, “Formal aspect-oriented modeling and analysis by Aspect-Z,” pp. 169–174, 2005.

[6] C. Vidal, R. Saens, C. Del R´ıo, and R. Villarroel, “Aspect-oriented modeling: Applying aspect-oriented UML use cases and extending aspect-z,” Computing and Informatics, vol. 32, no. 3, pp. 573–593, 2013.

[7] C. Vidal-Silva, R. Saens, C. Del Río, and R. Villarroel, “OOAspectZ y diagramas de clase orientados a los aspectos para la modelacion´ orientada a aspectos (MSOA),” Ingeniería e Investigación, vol. 33, no. 3, pp. 66–71, 2013.

[8] C. Vidal, R. Villarroel, R. Schmal, R. Saens, T. Tigero, and C. Del R´ıo, “Aspect-Oriented Formal Modeling: (AspectZ + Object-Z) = OOAspectZ,” Computing and Informatics, vol. 34, no. 5, pp. 996–1016, 2015. [Online]. Available: http://www.cai.sk/ojs/index.php/cai/article/view/1380

[9] F. Mostefaoui and J. Vachon, “Verification of aspect-uml models using alloy,” in Proceedings of the 10th International Workshop on Aspect-oriented Modeling, ser. AOM ’07. New York, NY, USA: ACM, 2007, pp. 41–48. [Online]. Available: http://doi.acm.org/10.1145/1229375.1229382

[10] E. Bodden, E. Tanter, and M. Inostroza, “Join point interfaces for safe and flexible decoupling of aspects,” ACM Trans. Softw. Eng. Methodol., vol. 23, no. 1, pp. 7:1–7:41, 2014.

[11] E. Bodden, “Closure joinpoints: Block joinpoints without surprises,” in Proceedings of the Tenth International Conference on Aspectoriented Software Development, ser. AOSD ’11. New York, NY, USA: ACM, 2011, pp. 117–128. [Online]. Available: http://doi.acm.org/10.1145/1960275.1960291

[12] M. Inostroza, E. Tanter, and E. Bodden, “Join point interfaces for modular reasoning in aspect-oriented programs,” in Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ser. ESEC/FSE ’11. New York, NY, USA: ACM, 2011, pp. 508–511. [Online]. Available: http://doi.acm.org/10.1145/2025113.2025205

[13] J. Woodcock and J. Davies, Using Z: Specification, Refinement, and Proof. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996.
[14] G. Smith, The Object-Z Specification Language. Norwell, MA, USA: Kluwer Academic Publishers, 2000.
[15] C. Vidal-Silva and R. Villarroel, “JPI UML: UML class and sequence diagrams proposal for aspect-oriented JPI applications,” in 33rd International Conference of the Chilean Computer Science Society, SCCC 2014, Talca, Maule, Chile, November 8-14, 2014, 2014, pp. 120–123.
[16] S. Apel, D. Batory, C. Kstner, and G. Saake, Feature-Oriented Software Product Lines: Concepts and Implementation, 1st ed. Springer Publishing Company, Incorporated, 2016.