JPIAspectZ
A formal specification language for aspect-oriented JPI applications

Cristian Vidal Silva
Departamento de Computación e Informática
Facultad de Ingeniería, Universidad de Playa Ancha
Av. Lepoldo Carvallo 270, Playa Ancha, Valparaíso, Chile
cristian.vidal@upla.cl

Cristian Pereira Vásquez
Simpartners Software
1601 Sherman Ave, 3rd Floor Evanston, IL 60201, USA
cpereira@simpartners.com

Rodolfo Villarroel
Escuela de Ingeniería Informática
Facultad de Ingeniería, Pontificia Universidad Católica de Valparaíso
Av. Brasil 2241, Valparaíso, Chile
rodolfo.villarroel@ucv.cl

Abstract—This article proposes and describes JPIAspectZ, a formal language used to write Aspect-Oriented (AO) software specification to support the use of Join Point Interfaces (JPI) between advised classes and aspects. JPIAspectZ looks for a concept and model consistency for a JPI software development process. The main characteristic of JPI is to define explicts associations between aspects and advised classes by means of join point interfaces, so that classes are no longer oblivious of their possible interaction with aspects and as well aspects, for their action effectiveness, do not depend on signatures of advisable class methods. Only JPIAspectZ supports this JPI principle. An application example of JPIAspectZ is presented to analyze the pros and cons of our proposal.

Keywords—JPI; join point interfaces; aspects; JPIAspectZ

I. INTRODUCTION

Aspect-Oriented Programming (AOP) proposed by [11], permits modularizing crosscutting concerns in Object-Oriented Software Development (OOSD) stages. Since AOSD was born at the Object-Oriented (OO) programming stage, to reach a transparency of concepts and design for a complete AOSD process seems rather complex. Looking for this transparency, there have been different proposals of different language adaptations to support AOSD such as Aspect-Oriented (AO) UML use case diagrams [1] and AO UML class diagram [2]. Specifically, [3] presents a survey of AO UML languages. Nevertheless, only a few articles about aspect-oriented formal languages for requirement specifications have been proposed so far, i.e., [4] [5] presented AspectZ, and [6] presents an AO alloy version.

In addition, Bodden et al. [7] indicated that a double-dependency between base modules and aspects exists in traditional AOSD. To solve this issue, Bodden et al. [7] [8] [9] proposed the use of join point interface, JPI elements, between classes and aspects. Thus, with the purpose of obtaining JPI solutions and getting a transparency of concepts in stages of the AOSD-JPI process, this article proposes JPIAspectZ, an extension of OOAspectZ [10], for the requirement specification of JPI software applications.

The next section gives the main properties of JPIAspectZ; following that section, Sample Application of AspectZ presents a simple application of our formal language proposal; and the final section concludes and presents future work ideas.

II. JPIASPECTZ SPECIFICATION LANGUAGE

This section gives details about our formal language proposal, JPIAspectZ. To understand its main properties, a short review of main concepts and ideas of JPI is necessary.

A. Join Point Interfaces

Aspect-Oriented Programming (AOP) proposed by [11], permits modularizing crosscutting concerns as aspects in OOP, so that aspects advise classes, i.e., like events, aspects introduce behavior into classes. Nevertheless, as [7] indicated, 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 completely oblivious of possible changes on their behavior. Second, aspects can be ineffective or spurious for signature changes on advised methods of target classes. As [7] [8] mentioned, the last issue is known as the fragile pointcut problem. Likewise, [7] also indicated that in classic AOP independent development of base code and aspect modules is compromised since for independent development in classic AOP both developers of base code and aspects must present global knowledge about the modules of a program, i.e., aspects and classes.

To isolate crosscutting concerns and get modular AOP programs without the mentioned implicit dependencies,
[7] proposed the JPI programming methodology. JPI introduces the idea of join point interface on classic AOP [11]. Like classic AOP [11], for JPI applications, aspects represent crosscutting functionalities, but without PCs rules. Aspects only indicate join point interfaces in JPI. In addition, in JPI, non-oblivious advised classes exhibit explicit join point interfaces, i.e., classes know about potential changes on their methods. Figures 1 and 2 [7] illustrate dependencies between aspects and classes in classic AOP and JPI applications, respectively.

B. JPIAspectZ

Z and Object-Z are formal languages for requirement specification [12] [13]. Likewise, AspectZ and OOAspectZ represent Z extensions for specifying requirements of AOP applications and their integration with Z and ObjectZ, respectively. Considering JPI ideas, we propose JPIAspectZ, a Z extension to model JPI applications and their integration with Object-Z. The next lines present the main elements of JPIAspectZ.

- Base Modules.

For JPIAspectZ, base modules are specified as Object-Z class modules [13] which include an exhibits rule concerning advisable operations of advised class instances. Figure 3 shows the structure of a JPIAspectZ class 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 indicating the join point interface exhibited by the class, and second, a set of conditions for the join point event. So far, we consider basic AOP and JPI conditions for dynamic and static crosscuts: call operation; execution operation; logic conectors &&, ||, !; args(arguments list); this(object); and target(object). The next section presents a sample application of JPIAspectZ.

- Join point interfaces.

In JPIAspectZ, for a system specification, join point interfaces are represented as operation schemas, named JPI schemas. The name of JPI schemas starts with <<JPI>>. Furthermore, JPI schemas only present a declaration section to indicate their list of parameters.

- Aspects.

JPIAspectZ aspects are represented like Object-Z class diagram name aspect-schemas. Aspect-schemas include state schemas to define attributes and invariants, and also operation schemas. As a distinction regarding class schemas, Aspect-schemas permit the inclusion of before, after, and around operations, to specify the order of operation actions defined in JPI schemas. Semantically, aspect-schemas advise operation schemas, i.e., in general, aspect schemas insert declaration, and, specifically, behavior at the beginning, end, beginning and end of advised schemas, respectively.

From advised operation or method schemas and associated aspect-schemas, JPIAspectZ permits obtaining woven schemas.

III. SAMPLE APPLICATION OF JPIAspectZ

To apply JPIAspectZ, we use the same example as [5], Health-Watcher system, for which there are two main associated classes, Unit and Specialty, as Figure 4 shows. Note that a Unit instance is associated to many Specialty instances, and to individualize Unit and Specialty instances, specialtyCode and unitCode attributes are part of the classes Unit and Specialty respectively. For more details about Health-Watcher, review [5].

Figure 5 presents JPIAspectZ specification of class Health-Watcher. Note that Health-Watcher includes sets of
**Unit and Specialty objects.** Class Health-Watcher includes methods getUnits and getSpecialties to obtain, for a given a codeSpecialty and codeUnit, obtain sets of associated Unit and Specialty instances, respectively. Since verifying the given information for these methods correctness represents a crosscutting concern, verification is not part of the class Health-Watcher methods nature. So, for getUnits and getSpecialties operations, class Health-Watcher exhibits join point interfaces JPIVerificationSpecialty and JPIVerificationUnit.

**IV. CONCLUSIONS AND FUTURE WORK**

JPIAspectZ permits obtaining non-dependent class and aspect formal requirement models according to the JPI main principle.

In addition, this extended abstract demonstrates that a transparency of models and concepts for a JPI software development process is attainable, i.e., a consistency between requirements and code modules for JPI solutions is viable.

For future work, we want to continue proposing extensions on JPIAspectZ to model closure join points as well as advanced dynamic crosscuts [14]. In addition, for an automatic validation of JPIAspectZ specifications we propose developing a JPIAspectZ specification validation tool.

**REFERENCES**


**Fig. 4.** UML class diagram of part of Health-Watcher system.

**Fig. 5.** JPIAspectZ diagram of Health-Watcher system.


