Computing and Informatics, Vol. 34, 2015, 996–1016

ASPECT-ORIENTED FORMAL MODELING: (ASPECTZ + OBJECT-Z) = OOASPECTZ

Cristian Vidal Silva

Departamento de Computación e Informática
Facultad de Ingeniería
Universidad de Playa Ancha
Avenida Leopoldo Carvallo 270, Valparaíso, Chile
e-mail: cristian.vidal@upla.cl

Rodolfo Villarroel

Escuela de Ingeniería Informática
Facultad de Ingeniería
Pontificia Universidad Católica de Valparaíso
Avenida Brasil 2241, Valparaíso, Chile
e-mail: rodolfo.villarroel@ucv.cl

Rodolfo Schmal Simón

Ingeniería Informática Empresarial
Facultad de Economía y Negocios
Universidad de Talca
Avenida Lircay S/N, Talca, Chile
e-mail: rschmal@utalca.cl

Rodrigo Saens, Tamara Tigero, Carolina Del Rio

Ingeniería Comercial
Facultad de Economía y Negocios
Universidad de Talca
Avenida Lircay S/N, Talca, Chile
e-mail: {rsaens, ttigero, cdelrio}@utalca.cl

Abstract. The aspect-oriented software development (AOSD) paradigm permits modularizing crosscutting concerns of base modules, a non-usual task in other software development paradigms. Since AOSD was born in the programming stage as an extension of an object-oriented (OO) programming language, and AOSD considers, in addition to base modules, new modules named aspects, then a complete AOSD process requires that each stage considers the base and aspect modules. Therefore, looking for an AOSD process, mainly to apply AOSD in other phases of the OO software development process, so far, different OO modeling tools and language extensions to support AOSD have been proposed. As an example, AspectZ is an extension of the formal language Z to support AOSD. To reach a transparency of concepts and design in AOSD, the main contribution of this article is to propose OOAspectZ, a formal language for the requirements specification stage of aspect-oriented (AO) software applications, that, firstly, extends AspectZ and, secondly, integrates Object-Z and AspectZ formal specifications. Thus, OOAspectZ supports relevant AO elements such as join points, and *This* and *Target* objects for join point events. As an application example, this article applies OOAspectZ to a system named GradUTalca for a Chilean university. For GradUTalca, this article presents AO UML use cases and UML class diagrams, formal Object-Z and OOAspectZ specifications, and a final woven specification to show an integration of Object-Z and OOAspectZ specifications.

Keywords: AspectZ, Object-Z, Z, aspects, crosscutting concerns

1 INTRODUCTION

Even though object-oriented software development (OOSD) is a dominant paradigm nowadays, crosscutting concerns, scattered and tangled elements (crosscutting concerns) are usually part of object-oriented solutions. To isolate crosscutting concerns at the programming phase of the OOSD process, for oblivious classes, i.e., classes which do not know about aspects and possible changes in their behavior and structure produced by the aspects [1], we have here proposed the aspect-oriented programming (AOP) approach.

Apel et al. [2] distinguished between static and dynamic crosscutting concerns classified as homogeneous and heterogeneous. Globally, AOP is able to modularize any kind of crosscutting concerns [2], though AOP is more elegant to modularize static and dynamic homogeneous crosscutting concerns to avoid code replication, since collaboration of classes are typically of a heterogeneous structure. For more details, review works of [3, 4, 5].

For getting modular OOSD, there exist design pattern proposals such as the *Template Method* pattern [6] in which subclasses inherit abstract and final methods from an abstract super class, so the subclasses implement abstract methods and execute inherited final methods as well. However, subclasses are not oblivious of inherited methods, subclasses implement part of inherited methods, and explicitly

demand for execution of inherited methods, i.e., a separation of concerns is not complete using the *Template Method* pattern.

Original AOP identifies and encapsulates crosscutting concerns on base modules, i.e., functionalities which crosscut the classes structure and behavior, hence oblivious base modules deal only with their base concern and crosscutting concerns are separate and independent modules, the aspects, to advise base modules [7]. In original AOP, aspects advise base modules without an explicit invocation and announcement [1, 8]. Classic examples of crosscutting concerns in software applications are security, caching or logging functionalities [1, 9, 10, 11].

According to [9, 10, 12], AOSD is an extension of OOSD because AOSD provides a new type of module for crosscutting concerns, aspects, to encapsulate, isolate, and define associations between aspects and base modules. So, compared to OOSD models, AOSD models seem easier to understand, to use and to maintain, and classes are able to work only on one responsibility [13]. Nevertheless, such as Bodden et al. indicated [14], for classic AOP, aspects define pointcut rules, to be effective to advise classes, rules which depend on advisable modules signatures, i.e., aspects depend on classes and their methods signatures, a current issue for evolving software. Similarly, oblivious classes can perceive non-expected changes on its structure and behavior, therefore advisable classes depend on aspects. Clearly, these issues represent a double-dependency in classic AOP modules. In addition, as Sullivan et al. [7] also said, although the ability of AOSD to modularize crosscutting concerns seems to improve quality of software applications, aspects also may be used in a harmful way to annul desired properties and even destroy conceptual integrity of systems.

Considering the potential advantages of AOSD to produce a modular software, and since classic AOSD represents a promising solution for the modular software production, there already exist extensions for software design languages to support aspect-oriented modeling such as [9, 10, 15]. Nevertheless, only a few proposals of formal languages for aspect-oriented software requirements specification exist. Thus, looking for a complete separation of concerns and transparency of models and concepts in stages of the AOSD process, the main goal of this article is to propose the OOAspectZ formal language, an extension of AspectZ formal language [16, 17] for its integration with Object-Z [18, 19, 20], and apply OOAspectZ to a case study to show its potential benefits. Clearly, OOAspectZ represents as extension of the Z formal language [21]. We use the graduation process for students of a college to apply OOAspectZ.

The rest of the paper is organized as follows: Section 2 presents an example of an information system used to assist students who need to certify their graduation status. The system is named GradUTalca. Section 3 introduces and explains elements of Object-Z used to model the GradUTalca system. Section 4 describes main elements of the original proposal of Aspect-Z and gives syntax and semantic details of OOAspectZ as an extension of Aspect-Z. Section 5 presents the application of OOAspectZ on the GradUTalca system. Finally, Section 6 concludes the article and suggests future work.

2 EXAMPLE (CASE STUDY)

GradUTalca is an information system used by the University of Talca, Chile, to assist students in their graduation process. Figure 1 shows associated university offices for an usual graduation process; so, a student who asks for a degree certification, 1st, has to go to the academic registration office and asks for degree certificates. Afterward, the student has to go, for each required degree certificate, to the associated school office to certify a required academic credits completion as well as a no bills existence in other defined offices at the university. For the last mentioned process, the school office provides a document, an academic record, to the student to be signed by each required office director. Therefore, to obtain a degree certificate, a student should proceed in each defined university office to certify he or she does not bill and get signature of each office director to present those signatures to the school office to proceed his or her graduation process. Note, that school office is the only office linked to the academic registration office to certify a student is able to graduate, so the school office reviews the correctness of each office director signature and verifies whether the student has completed his required credits to graduate. If so, the head of the school signs the academic record, and the student waits for about 15 days to receive his or her degree certificate from the academic registration office.

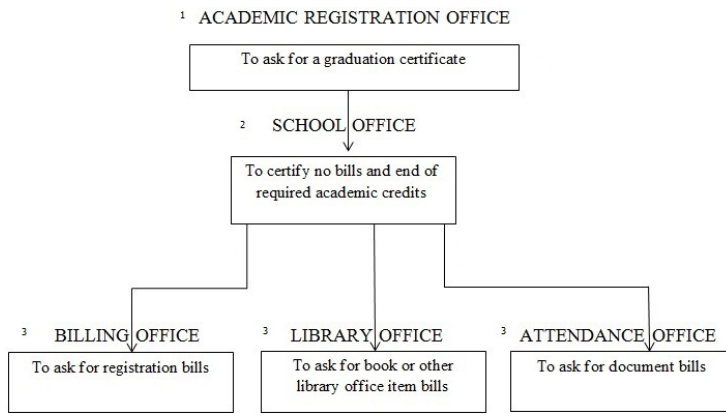

Figure 1. Steps for getting a degree or title certificate at the University of Talca, Chile

Since, the main function of each office of GradUTalca is to, as a precondition, verify the student status before giving a signature, in order to avoid crosscutting concerns on these units, verification processes are modularized as aspects. Only school office has also to check the correctness of signatures of other offices as well as whether the student, who is asking for degree certificates, has already completed his or her studies, i.e., school office has to apply a double verification (a nested-aspect).

Figure 2 shows an AO UML use-case diagram of GradUTalca. This figure follows the AO principles described and applied by [9, 17].

As a structural model of GradUTalca system, class diagram of Figure 3 shows a main class GradUTalca composed by individual instances of classes AcademicRegistration, BillingOffice, LibraryOffice and AttendanceOffice. In addition, class GradUTalca is composed by a set of class SchoolOffice instances since the University of Talca presents different schools. These mentioned classes are also part of the GradUTalca system. Classes GradUtalca, AcademicRegistration, BillingOffice, LibraryOffice, and AttendanceOffice share the same set of students, and each of these classes adds, for each student, a proper class information along with the student status to determine if a student is able to graduate or not. Conceptually, each office reviews information of a student to determine his or her status. So, each class presents methods to assign a true value to the status of a given student *st*, *Sign(st: Student)* and to know the current status of a student *st*, *getStatus(st: Student)*.

Figure 3 uses the same notation to represent AO UML class diagrams proposed and applied by [15]. Figure 3 presents, for each class, before executing the *Sign(st: Student)* method, a verification aspect to verify the student status, i.e., some offices at the University of Talca review if there are not issues regarding the student *st* information to proceed, so if some office of the GradUTalca system does not proceed, thus, method *Sign(st: Student)* will not proceed as well.

Surely, an OO GradUTalca system version would present verification operation as part of the system classes (crosscutting concern), hence a single responsibility principle, a basic OO design principle [13], is not respected by these classes. Therefore, by means of the aspects, a more modular GradUTalca system is reachable. Particularly, for method *Sign(st: Student)* of class SchoolOffice, it is necessary to know the student status at each other system office before giving permission for a degree certification; therefore, for the method *Sign(st: Student)* of class SchoolOffice, an additional aspect to verify the student status at other defined offices of the GradUTalca system is defined. For this scenario, following AO principles to identify objects associated to aspects, since a GradUtalca class object *g1* instantiates objects of SchoolOffice and also *g1* invokes the execution of *Sign(st: Student)* of these SchoolOffice instances, for the call of a method *Sign(st: Student)* of class SchoolOffice, *This* and *Target* objects are recognized and different; *This* object is the instance of GradUTalca *g1* whereas *Target* object is an instance of SchoolOffice, respectively. Thus, by means of the object *This*, at the call of method *Sign(st: Student)* at the current instance of SchoolOffice, attributes of GradUTalca instance are accessible, so it is possible to know the student *st* status at other offices of the GradUTalca system. Section 4 describes of *This* and *Target* objects for our OOAspectZ formal language proposal, and Section 5 presents the use of OOAspectZ and *This* and *Target* objects.

As an AO UML class diagram, Figure 3 presents pointcut and aspect elements for classes SchoolOffice (SO) and LibraryOffice (LO) only. A complete class diagram version of GradUTalca system includes similar pointcut and aspect elements for classes BillingOffice (BO) and AttendanceOffice(AO) as well.

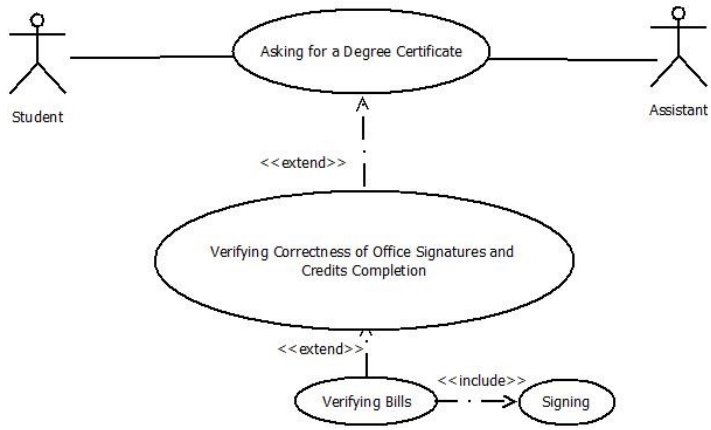

Figure 2. AO UML use-case diagram of GradUTalca System

3 OBJECT-Z

Object-Z is an extension of the Z formal language [21] to support characteristics of the OO software development (OOSD) [18, 19, 20]. Object-Z preserves and extends properties and elements of the Z formal language to support OOSD. For example, since a schema is a basic element of Z formal language to formally specify system properties and operations, Object-Z presents class schema, i.e., an adaptation of Z schema, to formally specify classes and encapsulate attributes and method schemas. In this article, methods schemas are known as operation schemas.

Using an Object-Z formal model for the classes of GradUTalca system, and looking for a separation of concerns as well as a transparency of concepts and design between AO models and code of GradUTalca, Figure 4 specifies general data for the system specification and class SchoolOffice; Figure 5 specifies class LibraryOffice; and Figure 6 specifies classes GradUTalca and AcademicRegistration. Classes BillingOffice and AttendanceOffice can be defined like classes SchoolOffice and LibraryOffice. Note that, for the GradUTalca system, there exists a consistency between class digram and Object-Z model regarding classes structure and behavior along with associations among classes.

Note, that looking for a separation of concern, schemas of classes SchoolOffice and LibraryOffice do not verify student *st? Info* to proceed in the operation schema *Sign*. In addition, class schema of SchoolOffice does not verify, before signing, that *Status* of *st?* is valid in other offices of the GradUTalca system, since these verification processes are crosscutting concerns.

Classes SchoolOffice and LibraryOffice, as well as classes BillingOffice and AttendanceOffice, present an attribute students for a composed set of Student object, *Info* data, and *Status* data. As Figure 5 shows, each office presents an invariant for the uniqueness of a student object in the set of students, i.e., there can not be

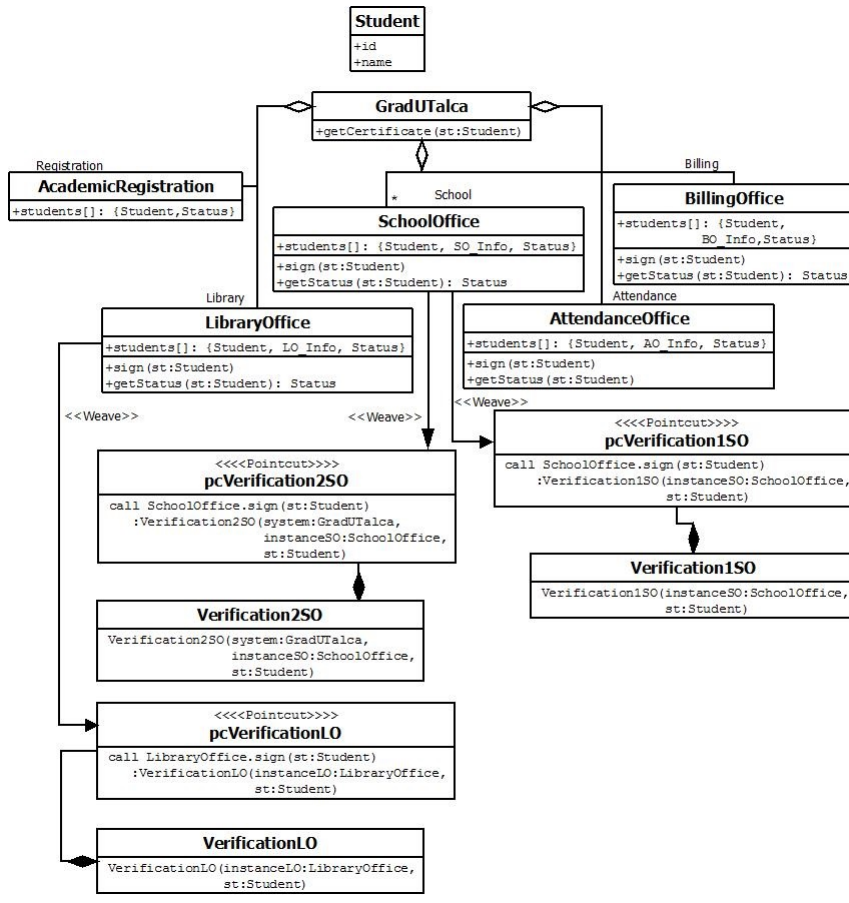

Figure 3. SO: SchoolOffice, BO: BillingOffice, LO: LibraryOffice, AO: AttendanceOffice AO UML class diagram of GradUTalca system

two tuples associated to the same student, even though a student can be in two different schools. In addition, as Figure 6 presents, as an invariant of class GradUTalca, instances of components of class GradUTalca present the same set of Student objects. For doing so, to refer to a particular component of a tuple attribute, we use a projection notation [21], so to access the 1st component of the set students of an object *registration* of class GradUTalca, i.e., the set of students only, we use the *registration.students.1* notation.

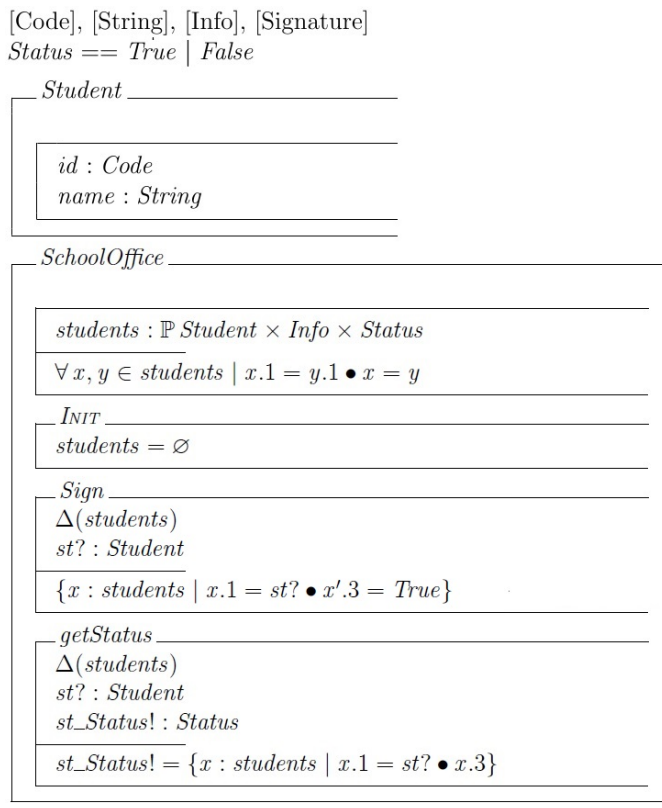

Figure 4. Object-Z model of class SchoolOffice of GradUTalca system

4 ASPECT-Z

AspectZ is an extension of the Z language proposed by [16]. Even though Yu et al. [16] presented application examples of AspectZ, important elements of AOSD [1] are not supported by AspectZ. Specifically, according to [1, 9, 12, 10, 11], in AOSD, for the occurrence of a join point event, aspects advice before, after or around that event. Therefore, an AO language has to support those kind of advices. Nevertheless, original AspectZ permits definitions of aspect-schema only for *insert* and *replace* advice on operation schemas, i.e., to reinforce some restraint on the operation schema and modify functions defined in the base section of the advised operation schema, respectively.

In addition, original AspectZ identifies a join point in an operation schema by the existence in it of a defined expression or predicate. Hence, when a join point is identified, associated aspect-schema adds (insert) or changes (replace) the identified expression or predicate in the advised operation schema. Therefore, since

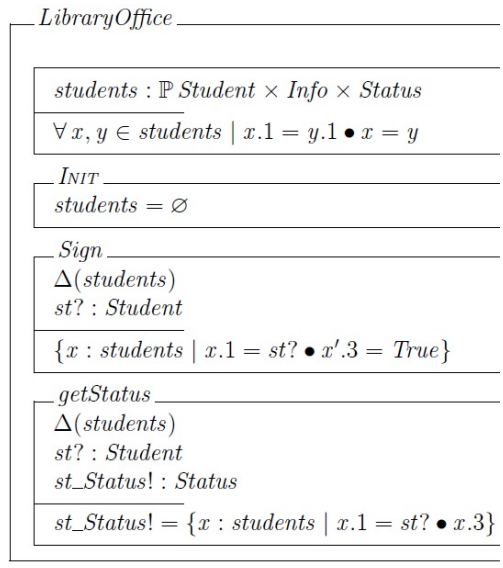

Figure 5. Object-Z model of class LibraryOffice of GradUTalca system

predicate section of Z and Object-Z schemas do not evaluate predicates and rules in an established order; this kind of reaction disagrees with AO kind of advices.

Figure 7 shows the structure of an AspectZ aspect-schema. Clearly, an aspect-schema is a normal Z schema that includes a declaration and predicate section, at the top and bottom, respectively. This figure presents that the declaration section of an aspect-schema uses the symbol Ω to indicate advisable schemas. Like the symbol δ for Z and Object-Z operation schemas to indicate properties of a system, or attributes of a class potentially affected by the operation, Ω in an aspect-schema indicates a list of schemas advisable by the aspect-schema for join point events.

Taking in account that aspects have to advise join point events on class instances, i.e., basically for the methods execution and call of class instances, and also considering mentioned issues of original AspectZ proposal [16], then aspect-schemas of AspectZ have to be redefined in a more appropriate way for a transparency of concepts and design in an AOSD process. As a solution, we present OOAspectZ, an AspectZ extensions to permit an easy integration of AspectZ and Object-Z formal specifications.

4.1 Extensions of Aspect-Z

Even though Yu et al. [16] presented AspectZ application examples, they only allow to advise local operations of Z operation schemas. Thus, an integration of traditional AspectZ and Object-Z seems not possible.

Since an integration of aspect-schema and operation schema is fundamental to

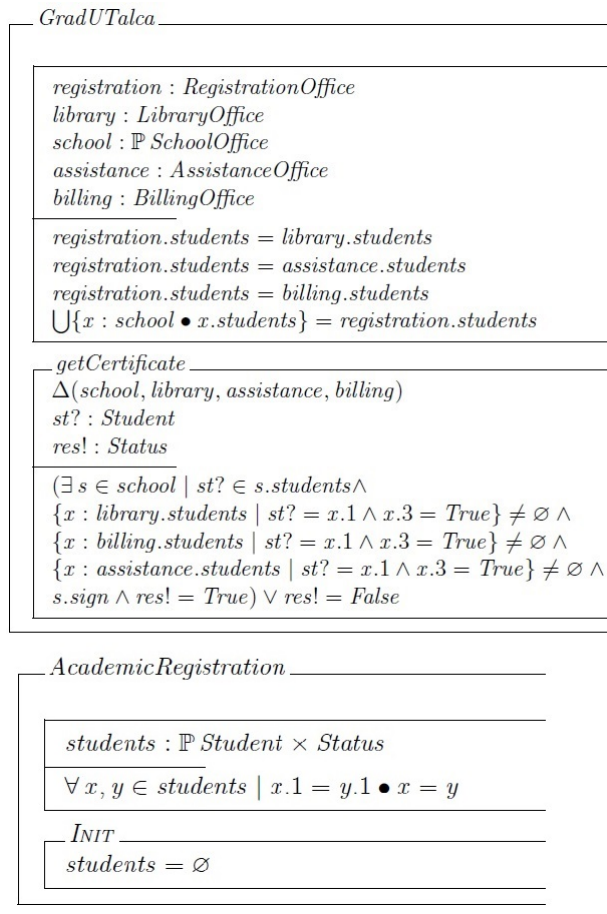

Figure 6. Object-Z model of classes GradUTalca and academic registration of GradUTalca system

support AOSD in Z and Object-Z formal languages, and original Z [21] and Object-Z formal languages [18, 19, 20] permit using the sequential composition operator $\fatsemi$ to define sequential operations, then the integration and interleaving of aspect-schemas and operation schemas seem possible.

Figure 8 shows the use of operator $\fatsemi$ for operations *OpOne* and *OpTwo* where postconditions of *OpOne* are preconditions of *OpTwo*. Therefore, an AspectZ woven schema represents the sequential composition of aspect-schemas and operation schemas in a defined order, i.e., sequential composition is directly applicable to compose aspect-schemas and operation schemas to integrate AspectZ and Object-Z. For example, if a *before* aspect-schema advises an operation schema, then the advised operation schema will wait for the postconditions of the aspect-schema. Similarly, if

$$
\begin{aligned}
\mathit{Declaration} &::= \mathit{BasicDecl};\ \ldots;\ \mathit{BasicDecl} \\
\mathit{BasicDecl} &::= \mathit{Ident}, \ldots, \mathit{Ident} : \mathit{Expr} \\
&\quad \mid \mathit{SchemaRef} \\
&\quad \mid \Omega \mathit{SchemaRef} \\
&\quad \mid \mathit{PointcutDecl} \\
\mathit{SchemaRef} &::= \mathit{SchemaName}\ \mathit{Decoration}[\mathit{Renaming}] \\
\mathit{PointcutDecl} &::= \mathit{Pointcut}\ \mathit{Ident} : \mathbb{P}(\mathit{Ident} : \mathit{Expr}) \\
\mathit{Spec} &::= \mathit{Predicate} \mid \mathit{Advice} \\
\mathit{Advice} &::= [\mathit{insert} \mid \mathit{replace}]\mathit{PointcutName} : \mathit{Predicate}
\end{aligned}
$$

Figure 7. Aspect-schema of original proposal of AspectZ [16]

$$
\begin{aligned}
\mathit{OpOne} \fatsemi \mathit{OpTwo} \ = \ & \exists\, \mathit{State}'' \bullet \\
& \exists\, \mathit{State}' \bullet [\mathit{OpOne};\ \mathit{State}'' \mid \theta \mathit{State}' = \theta \mathit{State}''] \\
& \land \\
& \exists\, \mathit{State} \bullet [\mathit{OpTwo};\ \mathit{State}'' \mid \theta \mathit{State} = \theta \mathit{State}'']
\end{aligned}
$$

Figure 8. Sequential composition operator of Z and Object-Z [19, 21]

the aspect-schema advises *after* the operation schema, then postcondition of the advised operation schema represents a preconditions of the aspect-schema. In the last scenario, the aspect-schema postcondition represents the final state of the woven operation. Nonetheless, sequential composition is not directly applicable to integrate AspectZ *around* aspect-schemas and Object-Z operation schemas since an *around* aspect-schema predicate, per default, adds behavior on the advised operation *before* and *after*, and the operation schema only proceeds if there is a $\mathit{Proceed}_{\Omega}$ instruction in the mentioned aspect-schema.

Furthermore, since each aspect-schema can define sets of input, output, and local elements valid only in their scope, operator $\fatsemi$ seems not applicable. So, without any doubt, it is necessary to define new rules to integrate AspectZ and Object-Z for our OOAspectZ formal language proposal.

Since Yu et al. [16] introduced the use of Ω to identify advisable operation schemas, we use Ω to identify elements in our OOAspectZ proposal described below:

1. An aspect-schema presents two sections, one section for pointcut definitions and one section for pointcut advices.

   - Pointcut definition is in the aspect-schema top division. A pointcut definition uses Ω and permits identifying kind of actions, i.e., for now, call and execution of class methods. It is possible to identify method signature and

arguments.
- Pointcut advices section permits, for each advice, to define kind of advices, conditions, and associated actions.

2. Because Ω can refer to more than one schema (operation schema and aspect-schema), an aspect-schema can advise more than one schema; thus, clearly one schema can be advised multiple times by multiple aspect-schemas. Note that, following original OASD principles [1], advisable schemas are oblivious in AspectZ formal specifications.
3. Since identifying the source of aspects is an important characteristic of AOSD and there are two important sources for each aspect [1, 12], *This* and *Target* objects, for OOAspectZ, as an AspectZ extension, we propose $This_\Omega$ and $Target_\Omega$ to represent the object that asks for a method – not necessarily the owner of the method, and the owner of the advised method, respectively. In addition, $This_\Omega$ and $Target_\Omega$ permit differentiating between the *call* and *execution* of methods in AspectZ modules. Specifically, $This_\Omega$ and $Target_\Omega$ present the following properties:

   .*Aspect*: Identifies the associated aspect. Null value if the associated aspect does not exist.

   .$Method_{args}$: Identifies a vector with the set of arguments of the advised method. If there are no elements for a given index, a null value is obtained. A vector starts in the position 0.

   .$Method_{name}$: Identifies the name of the method advised by the aspect.

   .$Method_{signature}$: Represents a complete signature of the method advised by the aspect. A complete method name is the method name along with its set of arguments.

   .$Target_\Omega$: By means of the attribute $Target_\Omega$, the *Target* object is obtained.

   .$This_\Omega$: By means of the attribute .$This_\Omega$, the object in which the join point event occurs is obtained.

Undoubtedly, the mentioned new properties of AspectZ, i.e., OOAspectZ properties, are consistent with the original OOASD ideas [1]. Figure 8 presents syntax and structure of an OOAspectZ aspect-schema. Note, that an OOAspectZ aspect-schema identifies pointcut definitions, and for each pointcut declaration, it is possible to indicate the associated action for the join point events occurrence (execution or call). The last element is relevant to distinguish between $This_\Omega$ and $Target_\Omega$ objects in an aspect-schema.

Because advice elements are defined in the predicate part of aspect-schemas, advice elements establish preconditions and postconditions for advised schemas.

In the next section there is an application of the OOAspectZ to the case study GradUTalca.

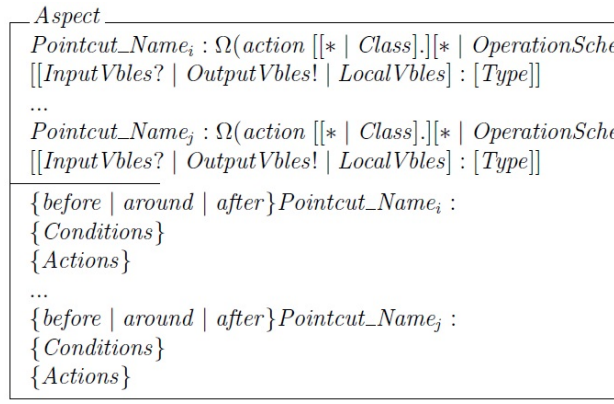

Figure 9. Aspect-schema of OOAspectZ

5 OOASPECTZ MODEL APPLICATION

This section presents OOAspectZ aspect-schemas for the GradUTalca system as well as woven schemas from OOAspectZ aspect-schemas and Object-Z class schemas of the GradUTalca system.

As Section 2 mentioned, the OO solution for the GradUTalca system would include verification operations as part of the system classes even though these verification functions are not part of the system classes nature. Therefore, according to [13], a traditional GradUTalca OO solution would violate the single responsibility basic object-oriented design principle. In addition, according to [1] and [12], encapsulation principle, non-respected by a traditional OO GradUTalca solution as well, represents a basic principle of the OOSD paradigm. These issues demonstrate that verification functions are clear aspects in the GradUTalca system, each of them advises before proceeding with the associated *Sign(st?: Student)* operation of the GradUTalca system classes.

Assuming the existence of the GradUTalca system classes described in Section 3, Figures 10, 11, 12, and 13 show the associated OOAspectZ aspect-schemas for classes SchoolOffice, LibraryOffice, BillingOffice, and AttendanceOffice, respectively. Note, that for the GradUTalca system, there exists a consistency between class diagram and OOAspect-Z model, this time, regarding aspects structure, associations, and behavior. Next lines give more details about these figures:

- First, Figure 10 shows the aspect Verification1 and Verification2 for SchoolOffice class. Verification1 checks/verifies that the student *st?*, as the $Target_{\Omega}$ object that is an instance of SchoolOffice advised by the aspect, belongs to the Student set, *students.1*, of that SchoolOffice instance. Furthermore, Verification1 assures that the Info of the Student *st?* is in agreement with the information of a SchoolOffice instance. On the other hand, the aspect Verification2 veri-

fies whether the student *st?*, as the $Target_{\Omega}$ object, is a valid student for the other GradUTalca system offices, i.e., whether the *st?* Status is valid for all the other offices, components of class GradUTalca, to proceed the graduation process for *st?*. Note, that either the set of *students* or the method getStatus of each GradUTalca system class must be public to access the associated student status value to avoid **Inaccessible join points** [7].

- Second, Figure 11 shows the aspect Verification3 for the class LibraryOffice.
- Third, Figure 12 shows the aspect Verification4 for the class BillingOffice.
- Fourth, Figure 13 shows the aspect Verification5 for the class AttendanceOffice.

*Verification*1

$pcVerification1 \ : \ \Omega(call \ SchoolOffice.Sign)$

around $pcVerification1$:
$\{\exists\, x : Target_{\Omega}.students \mid x.1 \in Method_{args} \wedge x.2 = \ll Value_{SO} \gg \bullet$
$x.3 = true\}$
$Proceed_{\Omega}$

*Verification*2

$pcVerification2 \ : \ \Omega(call \ SchoolOffice.Sign)$

around $pcVerification2$:
$\{\exists\, x : This_{\Omega}.registration.students \mid x.1 \in Method_{args} \wedge x.3 = true\}$
$\{\exists\, x : This_{\Omega}.library.students \mid x.1 \in Method_{args} \wedge x.3 = true\}$
$\{\exists\, x : This_{\Omega}.assistance.students \mid x.1 \in Method_{args} \wedge x.3 = true\}$
$\{\exists\, x : This_{\Omega}.billing.students \mid x.1 \in Method_{args} \wedge x.3 = true\}$
$Proceed_{\Omega}$

Figure 10. Aspect-schemas Verification1 and Verification2 for method Sign of class SchoolOffice

Aspects Verification1 and Verification2 define a pointcut for the call of method *Sign* of class SchoolOffice. According to defined AspectZ extensions in the previous section, an aspect-schema completely accesses the parameters of advised operation schema. Thus, in the system GradUTalca, these aspects advise call events of method *Sign(st?: Student)* of class SchoolOffice.

Aspect-schemas in Figure 10 show the use of the predicate $Proceed_{\Omega}$ which is always valid for the composition of aspect-schema and advised schema predicates and rules. When the composition of an aspect-schema that advises *around* a pointcut, i.e., an *around* aspect-schema, includes a $Proceed_{\Omega}$, the advised schema continues its process. However, when the composition of an *around* aspect-schema is not valid, i.e., a $Proceed_{\Omega}$ does not occur, so the advised schema does not continue its process.

The aspect-schema Verification1 proceeds (postcondition), i.e., advised *Sign(st?: Student)* method of the SchoolOffice instance proceeds, when the attribute *st?* of

the operation schema is part of the set of students of the $Target_{\Omega}$ object (object SchoolOffice), and the student *st?* presents info of SchoolOffice (preconditions). Similarly, the aspect-schema Verification2 proceeds giving the control flow to the advised method of class SchoolOffice when the attribute *st?* of the operation schema is part of the set of students of the $This_{\Omega}.registration$ object (registration object of the current instance of class GradUTalca). This verification considers the established invariant rule of class GradUTalca.

Figures 11, 12, and 13 show aspect-schemas similar to aspect-schema Verification1 to advise *Sign(st?: Student)* for classes LibraryOffice, BillingOffice, and AttendanceOffice, respectively. Even though the aspects-schemas seem specific to advised classes, they represent a crosscutting concern and permit classes to respect the single responsibility principle [13].

$$
\begin{array}{|l}
\hline
\textit{Verification3} \\
\textit{pcVerification3} \ : \ \Omega(\textit{call LibraryOffice.Sign}) \\
\hline
\textit{around pcVerification3} : \\
\{\exists\, x : \textit{Target}_{\Omega}.\textit{students} \mid x.1 \in \textit{Method}_{\textit{args}} \wedge x.2 =\ll \textit{Value}_{LO} \gg \bullet \\
\ \ x.3 = \textit{true}\} \\
\textit{Proceed}_{\Omega} \\
\hline
\end{array}
$$

Figure 11. Aspect-schema Verification3 for method Sign of class LibraryOffice

$$
\begin{array}{|l}
\hline
\textit{Verification4} \\
\textit{pcVerification4} \ : \ \Omega(\textit{call BillingOffice.Sign}) \\
\hline
\textit{around pcVerification4} : \\
\{\exists\, x : \textit{Target}_{\Omega}.\textit{students} \mid x.1 \in \textit{Method}_{\textit{args}} \wedge x.2 =\ll \textit{Value}_{BO} \gg \bullet \\
\ \ x.3 = \textit{true}\} \\
\textit{Proceed}_{\Omega} \\
\hline
\end{array}
$$

Figure 12. Aspect-schema Verification4 for method Sign of class BillingOffice

$$
\begin{array}{|l}
\hline
\textit{Verification5} \\
\textit{pcVerification5} \ : \ \Omega(\textit{call AssistanceOffice.Sign}) \\
\hline
\textit{around pcVerification5} : \\
\{\exists\, x : \textit{Target}_{\Omega}.\textit{students} \mid x.1 \in \textit{Method}_{\textit{args}} \wedge x.2 =\ll \textit{Value}_{AO} \gg \bullet \\
\ \ x.3 = \textit{true}\} \\
\textit{Proceed}_{\Omega} \\
\hline
\end{array}
$$

Figure 13. Aspect-schema Verification5 for method Sign of class AttendanceOffice

Finally, since for woven OOAspectZ schemas, translations of OOAspectZ elements $This_\Omega$ and $Target_\Omega$ into object references for Object-Z schemas are necessary, we listed a few situations where to obtain woven OOAspectZ schemas should be considered:

- Aspect-schemas which advise the *execution* of operation schemas, so advised operation schemas can be defined with the inclusion of the aspect behavior. This is because for execution of operation schemas, $Target_\Omega$ and $This_\Omega$ refer to the same object, i.e., the object owner of the executed operation schema.
- Aspect-schemas which advise *call* of operation schemas and access only $Target_\Omega$ objects. Similar to the previous situation, since aspect-schema refers only to $Target_\Omega$ object, therefore advised aspect-schema can be directly redefined since it only accesses object elements.
- Aspect-schemas which advise *call* of operation schemas and access $Target_\Omega$ and $This_\Omega$ objects. This situation refers to the aspect which requires access to $Target_\Omega$ and $This_\Omega$ objects, or $This_\Omega$ object for call of operation schemas such as the aspect *Verification2* in Figure 10. In this case, because the aspect references the $This_\Omega$ object, it needs to access elements of the whole instance. Therefore, looking for a translation transparency for a woven OOAspectZ schema as well as avoiding references to the whole instance in the part instances, this article proposes to define methods with similar names to those of the methods of the part instances in the whole class. Since methods in the whole class have a complete access to the objects previously advised by the aspects, then those new methods of the whole class are implicitly related to the aspect code.

Figures 14 and 15 show final woven Object-Z classes GradUTalca and SchoolOffice, respectively. These figures show the mentioned classes after the weaving process for our proposal of integrating operation schemas and aspect-schemas.

For the remaining classes, AttendanceOffice, LibraryOffice, and BillingOffice advised by the aspects similar to Verification1, OOAspectZ woven class schemas are similar to the class schema SchoolOffice of Figure 15. In addition, to reuse OOAspectZ schemas for GradUTalca system, it is possible to define only one aspect-schema to advise AttendanceOffice, LibraryOffice, BillingOffice, and one of the verification processes of SchoolOffice, because aspect-schema permits the join point integration. Likewise, conceptually only one class for offices of GradUTalca system can be defined given their similarity in behavior and structure.

6 CONCLUSIONS

This article presented OOAspectZ formal language to integrate AspectZ and Object-Z formal specifications. For that purpose, AspectZ extensions were presented which maintain the main principles of AOSD taking the ideas of the original AspectZ language, as well as including elements of AOSD not supported by the original

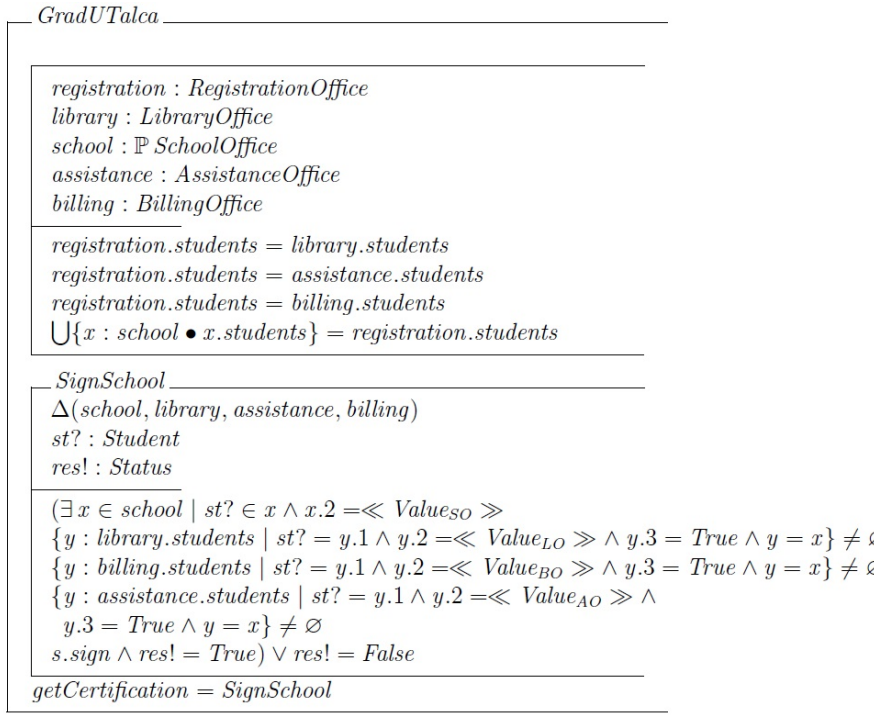

Figure 14. Woven Object-Z class schema GradUtalca

version of AspectZ to facilitate the Object-Z and AspectZ specifications integration, i.e., OOAspectZ formal specifications.

This article shows examples to validate the main OOAspectZ goal of integrating AspectZ and Object-Z formal specification for the construction of woven specifications. Thus, looking for a transparency of the concept and design in the AOSD process, this article presented OOAspectZ, a formal language to extend AspectZ to facilitate its integration with Object-Z. By means of an application example, this article showed that a transparency of concepts and design between AO UML class diagrams and OOAspectZ formal specifications is reachable. A related future work is to analyze a transparency of AO models and a final AO implementation.

The use of OOAspectZ as an AO formal language requires to pay more attention to modeling stages of the software development process. Hence, considering AspectZ as an extension of Z, a known and used formal language to support AOSD along with the integration of AspectZ and Object-Z languages, the use of OOAspectZ definitely promotes the application of AOSD in the design and modeling stages for the AOSD process.

Even though this article showed OOAspectZ to integrate AspectZ and Object-Z, there are elements of AOSD where the integration is not present in this article.

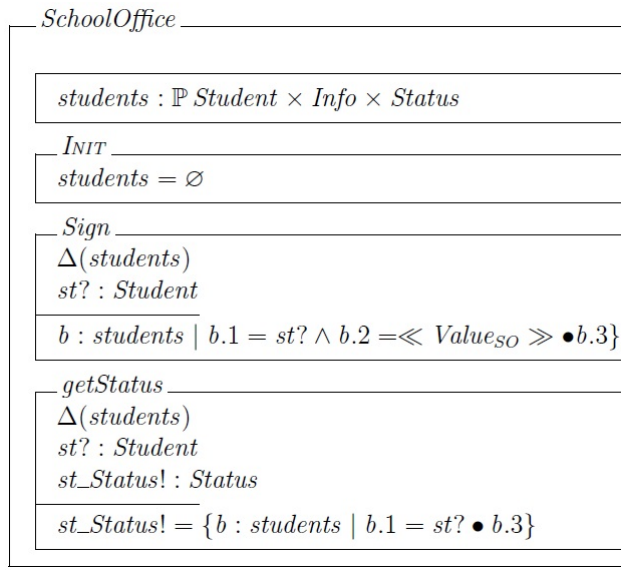

Figure 15. Woven Object-Z class schema SchoolOffice

Therefore, OOAspectZ future work will include definitions of introductions and redefinition of inheritance association among classes, as well as review interference among aspects and classes.

An additional future work is to analyze a potential support of OOAspectZ for the symmetric AOSD approach [11] and to extend OOAspectZ to support join point interface (JPI) models [14]. Furthermore, given the current prominence of using tools for verifying and validating design models, the future work is to develop an integration of OOAspectZ with current Z and Object-Z tools [22] for the type and model checking.

REFERENCES

[1] Kiczales, G.—Lamping, J.—Mendhekar, A.—Maeda, C.—Lopes, C. V.—Loingtier, J.-M.—Irwin, J.: Aspect Oriented Programming. Proceedings of European Conference on Object-Oriented Programming (ECOOP), Finland, June 1997, Springer-Verlag, LNCS, Vol. 1241, 1997.

[2] Apel, S.—Batory, D.—Kästner, C.—Saake, G.: Feature-Oriented Software Product Lines: Concepts and Implementation. Springer-Verlag Berlin, Heidelberg, 2013, pp. 129–174.

[3] Mezini, M.—Ostermann, K.: Variability Management with Feature-Oriented Programming and Aspects. Proceedings of International Symposium Foundations of Software Engineering (FSE), ACM Press, 2004, pp. 127–136.

[4] Ostermann, K.—Giarrusso, P. G.—Kästner, C.—Rendel, T.: Revisiting Information Hiding: Reflections on Classical and Nonclassical Modularity. Proceedings of European Conference Object-Oriented Programming (ECOOP), Springer-Verlag, Lecture Notes in Computer Science, Vol. 6813, 2011, pp. 155–178.

[5] Apel, S.—Leich, T.—Saake, G.: Aspectual Feature Modules. IEEE Transactions on Software Engineering (TSE), Vol. 34, 2008, No. 2, pp. 162–180.

[6] Gamma, E.—Helm, R.—Johnson, R.—Vlissides, J.: Design Patterns, Elements of Reusable Object-Oriented Software. Addison-Wesley, 1st Edition, 1994, pp. 325–330.

[7] Sullivan, K.—Griswold, W. G.—Song, Y.—Cai, Y.—Shonle, M.—Tewari, N.—Rajan, H.: Information Hiding Interfaces for Aspect-Oriented Design. Proceedings of 10th European Software Engineering Conference and 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE-13), 2005, pp. 166–175.

[8] Steimann, F.—Pawlitzki, T.—Apel, S.—Kästner, C.: Types and Modularity for Implicit Invocation with Implicit Announcement. ACM Transactions on Software Engineering and Methology (TOSEM), Vol. 20, 2009.

[9] Jacobson, I.—Ng, P.-W.: Aspect Oriented Software Development with Use Cases. Addison Wesley Professional, New York, USA, 1st Edition, 2004.

[10] Wimmer, M.—Schauerhuber, A.—Kappel, G.—Retschitzegger, W.—Schwinger, W.—Kapsammer, E.: A Survey on UML-Based Aspect-Oriented Design Modeling. ACM Computing Surveys, Vol. 43, 2011, No. 4, art. no. 28.

[11] Bálik, J.—Vranić, V.: Symmetric Aspect-Orientation: Some Practical Consequences. Proceeding of Workshop on Next Generation Modularity Approaches for Requirements and Architecture (NEMARA '12), ACM, 2012, pp. 7–12.

[12] Gradecki, J. D.—Lesiecki, N.: Mastering AspectJ: Aspect-Oriented Programming in Java. Wiley Publishing, Inc., Indianapolis, Indiana, 1st Edition, 2003.

[13] Wampler, D.: Noninvasiveness and Aspect-Oriented Design: Lessons from Object-Oriented Design Principles. Proceedings of 6th International Conference on Aspect-Oriented Software Development (AOSD '07), Vancouver, Canada, March 2007.

[14] Bodden, E.—Tanter, E.—Inostroza, M.: Join Point Interfaces for Safe and Flexible Decoupling of Aspects. ACM Transactions on Software Engineering, Vol. 23, 2014, No. 7, pp. 7–41.

[15] Liu, C.-H.—Chang, C.-W.: A State-Based Testing Approach for Aspect-Oriented Programming. Journal of Information Science and Engineering, Vol. 24, 2008, pp. 11–31.

[16] Yu, H.—Liu, D.—Yang, J.—He, X.: Formal Aspect-Oriented Modeling and Analysis by Aspect-Z. Proceedings of 17th International Conference on Software Engineering and Knowledge Engineering (SEKE '2005), Taipei, Taiwan, Republic of China, July 2005, pp. 124–132.

[17] Vidal Silva, C.—Saens, R.—Del Río, C.—Villarroel, R.: Aspect-Oriented Modeling: Applying Aspect-Oriented UML Use Cases and Extending Aspect-Z. Computing and Informatics, Vol. 32, 2013, No. 3, pp. 573–593.

[18] Smith, G.: An Object-Oriented Approach to Formal Specification. Ph.D. Thesis, Department of Computer Science, University of Queensland, Australia. October 1992.

[19] Smith, G.: The Object-Z Specification Language – Advances in Formal Methods. Vol. 1, Springer US, 2000.

[20] Duke, R.—Rose, D.: Formal Object-Oriented Specification Using Object-Z. Macmillan Press Limited, London, 1st Edition, 2000.

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

[22] Utting, M.—Toyn, I.—Sun, J.—Martin, A.—Dong, J. S.—Daley, N.—Currie, D.: ZML: XML Support for Standard Z. Proceedings of 3rd International Conference on Formal Specification (ZB '03), Springer-Verlag, 2003, pp. 437–456.

Cristian Vidal Silva is currently a Ph.D. Candidate in software engineering and technology at the University of Seville, Spain and his current research focuses on the automated analysis of feature models. He is also a Computer Engineer from Catholic University of Maule, Chile, and he received his M.Sc. degrees in computer science from University of Concepción, Chile, and from Michigan State University, MI, USA. Currently he is Professor and Researcher at the Departamento de Computación e Informática at the Universidad de Playa Ancha, Valparaíso, Chile. His research and teaching areas are software engineering, object, aspect, and feature-oriented software development.

Rodolfo Villarroel is Associated Professor at the Escuela de Ingeniería Informática of the Pontificia Universidad Católica de Valparaíso, Chile. He obtained his Ph.D. in computer science from the Universidad de Castilla-La Mancha at Ciudad Real, Spain, and his M.Sc. in computer science from the Universidad Técnica Federico Santa María, Chile. His research activity focuses on data warehouse and information systems security, software process improvement, and aspect-oriented modeling. He belongs to the Chilean Computer Science Society (SCCC).

Rodolfo Schmal Simón is the Industrial Civil Engineer from the University of Chile and Master in Informatics from Polytechnic University of Madrid, Spain. He works as Conference Professor in Business Informatics Engineer School of the University of Talca, Chile. His last publications and academic area of study and interests are data modeling, business process modeling, improvement plans of higher education, and economy of education.

Rodrigo Saens received his Ph.D. in economics from University of Connecticut, USA, his Master's degree in applied economics, B.A. in economics and B.A. in business administration from Pontificia Universidad Católica de Chile. He is Professor at the School of Economics and Business at Universidad de Talca, Chile. His fields of interest include financial and monetary economics, agricultural economics and economic modeling.

Tamara Tigero received her Master's degree in finance from Florida International University, USA and her B.A. in business administration from Pontificia Universidad Católica de Chile. She is Professor at the School of Economics and Business at Universidad de Talca, Chile. Her fields of interest include corporate finance, international finance and business modeling.

Carolina Del Río received her Master's degree in organizational behavior from Universidad Diego Portales, Chile, and her B.A. in business administration from Pontificia Universidad Católica de Chile. She is Professor at the School of Economics and Business at Universidad de Talca, Chile. Her fields of interest include organizational development, organizational behavior and business modeling.