Computing and Informatics, Vol. 32, 2013, 573–593

ASPECT-ORIENTED MODELING: APPLYING ASPECT-ORIENTED UML USE CASES AND EXTENDING ASPECT-Z

Cristian Vidal Silva

Business Informatics Administration
Universidad de Talca
Talca, Chile
e-mail: cvidal@utalca.cl

Rodrigo Saens, Carolina Del Río

Business Administration
Universidad de Talca
Talca, Chile
e-mail: {rsaens, cadelrio}@utalca.cl

Rodolfo Villarroel

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

Communicated by Valentino Vranić

Abstract. Considering predominant aspect-oriented software development (asymmetric AOSD), this paper discusses the application of aspect-oriented UML use case diagrams and formal language AspectZ to part of a classic AOSD case study, the Health-Watcher software system. In addition, this article proposes an extension of AspectZ to reach a new property for asymmetric AOSD which reacts after a schema successfully finishes, or not, showing messages for that situation, with an implicit join point; and a way for generalizing similar operations in a system using AspectZ.

Thus, the main goal of this article is to show the application of and differences between asymmetric aspect-oriented formal and non-formal modeling, and to highlight potential advantages of aspect-oriented formal modeling over aspect-oriented non-formal modeling. First, this article describes the main concepts of the classic AOSD paradigm focusing on problems unsolved by previous forms of software development and resolved by AOSD. Second, by applying aspect-oriented UML use case diagrams, this paper highlights the use of dominion classes and extend-relationships. Considering the Health-Watcher case study and an asymmetric AOSD approach, this study found that using extend-relationships in UML use cases does not completely follow the basic principles of the prevailing AOSD approach in which a base element does not know about aspects, whereas an extending use case must explicitly know its extension points. Third, this article shows a formal modeling of the case study using AspectZ. Moreover, extensions of this aspect-oriented formal language are proposed and applied to the same case study to show their practical properties for modeling. These extensions allow showing success or error messages, and inserting or not a new item in a set of elements to take care of invalid situations.

Keywords: AOSD, UML, aspect-oriented UML use cases, aspects, AspectZ, schema

1 INTRODUCTION

AOSD represents a software development paradigm that gives a solution to present problems, defined as tangling and scattering, common to traditional software development paradigms, including the object-oriented software development (OOSD) paradigm. These problems do not allow achieving a complete modularization in the software development process. In other words, it is not possible to reach a complete separation of concerns [1].

According to a recent study about a symmetric aspect-oriented approach [2], the dominant AOSD approach is defined as asymmetric. In addition, that article indicates that currently there is an academic tendency to study and apply the symmetric version of AOSD. In addition, the study [2] mentions that asymmetric AOSD distinguishes between base elements and external elements or aspects whereas in the symmetric form of AOSD that distinction does not exist because each entity of the system is considered to be an aspect. Nowadays, the main principle of asymmetric AOSD as accepted by industry and academia is to have base and aspect elements where the former do not know of the existence of aspects, and aspects clearly know the circumstances for their occurrence. This article focuses on formal and non-formal modeling of asymmetric AOSD along with potential extensions of the used languages – AspectZ and UML, respectively. Furthermore, possible advantages of formal modeling with respect to non-formal modeling are presented.

AOSD presents its bases so that a complete separation of concerns is achieved in each activity of the software development process – requirements specification,

analysis, design, implementation, and maintenance [3]. AOSD, from its beginning, has allowed a substantial advantage in the last two mentioned activities of the software development process, implementation and maintenance, because AOSD was introduced by a version of an object-oriented programming language and as a new programming paradigm with respect to the existing languages at that time: AspectJ, an extension of Java, and aspect-oriented programming (AOP) [4].

AOSD, as a software development paradigm, deals with each activity of the software development process. In addition, AOSD can be seen as an extension of the object-oriented software development (OOSD) paradigm.

Because UML is an object-oriented modeling language by default [1], and since AOSD is considered to be an extension of OOSD relative to aspect-oriented software modeling, UML use cases have already been extended and used to model asymmetric AOSD requirements [1, 2, 5].

Assuming that AOSD involves all the activities of the software development process, and considering that there are already studies that adapt and extend object-oriented modeling to AOSD [1, 2, 5, 6, 7], there is little evidence of work extending formal languages to support AOSD: AspectZ [8], and Alloy [9]. This study presents modeling of the Health-Watcher system [10] using UML use case diagrams and the aspect-oriented formal language AspectZ. In addition, comments about aspect-oriented UML use cases are given along with extensions of AspectZ.

This article presents an application of formal and non-formal asymmetric aspect-oriented modeling, even though, in practice, the use of both modeling techniques in the software development process is not explicitly required. Furthermore, formal techniques are not usually considered for noncritical systems. According to several studies [3, 5, 13], the use of formal methods requires spending more time to obtain and model system requirements, and that is necessary only for critical systems.

According to a classic reference about Z formal language and its application [11], using formal methods in the requirements stage of developing a software system usually implies spending more time defining the main elements of the system along with their activities, restrictions and relationships. Undoubtedly, by using formal methods a consistent math model can be produced, therefore contradictions in a formal model are easier to find than with the use of non-formal modeling techniques. Definitely, for software systems that require knowing the functionalities and restrictions, working with formal requirements specification is completely necessary. Therefore, spending time to gain knowledge of the system in the first stages of the software development process is essential.

AspectZ represents an extension of the formal language Z. Using Z, all operations and restrictions of the modeled software system are specified and analyzed, and often more attention is paid to special conditions and events than to the application of business logic [11]. On the other hand, AspectZ modeling schemas highly concentrated on business logic allow including aspect-schemas for special situations.

Software is always created after obtaining the user requirements. Assuming that M represents a user requirement, to model M traditional software development paradigms often consider special situations associated with M, such as its restrictions

and consequences, before modeling the essence or center of M. As a result, M is always modeled involving different mingled functionalities. For example, current software usually requires a login to get permission to perform an action with it, therefore that action is considered a special situation when a non-identified user wants to get access, and therefore this situation is modeled.

According to various studies [1, 4, 5, 7, 11], mainly due to the nearly complete separation of concerns, AOSD clearly allows putting attention on business logic to reach a complete modularization to avoid mingled system solutions. Undoubtedly, it is possible to identify and model each base requirement and crosscutting concern in the early phases of the software development process.

Certainly, formal modeling puts more attention on requirements from the early software development phases, and mixing AOSD and formal modeling enables an early identification of concerns.

This article is organized as follows: Section 2 presents UML use cases and AspectZ for AOSD; Section 3 describes the case study, and presents the application of UML use cases for a part of the case study; Section 4 presents an AspectZ formal specification for the case study with extensions of this aspect-oriented formal language; and finally, Section 5 delivers the main conclusions and associated future work.

2 UML USE CASES AND ASPECTZ

This section describes the main or global properties of UML use cases in Subsection 2.1, and of AspectZ in Subsection 2.2. Specifically, Subsection 2.1 shows how to express crosscutting concerns as extendrelationships in UML use case diagrams, and Subsection 2.2 describes the main elements of AspectZ and their relationship to the original Z language.

2.1 UML Use Cases

The UML use case diagram is a fundamental tool for identifying requirements in OOSD. In that AOSD is an extension of OOSD, the UML use cases application is desirable in AOSD because, as a software modeling tool, UML use cases allow for early crosscutting concerns identification represented by means of alternative flows or extend-relationships [1].

It is important to consider that UML use cases do not represent only a technique for requirements modeling. UML use case diagrams are a software engineering technique that completely directs the software development process [1, 5]. Identifying crosscutting concerns is very relevant in the software development process because early identification also allows early validation. In this way, UML use case diagrams are a useful tool for asymmetric AOSD.

Figure 1 presents a simple modeling example using UML use cases for a login information system. This figure shows that an extend-relationship ($\ll$ *extend* $\gg$)

allows modeling crosscutting concerns. According to a UML reference [12], an extending use case provides an additional behavior to augment the behavior of a base or extended use case, and the extending use case knows when it must react or add the new behavior to the base case. Moreover, when an extending use case finishes, it notifies the base use case which resumes its execution from the point where it was suspended. Thus, with a graph or text, an extend-relationship in a UML use case diagram captures the nature of aspects in asymmetric AOSD.

As an additional issue, the existence of an extend-relationship in a UML use case diagram clearly represents a crosscutting concern because the behavior of an extending use case is not directly a part of a base or extended case.

According to UML references [1, 5, 13], due to the fact that UML use cases allow capturing and modeling user requirements, and identifying crosscutting concerns represented by extend-relationships, aspect-oriented UML use case diagrams permit the early identification and modeling of crosscutting concerns in the user requirements phase.

UML use cases allow knowing and modeling user requirements in order to make advances in the desired behavior of a software system [1, 5, 12, 13]. In addition, there are arguments that UML use case diagrams allow modeling inclusion and inheritance relationships as extend-relationships [5, 12]. Therefore, from an abstract point of view, it is possible to model a complete software system using the aspect-oriented approach and UML use cases with extend-relationships.

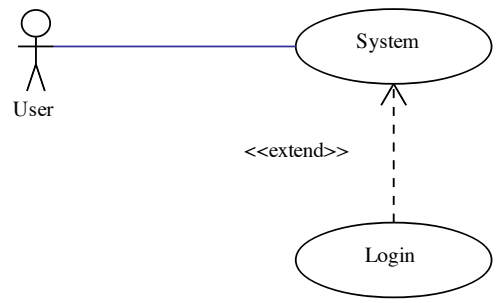


Fig. 1. UML use cases for a login information system [1, 5]

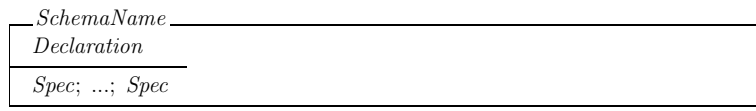
For an effective use of aspect-oriented UML use cases, it is necessary to identify the domain of the system which is represented by a dominion class diagram [1, 5, 13]. A dominion class diagram of a software system identifies global classes and their system-wide relationships to know the overall scope of the system, its functions, and restrictions [1, 5]. Therefore, it is advisable to identify the dominion classes of a software system before modeling it with UML use cases, but this is not absolutely necessary for every system.

Clearly, the extension of UML use case diagrams allows modeling with an asymmetric AOSD view. For a symmetric AOSD approach [2] it indicates and exemplifies

the use of peer UML use cases in which there is no difference between aspects or modules of the system.

2.2 AspectZ

AspectZ is an extension of the Z language which was originally based on the models theory [14, 15]. AspectZ, like the Z language [11], uses schemas to represent the properties and operation of a software system; however, AspectZ adds new concepts to support asymmetric aspect-oriented modeling. AspectZ, like the Z language, uses base, initial, and operation schemas for a software system's formal specification and aspect-schemas are also included to indicate properties and behavior to be added in referenced operation schemas. Figure 2 shows and describes an original AspectZ schema.



Declaration ::= BasicDecl; ...; BasicDecl
BasicDecl ::= Ident, ..., Ident: Expr
| ΔSchemaRef
| ΩSchemaRef
| PointcutDecl
SchemaRef ::= SchemaName Decoration[Renaming]
PointcutDecl ::= PointcutIdent : ℙ(Ident: Expr)
Spec ::= *Predicate* | *Advice*
Advice ::= [*insert* | *replace*] PointcutName : Predicate

Fig. 2. Original AspectZ schema [8]

Supporting the main ideas of asymmetric AOSD, AspectZ allows dividing a traditional Z schema between base and crosscutting elements so that obtaining a separation of concerns for these schemas is possible [5]. Using AspectZ, an operation schema is a model that considers only its main steps, and additional details or special situations are modeled by aspect-schemas. Clearly, thanks to early aspects detection, AspectZ allows a complete modularization at the first stage of the software development process, the requirements analysis phase [8, 15]. The main new elements of AspectZ are the use of the W symbol in aspect-schemas to refer to operation schemas, declaration of pointcut to refer to join points, and the use of the Insert and Replace keywords to indicate whether an aspect-schema behavior is added prior to the indicated schema operation or the aspect-schema behavior modifies a certain element of the referenced schema.

Furthermore, the use of a schema to indicate the initial state of a system, generally used to establish initial values for attributes of the system, is supported by AspectZ as it is in the traditional language Z [15].

Definitely, AspectZ permits working with the main concepts of asymmetric AOSD because base operations do not deal with aspects and associated concerns. Aspect-schemas clearly know when they must react and include behavior and properties in base elements in the software system.

3 CASE STUDY AND APPLYING ASPECT-ORIENTED UML USE CASES

Section 3.1 gives an overall description of the Health-Watcher system, and that part of the Health-Watcher system which is the case study. In addition, Section 3.2 presents the UML use case models for the case study with graphical notations and textual descriptions for each use case in the model. It is important to note that Figures 3 to 28 are aspect-oriented UML use case or AspectZ models of the case study or proposed extensions of AspectZ, and they have been created for this article.

3.1 Case Study: Health-Watcher System

Health-Watcher is a public health software system [10]. Health-Watcher represents an example of a software system that shows the advantages of the AOSD paradigm and its application. Next, some of the Health-Watcher system requirements are described and used in Sections 3.2 and 4 to present aspect-oriented UML use cases and AspectZ models, respectively.

Health-Watcher is a public system available without restriction for two main users-actors [10]:

- Attendant: health system employee.
- Citizen: any person who wishes to interact with the system.

Health-Watcher supports citizen queries; a citizen can ask for information within the *Health Guide* and for *Specialty Information*.

This article uses the requirements associated with the Health Guide [10] in which there are two queries:

1. Which health units take care of a specific specialty?
2. What are the specialties of a particular health unit?

Directly, there is a relationship between units and specialties. Moreover, unit and specialty have two attributes, Code and Description.

Due to the identification of dominion classes of the described part of the Health-Watcher system a simpler definition of the associated UML use cases diagram is supplied. Clearly for this case study, as Figure 3 shows, there are two classes, Unit and Specialty.

These requirements will be used in the next sections to model aspect-oriented UML use cases and AspectZ.

Fig. 3. UML dominion class diagram of a considered part of the Health-Watcher system

In the Health-Watcher system, it is supposed that Attendant will provide information to the system while Citizen will seek information related to queries.

3.2 Aspect-Oriented UML Use Case Models of the Health-Watcher System

Section 2 presented a description of UML use case diagrams adapted to asymmetric AOSD where the main difference with traditional UML use case diagrams is the use of extend-relationships for any relationship.

Figure 4 presents the UML use cases diagram for analysing considered requirements of the Health-Watcher system applying asymmetric aspect-oriented modeling.

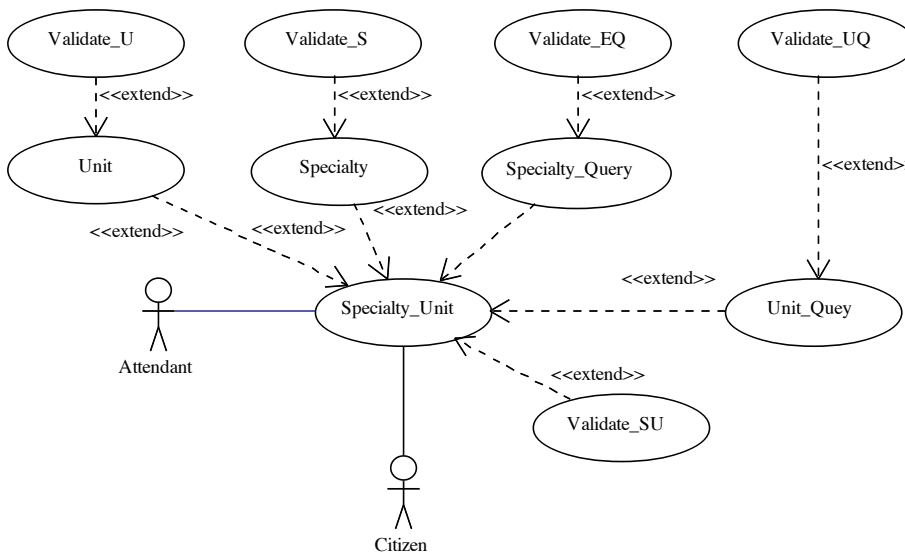


Fig. 4. UML use cases diagram of a considered part of the Health-Watcher system

Figure 4 presents many extend-relationships which allow connecting the base cases to external situations or additional behavior. For example, an Attendant and Citizen access the base use case Specialty_Unit for different purposes: an Attendant actor is in charge of managing the current information of Unit and Specialty and

has access to queries as well, meanwhile a Citizen actor asks for information about Specialty and Unit only. Clearly, Figure 4 does not show what operations are reachable for each actor, Citizen or Attendant. However, a textual description of each use case gives those details. Thus, without a doubt the use of both graphical notation and textual descriptions are necessary in a UML use case model for a complete explanation and understanding of its elements.

Figures 5, 6, 7, and 8 show the specification of Specialty_Unit, Validate_SU, Unit, and Validate_U use cases, respectively. Figures 9 and 10 present the specification for Specialty_Query and Validate_SQ use cases.

Clearly, the use case Unit should present extension points for each additional operation over the current Units which is not detailed in this article. Here, only Validate_U is considered to validate the operations in the Unit set.

Use Case: Specialty_Unit

Basic Flow → The Use Case starts when a user, Citizen or Attendant, desires to register information or get information.

1. User gets in the system.
2. User indicates his or her operation.
3. Use Case *finishes*.

Extension Points:

E1. User Gets in. This extension point occurs on the step 1.
E2. Attendant Chooses Unit. This extension point occurs on the step 2.
E3. Attendant Chooses Specialty. This extension point occurs on the step 2.
E4. Citizen Choose Specialty_Query. This extension point occurs on the step 2.
E5. Citizen Choose Unit_Query. This extension point occurs on the step 2.

Fig. 5. Textual specification of the Specialty_Unit use case

Furthermore, UML use case diagrams certainly allow introducing the base concepts and ideas of asymmetric AOSD. However, extension points must be explicitly indicated in the base case. This action is not part of the asymmetric AOSD paradigm. The next section presents an AspectZ model for a part of the case study shown in this section. AspectZ completely applies elements of AOSD preserving the main ideas of the asymmetric AOSD paradigm.

4 APPLYING AND EXTENDING ASPECTZ

This section presents formal modeling of part of the Health-Watcher system, and also gives extensions to this formal language and applies them to parts of the case study as well. With these additional elements, AspectZ represents a more complete aspect-oriented language than it was originally.

Use Case: Validate_SU

Basic Flow → This extended flow occurs at a given extension point; when a user, Citizen or Attendant, accesses the system and the user gets E1 (User gets in).

1. User presents his or her information.
2. If user is a valid Citizen, Use Cases Specialty_Query and Unit_Query are enabled; Unit and Specialty are disabled.
3. If user is a valid Attendant, all Use Cases are enabled.
4. If user is not valid, the Base Case must finish.
5. Use Case finishes.

Extension Pointcuts → E1 = User gets in.
Extension Points:

Fig. 6. Textual specification of the Validate_SU use case

Use Case: Unit

Basic Flow → This extended flow occurs at a given extension point, where the Attendant chooses Unit (E2).

1. Attendant indicates operation over the current set.
2. The Base Case must restart.
3. Use Case Finishes.

Extension Pointcuts → E2 = Attendant chooses Unit.
Extension Points:

E6. Validate_Unit. This extension point occurs in step 1.

Fig. 7. Textual specification of the Unit use case

Use Case: Validate_U

Basic Flow → This extended flow occurs at a given extension point, when an Attendant chooses an invalid operation over the current Units set (E6).

1. Indicate valid operations.
2. Show Error message.
3. The Base Case must restart.
4. Use Case Finishes.

Extension Pointcuts → E6 = Attendant chooses invalid operation.
Extension Points:

Fig. 8. Textual Specification of the Validate_U use case

Use Case: Specialty_Query

Basic Flow → This extended flow occurs at a given extension point, when a Citizen or Attendant chooses Specialty_Query to find specialty work carried out in a medical Unit (E4).

1. User Citizen or Attendant indicates Specialty information.
2. Set of Specialties is given.
3. The Base Case must restart.
4. Use Case Finishes.

Extension Pointcuts → E4 = Citizen or Attendant chooses Specialty_Query.
Extension Points:

E7. Validate_SQ. This extension point occurs in step 1.

Fig. 9. Textual Specification of the Specialty_Query use case

Use Case: Validate_SQ

Basic Flow → This extended flow occurs at a given extension point, when a Citizen or Attendant gives wrong Specialty information.

1. Indicate valid operations.
2. Show Error message.
3. Use Case Finishes.
4. The Base Case must restart.

Extension Pointcuts → E7 = Citizen or Attendant give wrong specialty information.
Extension Points:

Fig. 10. Textual Specification of the Validate_SQ use case

4.1 Formal Modeling of a Part of the Health-Watcher System Using AspectZ

AspectZ allows a complete separation of concerns [8, 15]. AspectZ, like language Z, allows using basic and composed data types. Figure 11 shows the declaration of data types used on the AspectZ specification for a part of the Health-Watcher system, hereafter called Health_Watcher.

Basic types Unit_Code and Unit_Description are declared to define the composed type called Unit. Likewise, basic types Specialty_Code and Specialty_Description to define the composed type Specialty are declared. There is an additional composed type Health_Unit which is declared using the basic types Unit_Code and Specialty_Code. The objective of Health_Unit is to present a relationship between Unit and Specialty. There are no multiplicity restrictions for this relationship. There is

also a data type for the type of query, Query_Type, which can be 1 for Q1 (which health units take care of a specific specialty?), and 2 for Q2 (What are the specialties of a particular health unit?). Similarly, there is a data type to recognize the kind of operation over the sets of Unit, Specialty, and Health_Unit (1, 2, and 3, respectively) where the possible operation is indicated by number after the number of the affected set. The operation values are .1 for getting a new element, .2 to modify an existing element, and .3 for deleting an existing element. For example, to identify an operation creating a new Unit, the Input_Type value is 1.1. In addition, there is a data type for the kind of messages given after an operation, OK for successful operations and Error for operations that are wrong or without effect.

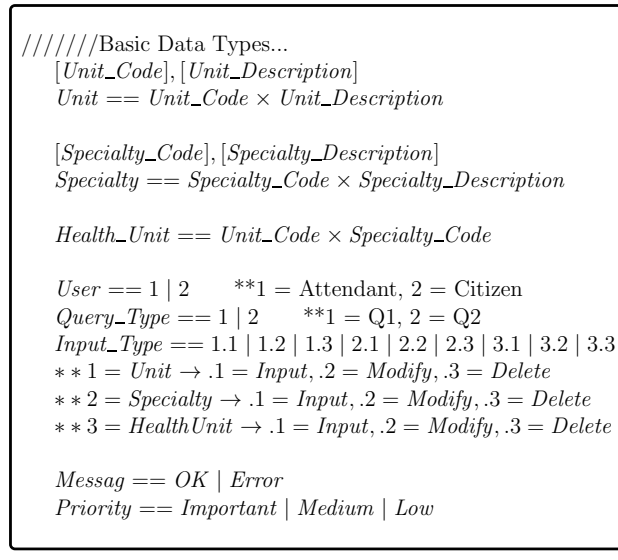


Fig. 11. Basic and composed data types used in AspectZ specification

Figure 12 presents the base schema in which there are sets of Unit, Specialty, and Health_Unit, respectively. In the second part of the base schema, there is an invariant indicating that for each element of HUs there is an element of Units and an element of Specs.

Figure 13 shows the Begin schema or initial state of the Health_Watcher system. As shown, the second part of this schema indicates that the set HUs is empty at the beginning, and this state agrees with the invariant defined in the base schema.

Next, Figure 14 shows a schema using original Z for the first Citizen query operation, Q1.

Clearly, in this AspectZ specification, an aspect-schema can be used to identify the given Query_Type and Specialty_Description, and if the conditions are respected then R01 will finish successfully. However, this solution is not general because it

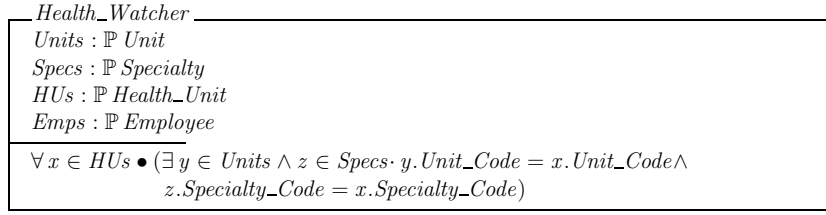


Fig. 12. Base schema of the Health_Watcher system AspectZ specification

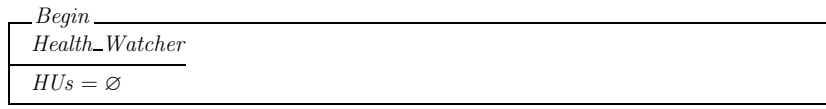


Fig. 13. Initial schema of the Health_Watcher system AspectZ specification

represents only a particular case (when the Query_Type variable T? takes a certain value). Figures 15 and 16 show the first attempt at modeling R01 using AspectZ for the base and aspect-schema, respectively.

4.2 Extensions of AspectZ

According to a study about rules for applying formal methods [16], formal languages, like Z and AspectZ, permit modeling and thinking of system operations more deeply than other modeling techniques. Considering asymmetric AOSD ideas, for example, it is possible to extend AspectZ to support messages at the end of an operation. There are two possible situations for each operation or schema: a successful operation with an OK message and unsuccessful operation with an error message. Therefore, the keyword *After* (similar to Insert and Modify) is included in AspectZ to add *Advice* at the end of an operation schema where it is possible to evaluate two cases: *FINISH* (valid preconditions and operation schema finishes) or *NOT_FINISH* (preconditions are not valid). It is necessary to use a special composition symbol $\pm$ to indicate that a schema is woven in at the end. In that sense, the symbol $\pm$ is used at the end of a composition. Accordingly, it is not completely necessary to identify a join point because it would be at the end of the schema for a successful operation, or where a schema precondition is not valid. Similarly, it is necessary to include the keyword *END* to finish an operation. Figure 17 presents the aspect-schema for a message after finishing successful or unsuccessful operations. Figure 18 shows the integrated aspect for the first Citizen query using the aspect AspectMessage.

Figure 19 shows an original Z schema for the second Citizen query operation, Q2. Similar to R01, for R02 there is an Aspect to validate T?. Figure 20 presents the AspectZ base schema for the R02 operation. Figure 21 presents the aspect-schema for the R02 operation schema, and Figure 21 shows the IntegratedR02 schema.

$R01$

$\Xi Health\_Watcher$
$T? : Query\_Type$
$E? : Specialty\_Description$
$U! : \mathbb{P}\ Unit$

$T? = 1 \land$
$U! = \{x : Unit, y : Health\_Unit, z : Specialty \mid (x \in Units \land y \in HUs \land z \in Specs) \land$
$z.Specialty\_Description = E? \land y.Specialty\_Code = z.Specialty\_Code \land$
$y.Unit\_Code = x.Unit\_Code \bullet x\}$

Fig. 14. Z schema for the First Citizen query

$R01$

$\Xi Health\_Watcher$
$E? : Specialty\_Description$
$U! : \mathbb{P}\ Unit$

$T? = 1 \land$
$U! = \{x : Unit, y : Health\_Unit, z : Specialty \mid$
$(x \in Units \land y \in HUs \land z \in Specs) \land$
$z.Specialty\_Description = E? \land$
$y.Specialty\_Code = z.Specialty\_Code \land y.Unit\_Code = x.Unit\_Code \bullet x\}$

Fig. 15. AspectZ base schema for the First Citizen query

After reviewing the Citizen queries, clearly considering them as an important priority (Priority = Important), it is possible to define a Citizen_Query schema as Figure 18 illustrates.

After reviewing and proposing the mentioned AspectZ extensions as well as examining the Health_Watcher Java source code, there is a new proposal for extending AspectZ. Using AspectZ it is possible to individualize aspects for each element, but in the Health_Watcher source code there is only one general aspect for all the associated classes. This aspect allows insertion operations: HDWDataCollection. Following this idea, AspectZ is extended to allow only one general operation for adding an element in a set with aspects to validate the presence of wrong situations such as required attributes with wrong values. Figure 23 shows a theoretical Add_Operation which represents all operations to add new items in a system set.

$AspectR01$

$\Omega R01$
$T? : Query\_Type$
$PointcutPC : \{E? : Specialty\_Description\}$

$InsertPC : T? = 1 \land * \in \{x : Specialty \mid x \in Specs \bullet x.Specialty\_Description\}$

Fig. 16. AspectZ aspect-schema for the First Citizen query

Figure 24 presents a schema to validate information. In this schema, Inserted_Values represents the set of all inserted values, function Required(x) indicates if x can be Null or not, function ThereIsRelationship(x) shows if there is an established relationship of sets by means of the x type, function Type(x) gives the data type for the value x, and function Relationship(x) indicates whether the value of x is a valid value supporting an existing relationship. Those functions are shown in Figures 25, 26, 27, and 28, respectively. It is important to know that these functions must be defined in AspectZ as additional functions because they are not directly a part of the language.

AspectMessage

$\Omega(Schema1, Schema2, ..., SchemaN)$
$PointcutPC\_M$
$M! : Message$

$AfterPC\_M : FINISH \land M! = OK \lor M! = Error$

Fig. 17. AspectZ AspectMessage schema

$Citizen\_Query \mathrel{\widehat{=}} (R01 \lor R02) \land Priority = Important$
$IntegratedR01 == (R01 + AspectR01) \pm AspectMessage$

IntegratedR01

$\Xi Health\_Watcher$
$E? : Specialty\_Description$
$U! : \mathbb{P}\ Unit$
$T? : Query\_Type$
$M! : Message$

$(T? = 1 \land E? \in \{x : Specialty \mid x \in Specs \bullet x.Specialty\_Description\}) \land$
$(U! = \{x : Unit, y : Health\_Unit, z : Specialty \mid (x \in Units \land y \in HUs \land z \in Specs) \land$
$z.Specialty\_Description = E? \land$
$y.Specialty\_Code = z.Specialty\_Code \land y.Unit\_Code = x.Unit\_Code \bullet x\} \land M! = OK) \lor M! = Error$

Fig. 18. AspectZ IntegratedR01 schema

In the Aspect_Add schema, there are three important elements necessary to be included in the AspectZ formal language: function Required(x) detailed in Figure 25, function ThereIsRelationship(x) detailed in Figure 26, and Relationship(x) detailed in Figure 28. There is an indirect function Type(x) to reveal the associated data type of the parameter x detailed in Figure 27.

Clearly, the Required(x) function gives the True value if x does not accept a Null value by definition of the associated data type. Type(x) indicates the type of the x attribute by definition. ThereIsRelationship returns True, if x represents a value of an attribute which is related to another set – in other words, if there is a defined relationship of the data type x, set of attributes Set*, and another data type, set

$$
\begin{array}{|l}
\hline
R02 \\
\hline
\Xi Health\_Watcher \\
T? : Query\_Type \\
U? : Unit\_Description \\
P! : \mathbb{P}\, Specialty \\
\hline
T? = 2 \land E! = \{x : Specialty, y : Health\_Unit, z : Unit \mid (x \in Specs \land y \in HUs \land z \in Units) \land \\
z.Unit\_Description = U? \land y.Unit\_Code = z.Unit\_Code \land \\
y.Specialty\_Code = x.Specialty\_Code \bullet x\} \\
\hline
\end{array}
$$

Fig. 19. Z schema for the Second Citizen query

$$
\begin{array}{|l}
\hline
R02 \\
\hline
\Xi Health\_Watcher \\
U? : Unit\_Description \\
P! : \mathbb{P}\, Specialty \\
\hline
E! = \{x : Specialty, y : Health\_Unit, z : Unit \mid (x \in Specs \land y \in HUs \land z \in Units) \land \\
z.Unit\_Description = U? \land y.Unit\_Code = z.Unit\_Code \land \\
y.Specialty\_Code = x.Specialty\_Code \bullet x\} \\
\hline
\end{array}
$$

Fig. 20. AspectZ base schema for the Second Citizen query

$$
\begin{array}{|l}
\hline
AspectR02 \\
\hline
\Omega R02 \\
T? : Query\_Type \\
PointcutPC2 : \{U? : Unit\_Description\} \\
\hline
InsertPC2 : T? = 2 \ \land * \ \in \{x : Unit \mid x \in Units \bullet x.Unit\_Description\} \\
\hline
\end{array}
$$

Fig. 21. AspectZ Aspect-Schema for the Second Citizen query

$$IntegratedR02 == (R02 + AspectR02) \pm AspectMessage$$

$$
\begin{array}{|l}
\hline
IntegratedR02 \\
\hline
\Xi Health\_Watcher \\
T? : Query\_Type \\
U? : Unit\_Description \\
P! : \mathbb{P}\, Specialty \\
M! : Message \\
\hline
(T? = 2 \land x : Unit \mid x \in Units \bullet x.Unit\_Description) \land \\
(E! = \{x : Specialty, y : Health\_Unit, z : Unit \mid (x \in Specs \land y \in HUs \land z \in Units) \land \\
z.Unit\_Description = U? \land y.Unit\_Code = z.Unit\_Code \land \\
y.Specialty\_Code = x.Specialty\_Code \bullet x\} \land M! = OK) \lor M! = Error \\
\hline
\end{array}
$$

Fig. 22. AspectZ IntegratedR02 schema

Add_Operation

$\Delta Health\_Watcher$
$T? : Input\_Type$
$Key\_Val? : Unit\_Code$
$Field\_1? : Field\_Type1$
...
$Field\_N? : Field\_TypeN$

$T? = Value \land SetX' = SetX \cup \{(Key\_Val?, Field\_1?, ..., Field\_N?)\}$

Fig. 23. AspectZ Add_Operation schema

Aspect_Add

$\Omega(Add\_Op1, ..., Add\_OpN)$
$Set : Set1 \mid Set2 \mid ... \mid SetN$
$M! : Message$
$PointcutPC : (SetX' = SetX \cup \{Values\})$

$Insert\ PC : \exists\, x \in Inserted\_Values \mid (Required(x) \lor ThereisRelationship(x)) \land$
$(x = \varnothing \lor (Relationship(x) = False))\ \bullet\ M! = Error \land END$

Fig. 24. AspectZ Aspect_Add schema

of attributes Set**. Finally, Relationship(x) indicates, in the case that x is part of a relationship, whether the value of x fits the established relationship among sets.

Regarding ThereIsRelationship(x), it is necessary to define the function Type(x) as Figure 27 shows.

It is important to indicate that the proposed extensions shown in Figures 25, 26, 27 and 28, which are applied in Figure 24 – Aspect_Add (global schema), are elements necessary to add in the AspectZ language. Possibly, these elements could be members of a library to use in an AspectZ specification.

After applying AspectZ, clearly there is a complete symmetric aspect-oriented understanding and model of the system, thus allowing to continue with asymmetric aspect-oriented modeling and implementation. In addition, as this article shows, using AspectZ does not require a high level of math skills, and by applying it a complete separation of concerns can be obtained.

$$Required(x) = \begin{cases} True & \text{if x can not be Null} \\ False & \text{if x can be Null} \end{cases}$$

Fig. 25. Required(x) function

$$ThereIsRelationship(x) = \begin{cases} True & \text{if } (\text{Type(x)} = Attr_i) \wedge (Attr_i \in Set^*) \wedge \\ & (Set^* == Atr1 \times ... \times Attr_N) \wedge \\ & \left\{ \begin{array}{l} \exists\ (Set^{**} = Attr_{*1} \times ... \times Attr_{*N}) \wedge \\ \exists\ Attr_{*j} \in Set^{**} \mid Attr_i = Attr_{*j} \end{array} \right\} \\ False & \text{in other case} \end{cases}$$

Fig. 26. ThereIsRelationship(x) function

$$Type(x) = \{\, Type\_X, \textit{at the base case } x \textit{ is defined like } x : \ Type\_X$$

Fig. 27. Type(x) function

5 CONCLUSIONS

AOSD permits modeling a software system and reaching complete modularization. This study presents the application of the UML use cases and AspectZ languages to a known asymmetric AOSD case study. Both tools allow modeling following the main rules of asymmetric AOSD. However, on this point, aspect-oriented UML use case diagrams do not completely respect the asymmetric AOSD paradigm due to base use cases clearly knowing about the existence of their aspects, even though this is a clear characteristic of textual representation of extend-relationships of UML use case diagrams. Thus, using aspect-oriented UML use cases, base and aspect functionalities are identified and modeled, base cases and extension cases respectively, but they are not completely independent because aspects must be explicitly indicated in each base or extended base case. On the other hand, AspectZ as an extension of the formal language Z respects asymmetric AOSD principles not respected by aspect-oriented UML use cases.

It is relevant to argue that since the proposal of AspectZ, there has not yet been an article mentioning the use or extension of this formal language. This article proposes syntactic and semantic extensions of AspectZ to weave in aspect-schemas at the end of base schemas which can be used to put messages in the specification or similar operations, and a review including a conceptual extension to use only one aspect-schema involving similar operations on the system-wide schema, to add items to the system in this case. Likewise, updating and deleting information are examples of using the idea of system-wide schemas.

$$Relationship(x) = \begin{cases} True & \text{if } ThereIsRelationship(x) = True \textit{ for } (Set^* \wedge Set^{**}) \wedge Type(x) \in Set^* \wedge \\ & \textit{there is an } Attr_{*j} \in Set^{**} \mid x \in \{Attr_{*j}\} \\ False & \text{in other case} \end{cases}$$

Fig. 28. Relationship(x) function

Knowing that the main benefit of using formal modeling is discovering mistakes in a software system for their correction and solution as early as possible in the software development process, the main goal of this article was to discuss the application of and giving extensions to the AspectZ formal language. Therefore, in this article, after modeling part of the Health-Watcher system, it is not possible to indicate whether this system presents mistakes in its modeling and implementation. Nevertheless, the potential advantages of aspect-oriented formal modeling with respect to non-formal modeling were mentioned where AspectZ adheres to the main principles of asymmetric AOSD.

Considering ideas of including formal modeling in the software development process, including a verification of consistency among models, a process of formal refinement models will be essential to consider in the short-term. Thus, as a future work the authors of this article are considering evaluating and defining steps for an aspect-oriented formal refinement process with AspectZ.

In following up this work, the authors wish to use and extend AspectZ for defining more applications of the AOSD paradigm. In addition, ideas for producing a public tool for AspectZ modeling and automatically generating source code from an AspectZ specification are also part of the current goals of the authors. This goal is similar to previous experiences using JAXB and ZML [17]. Using ZML and JAXB as a base to produce a similar tool to produce AspectJ code from an AspectZ specification seems completely viable given that they share asymmetric AOSD principles and take into consideration previous mentioned works [17].

Furthermore, to verify the correctness and validity of an AspectZ specification, considering formal verification, as current research, one of the authors is working on the translation of AspectZ specification to Alloy specifications to use the Alloy analyzer tool [9] and verify correctness of AspectZ formal specifications.

Acknowledments

To Sharon Goulart for reviewing the English in this article, and giving assistance to continue working and producing a publishable version of this paper. Definitely, we continue being great friends and colleagues.

REFERENCES

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

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

[3] Pressman, R. S.: Software Engineering, a Practitioner's Approach. McGraw-Hill Higher Education, 5th edition, 2001.

[4] Laddad, R.: AspecJ in Action: Practical Aspect-Oriented Programming. Manning Publications Co., London, England, 2003.

[5] Vidal, C.—Hernández, D.—Pereira, C.—Del Río, M.: Aspect-Oriented Modeling Application. Información Tecnológica Journal, Vol. 23, 2012, No. 1, La Serena, Chile, pp. 3–12.

[6] Skipper, M. C.: Formal Models for Aspect-Oriented Software Development. London Imperial College, Thesis of Ph. D. in Computer Science, London, England, 2004.

[7] Li, G.: Identifying Crosscutting Concerns in Requirement Specifications – A Case Study. Queen's University, Thesis of Master in Computer Sciences, Kingston, Ontario, Canada, 2009.

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

[9] Nakajima, J.—Tamai, T.: Lightweight Formal Analysis of Aspect-Oriented Models. Proceedings of the 5th Aspect-Oriented Modeling Workshop In Conjunction with UML 2004, Lisboa, Portugal, October 2004, pp. 120–127.

[10] Soares, S.—Laureano, E.—Borba, P.: Implementing Distribution and Persistence Aspects with AspectJ. Proceedings of OOPSLA '02 (Object-Oriented Programming, Systems, Languages, and Applications), ACM New York, NY, USA, 2002, pp. 174–190.

[11] Bowe, J.: Formal Specification and Documentation Using Z: A Case Study Approach. Revised Edition, International Thompson Computer Press (ITCP), London South Bank University, England, 2003.

[12] Pender, T.: UML Bible. 1st Edition, John Wiley & Sons, Indianapolis, IN, USA, 2003.

[13] Ambler, S.: Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process. 1st Edition, John Wiley & Sons, New York, NY, USA, 2002.

[14] Vidal, C.—Andrades, M.: Formal Specification of a Digital Electrocardiograph using Object-Z. Proceedings of the III Meeting of Quality on Informations and Communication Technology, Cuba, February 2007.

[15] Vidal Silva, C.—Gutiérrez Castillo, C.—Hernández Bustos, D.—Meza Torres, R.—López Lastra, L.: Aspect-Oriented Formal Modeling Using Aspect-Z. Proceedings of the XXIII Chilean Computing Meeting, Curico, Chile, November 2011.

[16] Bowen, J. P.: Ten Commandments of Formal Methods... Ten years later. IEEE Computers, January 2006, pp. 40–48.

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

Cristian Vidal Silva is a Chilean Fulbright Scholar. He has received his Computer Engineer degree from Catholic University of Maule, Chile, his M. Sc. in computer science from University of Concepcion, Chile, and currently he is a Ph. D. student in computer science at Michigan State University, USA. He is a Professor of business informatics administration at University de Talca, Chile. His research and teaching areas include formal modeling, aspect-oriented software development, and programming.

Rodrigo Saens received his Ph. D. in economics from University of Connecticut, USA, his M. Sc. in applied economics, B. A. in economics and B. A. in business administration from Pontificia Universidad Católica de Chile. He is a Professor at Universidad de Talca, Chile. His fields of interest include financial economics, monetary economics, infonomics and agricultural economics.

Carolina Del Río received her M. Sc. in organizational behavior from the Universidad Diego Portales, Chile, and her B. A. in business administration from Pontificia Universidad Católica de Chile. She is a Professor at Universidad de Talca, Chile. Her fields of Interest include organizational development, organizational behavior, strategic management and management information systems.

Rodolfo Villarroel in an Associate Professor at Escuela de Ingeniería Informática at Pontificia Universidad Católica de Valparaíso, Chile. He received his Ph. D. in computer science from Universidad de Castilla – La Mancha at Ciudad Real, Spain, and his M. Sc. in computer science from Universidad Técnica Federico Santa María, Chile. His research interests include security in data warehouses and information systems, software process improvement, and aspect-oriented modeling. He is the author of several papers on data warehouses security and software process improvement, and aspect-oriented modeling. He is a member of the Chilean Computer Science Society (SCCC).