

# Formal Aspect-Oriented Modeling and Analysis by AspectZ<sup>\*</sup>

Huiqun Yu<sup>1</sup>, Dongmei Liu<sup>1</sup>, Li Yang<sup>2</sup>, Xudong He<sup>2</sup>

<sup>1</sup> Department of Computer Science, East China University of Sci & Tech, Shanghai 200237, China  
{yhq|dmliu}@ecust.edu.cn

<sup>2</sup> School of Computer Science, Florida International University, Miami, FL 33199, USA  
{lyang03|hex}@cs.fiu.edu

## Abstract

*Separation of concerns is one of the software engineering design principles that is getting more attention from practitioners and researchers in order to promote design and code reuse. Aspect-oriented programming is a maturing technique to enhance concern modularization and integration, which arouses great interest in both research society and software industry. However, less attention has been paid to modeling and quality assurance of aspect-oriented software development method. This paper proposes a formal aspect-oriented modeling language called AspectZ, and an aspect-oriented modeling method in AspectZ. The basic idea is to provide means for observing behaviors of Z schemas and depicting their interrelationships, and to provide ways for weaving interrelated schemas. Correctness of aspect weaving can be formally verified by the reasoning mechanisms of Z notation.*

**Keywords:** *Aspect orientation, formal method, modeling, analysis*

## 1 Introduction

Aspect-oriented programming (AOP) [6] is a new programming paradigm which aims at improving separation of concerns in programs by providing new kind of modules and new ways of composition. Several successful aspect-oriented techniques have been proposed in literature, such as adaptive programming [9], AspectJ [5], Composition Filters [1], and multi-dimensional separation of concerns [10].

Most investigations so far have focused on language constructs for aspect description and aspect weaving at code

level. However, the significance of aspect-oriented design (AOD) at more abstract level has come to front recently [12], because quality assurance in the early stage of software life cycle can result in better design, and shorter time to market. Although some works have been done to identify high level aspects [4, 14, 7], they fell short in precise notations for expressing different concerns and for manipulating these concerns in a systematic fashion.

This paper proposes a formal aspect-oriented modeling language called AspectZ, and an aspect-oriented modeling method in AspectZ. We lift aspect-oriented method from code level to design level, which enhances quality assurance in the early stage of software life cycle. AspectZ is an extension to Z [13]. The basic idea is to provide means for observing behaviors of Z schemas and depicting their interrelationships, and to provide ways for weaving interrelated schemas. Correctness of aspect weaving can be formally verified by reasoning mechanisms of Z notations.

The contributions of this paper include:

1. proposing the formal aspect-oriented modeling language AspectZ, and
2. applying the AspectZ method to modeling and analyzing aspect-oriented software design.

The rest of the paper is organized as follows: Section 2 presents an aspect-oriented modeling method based on AspectZ. Section 3 provides a case study of applying the AspectZ method to modeling and analyzing role-based access control. Section 4 is the conclusion.

## 2 AspectZ: An Aspect-Oriented Modeling Method

### 2.1 Schemas in AspectZ

In the Z notation [16] there are two languages: the mathematical language and the schema language. The mathe-

---

<sup>\*</sup>This work was partially supported by the NSF of China under grants No. 60473055 and 60373075, the NSF of the USA under grant HRD-0317692, and the NASA of the USA under grant NAG 2-1440.

mathematical language is used to describe various aspects of a design: objects, and the relationships between them. The schema language is used to structure and compose descriptions: collating pieces of information, encapsulation them, and naming them for reuse.

AspectZ is a specification language to extend Z with aspect notations. The schema in AspectZ has the general form:

<i>SchemaName</i>
<i>Declaration</i>
<i>Spec</i> ; ...; <i>Spec</i>

*Declaration* ::= *BasicDecl*; ...; *BasicDecl*

*BasicDecl* ::= *Ident*, ..., *Ident* : *Expr*  
| *SchemaRef*  
|  $\Omega$ *SchemaRef*  
| *PointcutDecl*

*SchemaRef* ::= *SchemaName* *Decoration*[*Renaming*]

*PointcutDecl* ::= *Pointcut Ident* :  $\mathbb{P}$ (*Ident* : *Expr*)

*Spec* ::= *Predicate* | *Advice*

*Advice* ::= [*insert* | *replace*]*PointcutName* : *Predicate*

The schema in AspectZ is similar to that in classical Z. When a name has been attached to a schema, it can be used in a schema reference to refer to the schema by *SchemaRef*. A schema reference consists of a schema name, followed by a decoration (which is a possible empty sequence of ', ?, ! characters and subscript digits), and an optional list of renaming.

Information contained in schemas may be combined in a variety of different ways. There are several logical operators in Z: conjunction ( $\wedge$ ), disjunction ( $\vee$ ), negation ( $\neg$ ), quantification, and composition ( $\circ$ ) [13]. The major schema operator used in this paper is *pipe* ( $\gg$ ), which describes the effect of one schema's output is consumed by another schema as input. For example, suppose that *OpOne* and *OpTwo* are introduced by the following two schemas.

<i>OpOne</i>
$a? : A$ $b! : B$
$P(a?, b!)$

<i>OpTwo</i>
$b? : B$ $c! : C$
$Q(b?, c!)$

The combination *OpOne* $\gg$ *OpTwo* by pipe operation is equivalent to :

$a? : A$ $c! : C$
$\exists x : B \bullet (P[a?, x] \wedge Q[x, c!])$

## 2.2 Aspect Orientation in AspectZ

The aspect orientation notion in AspectZ is inspired by the idea of AspectJ [5]. Semantically,  $\Omega$ *SchemaRef* indicates that current aspect crosscuts *SchemaRef* module. Join points in AspectZ are functional identifiers in specification. A *pointcut* is a means of referring to a collection of join points and the common property at those join points; *Advice* is a mechanism used to declare that certain behavior should be performed at each join point in a pointcut. AspectZ supports *insert*, and *replace* advice. The "insert" advice is used to reinforce some restraint on the specification, while the "replace" advice is used to modify some function in the base model.

For example, suppose that there be a base model *ABaseModel* and an aspect model *AnAspectModel* defined as follows.

<i>ABaseModel</i>
$a? : A$ ; $result! : B$ $f : A \rightarrow B$
$result = f(a?)$

<i>AnAspectModel</i>
$\Omega$ <i>ABaseModel</i> $g : B \rightarrow B$ <i>Pointcut PC</i> : $\{f : A \rightarrow B\}$
<i>replace PC</i> : $*' = g \circ *$

Note that there is one join point in *ABaseModel* and *AnAspectModel*. The advice for the pointcut *PC* is to replace each element in *PC* by composing the function *g* with it. According to this advice, the integrated model for

*ABaseModel* + *AnAspectModel*  
is as follows.

<i>AnIntegratedModel</i>
$a? : A$ ; $result! : B$ $f : A \rightarrow B$ $g : B \rightarrow B$
$f' = g \circ f$ $result! = f'(a?)$

Another aspect model is as follows.

<i>AnotherAspectModel</i>
$\Omega$ <i>ABaseModel</i> <i>Pointcut PC</i> : $\{f : A \rightarrow B\}$
<i>insert PC</i> : $\forall x, y \in A \bullet x \neq y \rightarrow *(x) \neq *(y)$

The integrated model for  
*ABaseModel* + *AnAspectModel*  
is

$$\begin{array}{l}
\textit{AnotherIntegratedModel} \\
\hline
a? : A; \textit{result!} : B \\
f : A \rightarrow B \\
\hline
\forall x, y \in A \bullet x \neq y \rightarrow f(x) \neq f(y) \\
\textit{result!} = f(a?)
\end{array}$$

When more than one aspect model has to be woven with a base model, the weaving order is significant. For example, the integrated model for

$A\textit{BaseModel} + A\textit{nAspectModel} + A\textit{notherAspectModel}$  is:

$$\begin{array}{l}
\textit{IntegratedModel3} \\
\hline
a? : A; \textit{result!} : B \\
f : A \rightarrow B \\
g : B \rightarrow B \\
\hline
f' = g \circ f \\
\forall x, y \in A \bullet x \neq y \rightarrow f'(x) \neq f'(y) \\
\textit{result!} = f'(a?)
\end{array}$$

while the integrated model for

$A\textit{BaseModel} + A\textit{notherAspectModel} + A\textit{nAspectModel}$  is:

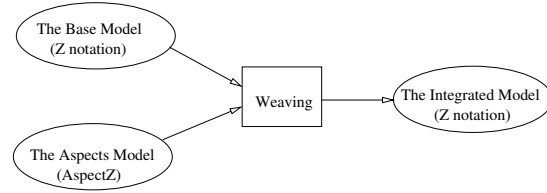
$$\begin{array}{l}
\textit{IntegratedModel4} \\
\hline
a? : A; \textit{result!} : B \\
f : A \rightarrow B \\
g : B \rightarrow B \\
\hline
\forall x, y \in A \bullet x \neq y \rightarrow f(x) \neq f(y) \\
f' = g \circ f \\
\textit{result!} = f'(a?)
\end{array}$$

### 2.3 An Aspect-Oriented Modeling Method

The aspect-oriented modeling framework by AspectZ is illustrated in Figure 1. The basic idea is to provide notations for separately modeling system functionality modules (the base model in pure Z notation) and other crosscutting concerns (the aspect model in AspectZ), and to provide mechanism for systematically composing (or weaving) these models into a complete system model. The aspect-oriented approach to system modeling in AspectZ consists of the following steps:

- (1) Separating aspects from the basic functionality components of the system, and identifying the join points that the functionality components and aspects interact. <sup>1</sup>
- (2) Specifying the base model in Z, and the aspects in AspectZ separately;

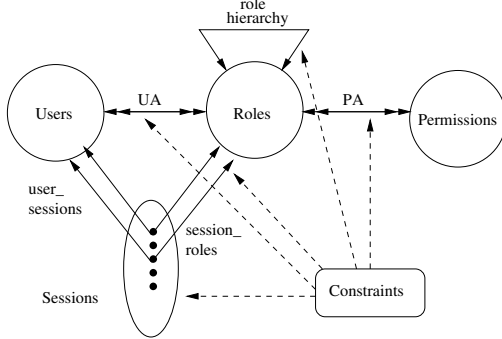
<sup>1</sup>Intuitively, a component can be cleanly encapsulated in a generalized procedure, which tends to be units of the system's functional decomposition, while an aspect cannot be cleanly encapsulated in a generalized procedure, but tends to be properties that affect the performance or semantics of the components in a systemic way [6].



**Figure 1. An aspect-oriented modeling framework**

- (3) For each join point in

- *Dynamic separation of duty (DSD)*, which enforces constraints on the roles that can be activated within or across a user's sessions.



**Figure 2. An RBAC model**

The basic elements in RBAC model are as follows.

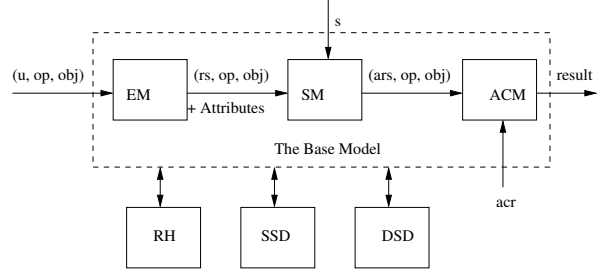
1.  $U, R, P$  and  $S$  (users, roles, permissions and sessions respectively), where  $P$  is the Cartesian product of operation  $OP$  and objects  $OBJ$ ,
2.  $PA \subseteq P \times R$ , a many-to-many permission to role assignment relation,
3.  $UA \subseteq U \times R$ , a many-to-many user to role assignment relation,
4.  $user\_sessions : U \rightarrow \mathbb{P}S$ , a function mapping each user  $u$  to a set of sessions.
5.  $session\_roles : S \rightarrow \mathbb{P}R$ , a function mapping each session  $s$  to a set of roles  $session\_roles(s) \subseteq \{r \mid (user(s), r) \in UA\}$  (which can change with time) and session  $s$  has the permissions  $\bigcup_{r \in session\_roles(s)} \{p \mid (p, r) \in PA\}$ .

### 3.2 Aspect-Oriented Modeling of RBAC

We apply aspect-oriented principle to modeling role-based access control. In the modeling framework, the base modules include *Element Management Module (EM)*, *Session Management Module (SM)*, and *Access Control Module (ACM)*, which define the basic elements, their relationship, and functionality of authorization process, while the aspect modules consist of RH, SSD and DSD, which describe the crosscutting properties among the base modules (see Figure 3).

**Table 1. Variables in Figure 3**

Variable	Description
u	user identity
op	operation
obj	object
s	session
rs	roles
ars	active roles
acr	access control rules



**Figure 3. An authorization architecture**

**The Base Model** Suppose basic types include  $U, R, S, OBJ, OP$ . The schema *Element*<sup>3</sup> defines the data types and functions for extracting content information of queries.

<i>Element</i>
$ua? : \mathbb{P}(U \times R)$
$refer! : \mathbb{P}(R \times U)$
$owner : OBJ \rightarrow U$
...
$refer! = ua?^{-1}$
...

The following Z schemas specify the three base modules in the authorization architecture, and the base model is defined as follows:

$$TheBaseModel = EM \gg SM \gg ACM$$

<i>EM</i>
$\exists Element$
$u? : U; op? : OP; obj? : OBJ; rs? : \mathbb{P}R$
$assigned\_roles : U \rightarrow \mathbb{P}R$
$\forall u : U \bullet assigned\_roles(u) = \{r : R \mid (u, r) \in ua?\}$
$rs! = assigned\_roles(u?)$

<i>SM</i>
$\exists Element$
$op? : OP; obj? : OBJ; s? : S; rs? : \mathbb{P}R$
$ars! : \mathbb{P}R$
$active\_roles : S \rightarrow \mathbb{P}R$
$session\_roles(s?) \subseteq rs$
$\forall s : S \bullet active\_roles(s) = session\_roles(s)$
$ars! = active\_roles(s?)$

<sup>3</sup>In *Element* schema,  $ua$  denotes the user to role assignment relation. For simplicity, irrelevant declarations and specification to the understanding of this paper are omitted.

### ACM

$\exists$ Element  
 $ars? : \mathbb{P}R$ ;  $op? : OP$ ;  $obj? : OBJ$ ;  $result! : BOOL$   
 $acr? : \mathbb{P}(R \times OP \times OBJ)$   
 $matched\_rules : R \times OP \times OBJ$   
 $\rightarrow \mathbb{P}(R \times OP \times OBJ)$

$\forall r : R, op : OP, obj : OBJ \bullet$   
 $matched\_rules(r, op, obj) =$   
 $\{(r, op, obj) \mid (r, op, obj) \in acr?\}$   
 $result! = \exists r \in ars? \bullet$   
 $(r, op?, obj?) \in matched\_rules(r, op?, obj?)$

**The Aspect Models** In the RBAC model, there are three aspects that crosscut the base model by enforcing privilege inheritance between roles and by adding constraints on user-role assignment and/or session-role assignment.

**Aspect 1** A role hierarchy is a pair  $(R, \preceq)$ , where  $R$  is the set of roles and  $\preceq$  is a partial order.  $r_2 \preceq r_1$  if all permissions of  $r_2$  are also permissions of  $r_1$ .

### RH

$\Omega$ TheBaseModel  
 $pi : \mathbb{P}R \rightarrow \mathbb{P}R$   
 $Pointcut PI : \{assigned\_roles : U \rightarrow \mathbb{P}R,$   
 $active\_roles : S \rightarrow \mathbb{P}R\}$   
 $\forall rs : \mathbb{P}R \bullet pi(rs) = \{r_2 : R \mid r_1 \in rs \wedge r_2 \preceq r_1\}$   
 $Replace PI : *' = pi \circ *$

The privilege inheritance function  $pi$  takes a role set as its input and gives an extended role set from role hierarchy as its output. Consequently, permissions of the role set  $rs$  contains also those inherited from  $rs$  through role hierarchy.

**Aspect 2** Static separation of duty [3] is a relation  $ssd \subseteq (\mathbb{P}R \times N)$ , which contains a collection of pairs  $(rs, n)$ , where each  $rs$  is a role set and  $n \geq 2$ , with the property that no user is assigned to  $n$  or more roles from the set  $rs$  in each  $(rs, n) \in ssd$ .

### SSD

$\Omega$ TheBaseModel  
 $ssd? : \mathbb{P}(\mathbb{P}R \times \mathbb{N})$   
 $Pointcut PT_{ssd} : \{assigned\_roles : U \rightarrow \mathbb{P}R\}$   
 $ssd\_checker : \mathbb{P}R \rightarrow BOOL$   
 $\forall x : \mathbb{P}R \bullet ssd\_checker(x) =$   
 $\forall n : \mathbb{N} \mid n \geq 2 \bullet$   
 $(\mid rs \mid \geq n \rightarrow \neg \exists (rs, n) \in ssd? \bullet rs \supseteq x)$   
 $Insert PT_{ssd} : \forall u : U \bullet ssd\_checker(*u)$

The function  $ssd\_checker$  requires its parameter to satisfy the property of static separation of duty.

**Aspect 3** Similarly, aspect of dynamic separation of duty is defined by the following schema, which requires that no subject may activate  $n$  or more roles from the set  $rs$  in each  $(rs, n) \in dsd$ .

### DSD

$\Omega$ TheBaseModel  
 $Pointcut PT_{dsd} : \{active\_roles : S \rightarrow \mathbb{P}R\}$   
 $dsd? : \mathbb{P}(\mathbb{P}R \times \mathbb{N})$   
 $dsd\_checker : \mathbb{P}R \rightarrow BOOL$   
 $\forall x : \mathbb{P}R \bullet dsd\_checker(x) =$   
 $\forall n : \mathbb{N} \mid n \geq 2 \bullet$   
 $(\mid rs \mid \geq n \rightarrow \neg \exists (rs, n) \in dsd? \bullet rs \supseteq x)$   
 $Insert PT_{dsd} : \forall s : S \bullet dsd\_checker(*s)$

**Aspect Weaving** We must be careful about the advice ordering when setting out to aspect weaving, for there are three aspects crosscut the base model. The SSD (or DSD) constraint may exist within role hierarchy relations. When applying the constraint in the presence of a role hierarchy, special care must be taken to ensure that the privilege inheritance do not undermine the SSD (or DSD) constraint. Therefore, a relation aspect has the precedence over a constraint aspect. The following three schemas are the results of weaving all of the above aspects with the base model.

### NewElementManagement

$\exists$ Element  
 $u? : U$ ;  $op? : OP$ ;  $obj? : OBJ$ ;  $rs! : \mathbb{P}R$   
 $ssd? : \mathbb{P}(\mathbb{P}R \times \mathbb{N})$   
 $pi : \mathbb{P}R \rightarrow \mathbb{P}R$   
 $assigned\_roles : U \rightarrow \mathbb{P}R$   
 $ssd\_checker : \mathbb{P}R \rightarrow BOOL$   
 $\forall rs : \mathbb{P}R \bullet pi(rs) = \{r_2 : R \mid r_1 \in rs \wedge r_2 \preceq r_1\}$   
 $assigned\_roles' = pi \circ assigned\_roles$   
 $rs! = assigned\_roles'(u?)$   
 $\forall x : \mathbb{P}R \bullet ssd\_checker(x) =$   
 $\forall n : \mathbb{N} \mid n \geq 2 \bullet$   
 $(\mid x \mid \geq n \rightarrow \neg \exists (rs, n) \in ssd? \bullet rs \supseteq x)$   
 $\forall u : U \bullet ssd\_checker(assigned\_roles'(u))$

### NewSessionManagement

$\exists$ Element  
 $op? : OP$ ;  $obj? : OBJ$ ;  $s? : S$ ;  $rs? : \mathbb{P}R$ ;  $ars! : \mathbb{P}R$   
 $dsd? : \mathbb{P}(\mathbb{P}R \times \mathbb{N})$   
 $pi : \mathbb{P}R \rightarrow \mathbb{P}R$   
 $active\_roles : S \rightarrow \mathbb{P}R$   
 $dsd\_checker : \mathbb{P}R \rightarrow BOOL$   
 $\forall rs : \mathbb{P}R \bullet pi(rs) = \{r_2 : R \mid r_1 \in rs \wedge r_2 \preceq r_1\}$   
 $session\_roles(s?) \subseteq rs?$   
 $active\_roles' = pi \circ active\_roles$   
 $ars! = active\_roles'(s?)$   
 $\forall x : \mathbb{P}R \bullet dsd\_checker(x) =$   
 $\forall n : \mathbb{N} \mid n \geq 2 \bullet$   
 $(\mid x \mid \geq n \rightarrow \neg \exists (rs, n) \in dsd? \bullet rs \supseteq x)$   
 $\forall s : S \bullet dsd\_checker(active\_roles'(s))$

*NewAccessDecision*

$\exists$ Element

$ars? : \mathbb{P}R; op? : OP; obj? : OBJ; result! : BOOL$

$acr? : \mathbb{P}(R \times OP \times OBJ)$

$matched\_rules : R \times OP \times OBJ \rightarrow \mathbb{P}(R \times OP \times OBJ)$

$\forall r : R, op : OP, obj : OBJ \bullet$

$matched\_rules(r, op, obj) =$

$\{(r, op, obj) \mid (r, op, obj) \in acr?\}$

$result! = \exists r \in ars? \bullet$

$(r, op?, obj?) \in matched\_rules(r, op?, obj?)$

The overall access control model *TheOverallModel* is:

*NewElementManagement*

$\gg$ *NewSessionManagement*

$\gg$ *NewAccessDecision*

### 3.3 Analysis of Aspect-Oriented Modeling

One of the major advantage of using a formal language like Z notation is that it is able to reason about the specifications written in it [16]. Take the privilege inheritance property for an example. Let  $auth(r, op, obj)$  represent the result for query  $(r, op, obj)$ . One correctness requirement is role hierarchy loyalty, which can be specified as follows.

$\forall r_1, r_2, op, obj \bullet$

$(r_2 \preceq r_1 \wedge auth(r_2, op, obj) \rightarrow auth(r_1, op, obj))$

*Proof:* Suppose  $r_2 \preceq r_1$ . By the advice in aspect *RH*, we have  $ars_2 \subseteq ars_1$ , where  $ars_1$  and  $ars_2$  are active role sets for  $r_1, r_2$ , respectively. Suppose  $auth(r_2, op, obj)$  is true, then there exists a role  $r$  in  $ars_2$ , such that  $(r, op, obj) \in matched\_rules(r, op, obj)$ . Obviously, the  $r$  is in  $ars_1$  by set inclusion. Therefore,  $auth(r_1, op, obj)$  is true, and this concludes the proof.  $\square$

## 4 Conclusion

This paper has presented a formal aspect-oriented modeling language called AspectZ, and an aspect-oriented modeling method based on AspectZ. AspectZ provides means for observing behaviors of Z schemas and depicting their interrelationships, and provides a systematic method for weaving interrelated schemas. Correctness of aspect weaving can be formally verified by reasoning mechanisms of Z notations.

Initial attempts to establish a formal aspect-oriented modeling method are presented in this paper. Currently, we are investigating the feasibility of the mechanization and automation of the analysis using HOL-Z [8, 15]. HOL-Z provides a conservative extension of HOL with Z. Using support for automated reasoning provided by the Isabelle system, proofs like the above can be carried out at high level and, in some cases, even completely automated. Our future research interests include more complex case studies for aspect-oriented modeling, tool support for aspect-oriented modeling, design and analysis.

## References

- [1] L. Bergmans and M. Aksits. Composing crosscutting concerns using composition filters. *Communications of the ACM*, 44(10):51–57, 2001.
- [2] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley Longman, Inc., 1999.
- [3] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, 2001.
- [4] G. George, R. France, and I. Ray. Design high integrity systems using aspects. In *Proceedings of the 5th IFIP TC-11 WG 11.5 Working Conference on Integrity and Internal Control in Information Systems*, pages 37–57, 2002.
- [5] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold. An overview of AspectJ. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP), LNCS 2072*, pages 327–353. Springer-Verlag, 2001.
- [6] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP), LNCS 1241*, pages 220–242. Springer-Verlag, 1997.
- [7] D. K. Kim, I. Ray, R. B. France, and N. Li. Modeling role-based access control using parameterized UML models. In M. Wermelinger and T. Margaria, editors, *Proceedings of 7th International Conference on Fundamental Approaches to Software Engineering (FASE 2004), LNCS 2984*, pages 180–193. Springer, 2004.
- [8] Kolyang, T. Santen, and B. Wolff. A structure preserving encoding of Z in Isabelle/HOL. In *Proceedings of TPHOLs'96, LNCS 1125*, pages 283–298. Springer-Verlag, 1996.
- [9] K. Lieberherr, D. Orleans, and J. Ovlinger. Aspect-oriented programming with adaptive methods. *Communications of the ACM*, 44(10):39–41, 2001.
- [10] P. Maes. Concepts and experiments in computational reflection. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 147–155. ACM, 1987.
- [11] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.
- [12] A. Rashid, A. Moreira, and J. Araújo. Modularisation and composition of aspectual requirements. In *Proceedings of the 2nd International Conference on Aspect-Oriented Software Development (ASOD 2003)*, pages 11–20. ACM, 2003.
- [13] J. Spivey. *The Z Notation: A Reference Manual (2nd Edition)*. Prentice Hall, UK, 1992.
- [14] J. Stankovic, R. Zhu, R. Poornalingam, C. Lu, Z. Yu, M. Humphrey, and B. Ellis. VEST: an aspect-based composition tool for real-time systems. In *Proc. of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 58–69, 2003.
- [15] P. Steggle and J. Hulance. Z tools survey. Technical report, 1994.
- [16] J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof*. Prentice-Hall, 1995.