Non-functionality analysis and verification based on Aspect-Z

Can Qu 1,a, Haihua Chen 2,b, Xuhui Zhang 3,c, Lichen Zhang 4,d

1 School of Information City College of Huizhou Huizhou, China
2 School of data science Guangzhou Huashang College Guangzhou, China
3 School of Electrical and Computer Engineering Nanfang College Guangzhou Guangzhou, China
4 Lichen Zhang School of computers Guangdong University of Technology Guangzhou, China

a 505609551@qq.com, b alice.hua@qq.com, c 1451607395@qq.com, d lczhanghai@gdut.edu.cn

Abstract. The non-functional requirements of the software system usually cross the whole system, and the object-oriented software development method can lead to the entanglement and dispersion of the code. ASECT-Z proposed a formal method for non-functional requirements, it is the formal specification for the extension of Z language. The language based on set theory and mathematical logic, with strict logic, and the formal specification of reasoning. In this paper, we describe a library management system based on Aspect-Z, and then give a formal proof of the formal specification for formal verification.

Keywords: ASECT-Z; formal verification; aspect oriented; reasoning proof.

1. Introduction
The non-functional requirement of the system is to provide the quality attributes and functional constraints for the software system in order to meet the needs of the users [1]. Aspect oriented programming [2] is a new technology based on the separation of concerns, the system can be separated from different concerns and separate design. Formal methods [3] can help developers find inconsistencies, uncertainties, or incompleteness of other systems that are not readily available. It is an important method to improve the security and reliability of the software system, which combines the aspect-oriented programming technology with the formal specification. Theorem proving is a kind of formal verification technique, which can be inferred from formal specification.

This paper gives a method to prove a theorem of Aspect-Z [4-6] based on formal verification. Formal specification can verify the written instructions, and describe an example of a library management system, based on the formal verification of theorem proving the existence of initialization.
2. non-functional analysis and verification methods

2.1. Aspect oriented formal language Aspect-Z

The statement part of the aspect model can accurately describe the position of the joint point in the system, and the predicate part mainly expresses the function that needs to be realized at the joint point [9-10]. Finally, the component mode and the aspect mode are integrated. The basic pattern of the Aspect-Z language [7-8] is shown as follows fig 1.

```
Spec; Spec
|___ Declaration
|___ Spec; Spec

Declaration ::= BasicDecl; BasicDecl
BasicDecl ::= Ident,…, Ident: Expr
| δSchemaRef
| ΩSchemaRef
| PointcutDecl
SchemaRef ::= SchemaName Decoration[Renaming]
PointcutDecl ::= PointcutIdent : Predica
Spec ::= Predicate | Advice
Advice ::= [insert | replace] PointcutName : Predica
```

Fig.1 Original Aspect-Z Schema

2.2. Formal verification method

After the use of formal specification, we prove the theorem [11] by using the method to verify some properties of specifications to meet our needs and in order to ensure the correctness of the process we describe. If satisfied, this part described is reasonable, otherwise the anti. There are three kinds of formal verification methods, which are the verification of the initial state, the verification of the pre and post conditions and the nature of the inference operation. In this paper, We mainly verify the existence of the initial state as follows.

In Aspect-Z, the initial state is represented by a Init pattern. After completing a formal specification, it is necessary to verify the existence of its initial state. The expression is:

\[
\text{Context} \models \exists x: X \cdot \text{Invariant}(x) \land \text{Init}(x)
\]

Context represents the context environment including global variables, local types, functions, and constant definitions. x: X is the state variable x defined in the class and the corresponding type X. Invariant (x) is the state variables. Init (x) represents the initial variable of the corresponding state variable.

3. Case study

3.1. The system architecture of the library management

This paper selects a small library management system, and carries on the demand analysis to this system. Library management system is transforming the traditional library into digital information library, and realizing automated management of the book. The library management system includes two types of users, the administrator and the reader, and login system based on the login information determines these two types of users. The administrator can add or delete a book to the library; query the book information according to the code of the book; and query the library information of the reader. The reader may
inquire about the book information; and may carry on the borrowed book, returning the book and renewal and so on.

The system has a total of five activators. They are People, Registrar, Reader, Book, and Database. People refers to a person in general who has a relationship with the system; Registrar is for the librarian who is responsible for adding, modifying the book information; Reader is for all readers who may occur the behavior of borrowing, renewal, return book; book contains the title, the object and related attributes of the book. Database is for the storage of various information database object. There are five use cases in the system. AddBook, ModifyBook, Borrow, Renew and Return. AddBook indicates the administrator to add the book information; ModifyBook indicates that can modify the book information; Borrow indicates that the reader can borrow books; Renew indicates the reader to renew the book; Return indicates the reader to return the book. The use case for the system is as follows fig2.

In addition to meeting the functional requirements described above, the non-functional definition is to log the system logs and prompt them against the corresponding results. Administrators add or delete the corresponding books, and updates to the database at the same time, then the system can call the appropriate aspects of this information that will be recorded to the system log. If the record has been successful, the prompt is successful, otherwise the contrary. Adding a book and calling the corresponding the structure of aspects module are shown in fig.3.
3.2. Non-functional requirements based on Aspect-Z

In the library management system, the non-functional requirements are as follows: When the book information changes, such as book lending or returning, adding books or deleting books, information will be updated to the database, and will be recorded in the system log.

According to the library management system requirements, Users, Book and Database will be created. Book contains only declarative model to describe the book's properties, while Users and Database combine with getting a complete system state. The internal variables of book mode include three attributes, title, author and subjects, and the same book contains multiple copies; the operation of the book is a copy of the operation. USERS mode is divided into managers and readers. Database mode includes all the copies information in the stack, the current copy information, the loan situation of each copy, the last lend record of each copy, the current borrowed situation of reader. As shown in Figure 4, Figure 5, Figure 6, Figure 7 below.

The formal description of system log record is as follows fig7.

![Fig.4 Z schema for the Book](image)

![Fig.5 Z schema for the Users](image)

![Fig.6 Z schema for the Database](image)

![Fig.7 Log Class Model](image)
The complete initialized library management system is as follows fig 8.

**Fig.8 Z schema for the Initialization of Library management system**

The administrator adds a new copy of the book, including the new copy attribute, including the title, author, and subject, as follows is fig 9.

**Fig.9 Z schema for the AddBook**

In the system, the system log of non-functional requirements is established aspects. The system log records every detail of the system operation and records the hardware, software and system information, which plays a vital role in the stable operation of the system. As follows is fig 10.

**Fig.10 Aspect-Z aspect-schema for Log**

In the operation of the system, in order to determine the implementation of the operation or not, the operation should be successful or failure to give tips, this information can also be prompted to establish aspects, the model is as follows fig 11.
This example is the log aspect and the operation of information feedback to cross-cut, which Composition is the cross-cutting rules to illustrate the aspects knitting at the joint point. The priority of LogAspect aspects is higher than AspectMessage aspects to feedback the success or not of the log. The priority of the definition is shown in figure 12.

The aspect module is woven into the newly added book integration module Integerated is shown in figure 13.

3.3. Verification of the existence of initial state
In this example, the context of the initial state is:

\[
\text{Context} = \left[ \text{Users} \ni = \text{Registrar} \land \text{Reader}, \text{Book} \ni \text{Title} \cup \text{Author} \cup \text{Subject}; \text{Registrar} \neq \emptyset; \text{Reader} \neq \emptyset; \text{Stock} = \emptyset \mid 0 \text{maxbooks}; \text{Available} = \emptyset \mid 0 \text{maxbooks}; \text{Checked_out} = \emptyset \mid 0 \text{maxbooks}; \text{Available} \cup \text{Checked_out} = \text{maxbooks}; \right]
\]

Therefore, it can be concluded as follows.

\[
\text{Context} \ni \exists \text{Registrar}, \text{Reader: Users;} \text{Title, Author, Subject: Book; stock: Stock; available: Available; checked_out: Checked_out \times registrar \neq \emptyset \land reader \neq \emptyset \land \text{stock} = \text{maxbooks} \land \text{available} = \text{maxbooks} \land \text{checked_out} = \emptyset \land \text{book_info} = \emptyset
\]

Use of point-rules can be simplified:

\[
\text{Context} \ni \exists \text{Registrar} \neq \emptyset \land \text{Reader} \neq \emptyset \land \text{maxbooks} \in \text{Stock} \land \text{maxbooks} \in \text{Available} \land \text{maxbooks} \in \text{Checked out} \land \emptyset \in \text{book_info}
\]
The following results can be obtained from the state variable and the given type.
Registrar≠∅, Reader≠∅, maxbooks ∈ Stock, maxbooks ∈ Available, ∅ ∈ Checked_out, ∅ ∈ book_info
So, we can prove Context ├ true, the initial state of existence.

4. Conclusion
The core idea of aspect oriented is to abstract the non-functional features of the system into aspects, and the modular non-functional requirements. It is a good solution to the problem of crosscutting concerns, which makes up for the lack of object-oriented programming. Formal methods can help to find other methods is not easy to find that the description of inconsistent and ambiguous or incomplete, and by inference verification to ensure the final product meets these requirements specification software. Formal methods can be used to verify the program and ensure the correctness of the program. Therefore, the formal verification can be used to check the errors in the development process. In this paper, a formal verification method based on Abject-Z theorem proving is given, we can check the existence of the initial state, the pre and post conditions and class attributes. This paper describes an example of a library management system based on Abject-Z, and on this basis, it gives the formal verification of its theorem proving.

Acknowledgments
The paper is supported by funds from Department of Science and Technology at Guangdong Province and Guangzhou City (2015A030313490).

References
[1] Keshav Bharadwaj A, Gopalakrishnan Nair T R. Mapping General System Characteristics to Non-Functional Requirements [C] // Proceedings of the 2009 IEEE International Advance Computing Conference, Patiala India, 2009 : 1634 - 1638.
[2] YUAN SH J, XU SH P, DU S P, et al. A Method Based Software Requirements Specification Description Quality Evaluation[J]. Electronic Design Engineering, 2013, 21(7):11-13.
[3] YAO Q Z, WANG J. Software Formalization Requirements Analysis And Verification Based OnUml[J]. Computer Engineering, 2010, 36(13):30-33.
[4] Lan H Y. Aspect-Oriented Programming[J]. ACM SIGSOFT Software Engineering Notes, 2001, 26(5):468.
[5] Rashid A, Sawyer P, Moreira A, et al. Early Aspects: A Model for Aspect-Oriented Requirements Engineering[C]// 2013 21st IEEE International Requirements Engineering Conference (RE). IEEE Computer Society, 2010:199.
[6] LIU J Y, ZHONG Y, ZHANG L. The Aspect-Oriented Modelling of Software Non-Functional Requirements[J]. Computer Application and software, 2010, 27(12):40-41.
[7] Zhang L. Aspect-Oriented Formal Techniques of Cyber Physical Systems[J]. Journal of Software, 2012, 7(4).
[8] Silva C V, Saens R, Del Rio C, et al. Aspect-Oriented Modeling: Applying Aspect-Oriented UML Use Cases And Extending Aspect-Z[J]. Computing & Informatics, 2013, 32(3):10
[9] Fabry J, Kellens A, Denier S, et al. AspectMaps: Extending Moose To Visualize AOP Software[J]. Science of Computer Programming, 2014, 79(1):6-22.
[10] Vidal C, Saens R, Rio C D, et al. Aspect-Oriented Modeling: Applying Aspect-Oriented UML Use Cases And Extending Aspect-Z[J]. Computing & Informatics, 2013, 32(3):1001-1021.
[11] WEN Z C, LIAO H K, ZHANG X L. Formal Verification Based on Object-Z Specification[J]. Computer science, 2007, 34(5):247-251.