2011 International Conference on Computer and Software Modeling
IPCSIT vol.14 (2011) © (2011) IACSIT Press, Singapore

Conceptual Framework of Transforming Informal Requirements Specification to Z Specification

Phanarut Srichetta

64 Thaharn Rd., Muang, Udon Thani Rajabhat University, Udon Thani, 41000 Thailand

Abstract. The effectiveness of a system is related to the correct understanding of the needs of the system's customer. Natural language is used to produce a requirements specification of a system from customer's requirements, but it has the inherent ambiguity that can lead to misunderstanding. Formal specification is a reliable requirements specification expressed in a language whose vocabulary, syntax, and semantics are formally defined, and has a mathematical basis. This paper aims to propose the conceptual framework of transforming the requirements specification of a system to formal specification. It focuses on the Z specification language which is a formal specification based on set theory and first order predicate logic. Three key concepts of the formal specification including data invariant, state, and operation are applied for specifying the requirements specification based on the restrictive and simple sentence statements. These statements are transformed into the corresponding parts of Z specification. This framework can help the software developers in specifying the system requirements formally.

Keywords: Requirements specification, Formal specification, Formal methods, Z specification

1. Introduction

In software development process, the main aim is to trap errors as early as possible. At the first phase this process, natural languages are often used to describe system requirements and produce a requirements specification. Although they aid users in understanding the system, they have inherent ambiguities that can lead to misunderstanding. Ince [1] stated that there are a number of major problems afflict software projects. Many arise from the unsatisfactory nature of the notations used to describe a software product as it progresses through its life cycle. Moreover, because of the increasing complexity in scale and functionality of software systems, the likelihood of subtle errors is much greater. In order to enable developers to construct systems that operate reliably despite this complexity, the formal methods should be used.

Formal methods [2, 3] are mathematically based languages, techniques, and tools for specifying and verifying the software systems. They can be used at a number of levels where a formal specification [4] is the first level that focused in this research. The formal specification is a specification expressed in a language whose vocabulary syntax and semantics are formally defined, and has a mathematical, usually formal logic, basis. It has various languages based on two approaches: algebraic and model-based. An algebraic approach describes system in terms of operations and their relationship. OBJ [5] and Larch [6] are examples of algebraic specification language. In a model-based approach, a model of the system is constructed using well-understood mathematical entities such as sets and sequences. The most popular languages are VDM [7] and Z [8]. In practice, model-based methods are more widely used than algebraic approach because the specification languages of the model-based ones provide a rich variety of mathematical constructs which make the specification more concise.

In this paper, the Z specification language, a formal specification based on set theory and first order predicate logic, is focused. Examples of systems that use Z specification to specify specification of the system are air traffic control [9] and financial systems [10]. This paper proposes a conceptual framework for transforming the informal requirements specification to Z specification. Three key concepts of the formal

specification including data invariant, state, and operation with pre-conditions and post-conditions are applied for specifying the requirements specification based on the restrictive and simple sentence statements. They are transformed into the corresponding parts of Z specification. This framework can help the software developers in specifying the system requirements formally.

2. Formal Specification

In order to generate the formal specification, the process of requirements engineering for formal specification is presented as shown in Fig. 1.

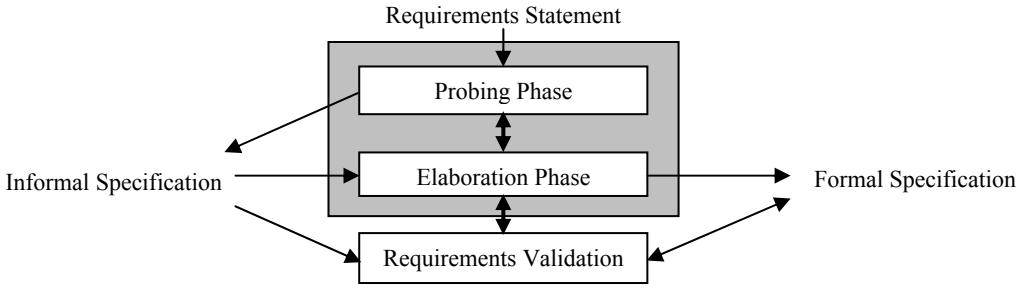


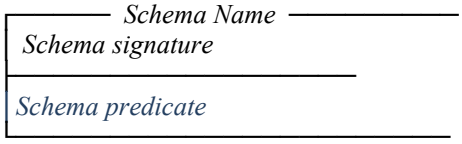
Fig. 1: The Formal Specification Process Model.

In the Probing phase, the specifier will impose a structure on the problem. Informal specification techniques both structured analysis [11] and object-oriented analysis [12] are well-suited to this phase. In the Elaboration phase, ambiguities, inconsistencies and incomplete requirements have to be identified. Attempts at resolving these requirements may lead to the modifications of the structure initially imposed on the problem. Problem analysis in this model is thus an iteration of probing and elaboration phase activities. The Requirements Validation is concerned with establishing the validity of the requirements specification. For the formal specification, there are three key concepts the formal specification has [13].

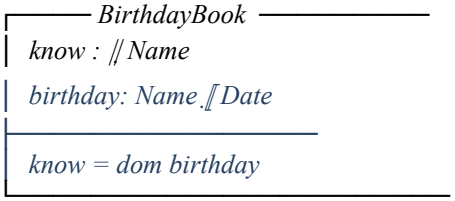
- *Data invariant:* A condition that is true throughout the execution of the system that contains a collection of data.
- *State:* The stored data which a system accesses and alters.
- *Operation:* An action that takes place in a system and read or writes data to a state. An operation is associated with two conditions: a pre-condition and a post-condition.

3. The Specification Language Z

The specification of an application using Z [8] is constructed basically by the definition of schemas as shown in Fig. 2.a. The schemas are used to describe both static and dynamic aspects of a system. A Z schema consists of a name, the signature part that specifies the attributes and the respective types of the entities being specified, and of a predicate constraining the possible values that the attributes can take. It is used to specify the state and operations of a system. Fig. 2.b illustrates the example of the Z schema, that is, the BirthdayBook schema. Z specification usually consists of six main parts: the declaration of types and global variables, the state space, the initial state, the operations, the error handling schemas, and the robust versions of operations. Each of them is formed in terms of the schema.



(a) The Structure of Z Schema



(b) The *BirthdayBook* Schema

Fig. 2: A Z Schema.

The declaration of types and global variable consists of basic types, free types, and axiomatic description. The state space presents the state variables and the relationships between the state variables. The initial state assigns initial values to the state variables. The operation schemas can either change the values of the state variables or preserve them. Their predicate part contains the pre-condition and post-condition. The error handling schemas specify what happens if the pre-condition in operation schema does not hold. Lastly, the robust versions of operations define the complete operations in the sense that they are able to handle errors which might occur when invalid date is input.

4. Conceptual Framework

The proposed conceptual framework consists of three modules: information specification module, transformation module and formal specification module, as illustrated in Fig 3. The requirements specification of a particular system is the input of the informal specification module. The data storage unit is used to keep the information obtained from the informal specification module which will be used by the transformation module. The transformation module transforms information of informal requirements specification kept in the data storage unit into the corresponding parts of formal specification. The result of this framework is the Z specification.

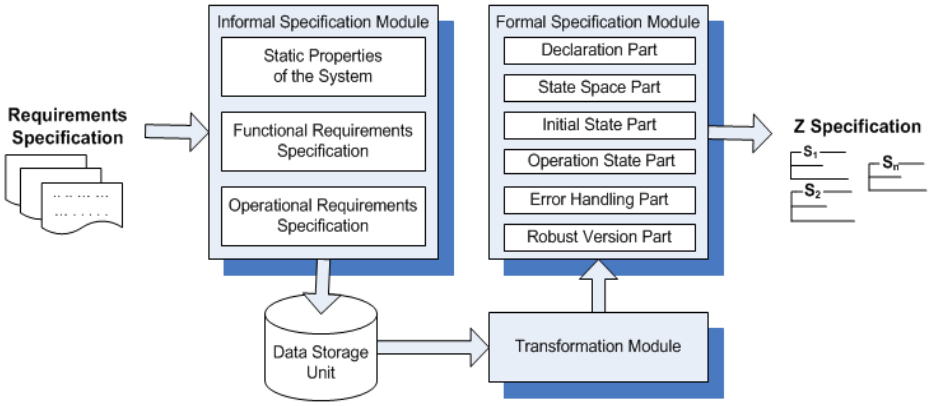


Fig. 3: A proposed conceptual framework.

4.1. Informal Specification Module

In this module, the requirements specification to be entered will be divided into three sections, that are, the static properties of the system, functional requirements specification, and operational requirements specification. The pattern of entering is represented in terms of restrictive and simple sentence statements (subject-verb-object) of natural language. The verb part is defined for meaning specifically.

1.) The Static Properties of the System. This section presents the properties of the system which have a number of objects. The main objects of the system have to be specified firstly. Each main object also has category, invariant properties and relationships. It divides into four parts: objects, category objects, relations, and relationship between objects.

Objects part relates to the main objects in the system. Each object can be divided into three parts: details, categories, and invariant properties, as shown in Table 1.

Table 1: Object properties template.

	Subject	Verb	Object
Object's Details	Object name	consists of unique / consists of a unique / consist of unique / consist of a unique / has unique / has a unique / have unique / have a unique /consists of /consist of / has a /have a	Object properties
Object's Categories	Object name	can be / can also be / can either be…or…	
Object's Invariant Properties	Object name	can only be / can either be …or…/ can not be / cannot be / can't be/ can not simultaneously be / cannot simultaneously be / can't simultaneously be / can not be both / cannot be both / can't be both / can not be a combination of / cannot be a combination of / can't be a combination of	Object details
	Each+Object name	has a /have a	Category object

Category Objects part relates to the invariant properties of each category of the main object in the previous objects part. This part uses simple sentence the same as objects part as shown in Table 2.

Relations part is related to the relation between set that can be the object or category object. This part divides into two sections: relation details and relation properties, as shown in Table 3. The fact about details or properties of relation must be specified in natural language. In deep details of the relation, what relation or function and what set relate to what set must be defined.

Relationship between Objects part is related to the invariant properties between the different objects that specify in the Objects part. This part uses the same template as Objects part but different particulars as shown in Table 4.

Table 2: Category object properties template.

Subject	Verb	Object
-A/An/Each + Object name	is a subset of / is a subset of…but not equal to / is a proper subset of	Category object relate with category object in this Subject part
-All + Object name	are subsets of /are subsets of…but not equal to / are proper subsets of	
-Every member of + Object name	is a member of / is a member of…but not equal to	
-All the members of (All above + Category object) + Object name	are members of / are members of…but not equal to	

Table 3: Relation details template.

Set 1	Relation / Function	Set 2
Object / Category Object	Relation / Partial function / Total function / Partial Injection / Total Injection / Partial Surjection / Total Surjection / Bijection	Object / Category Object

Table 4: Relationship between objects template.

Subject	Verb	Object
Category object with their quantitative such as A / An / Each/ All / Every member of / All the members of	-"is (V) by/are (V) by" -"is associated with"	Category object with its tip such as "all of / equal to / more than / less than / more than and equal to / less than and equal to / no more than / no less than".

2.) Functional Requirements Specification. In this section, the functions' details of the system are specified. The function's details consist of the function's name, description, type, and related objects or relations. The function's name and description can be specified freely. The function's type is limited to add, remove, and change data. The related objects or relations can be selected from the previous specify.

3.) **Operational Requirements Specification.** This section divides the operations for each function into two parts that are pre-condition and post-condition. These operations explain the given function in details.

Pre-condition part is about the predicates that are constraints about the values of global variables and parameters before the operation is executed. This part uses natural language as input template that can be divided into three parts of parameter existent checking, and one of the other parts.

- Check existing parameter (*must be / must exist in*)
- Check non-exist parameter (*must not be / must not exist in*)
- Check when will error occur (*Error can occur when*)

Post-condition part is about the predicates that are constraints about the values of global variables and parameters after the operation has been executed. This part uses natural language as input template that can be divided into three parts of checking the effect of parameter to the operation, and one of the other parts.

- Check placed in (is added to / is placed in)
- Check remove from (is removed from / is deleted from / is extracted from)
- Check no effect (is no effect to)

4.2. Formal Specification Module

This module involves generating Z specification. Therefore, the structure of the specification document consists of six parts: Declaration part (basic type, free type, axiomatic description), State Space part (state variable, invariant), Initial State part (initial value), Operation State Part (parameter, pre-condition, post-condition), Error Handling part (parameter, pre-condition, error message be post-condition), and Robust Version part (schema specifying operations and error-handling schemas using logic schema operations).

4.3. Transformation Module

The transformation module obtains the informal requirements specification of the system kept in the data storage unit and then transforms it into the corresponding parts of Z specification. It is noted that the tokenizing, stemming the stop words, and generating the specific key terms have to be performed in this module. The main objects specified in the static properties of the system will be transformed to Z specification as the scheme name in the *state space part*. Their details and categories, and invariant properties will be transformed as the scheme signature and scheme predicate respectively in terms of the set properties in mathematics. Fig 4 shows an example of the comparison of the informal and formal part in which (a) the static properties of object *Book* of the library management system can be transformed into (b) the declaration part and (c) the state space part. In this case, there is no value in free type and axiomatic description.

Object: Book
Object details and categories:
Book *consists of unique* book number.
Book *consists of* title, author, and isbn number.
Book *can be* borrowed, missing, or on the shelves.
Invariant properties:
Book *can only be* missing, borrowed or on shelves.
Book *can not simultaneously be* missing and on shelves.

(a) The static properties of the object Book [Informal]

Basic Type: [*BookNumbers*]

(b) The declaration part [Formal]

Books

$AllBooks : \mathbb{F}\ BookNumbers$
$Hastitle : BookNumbers \rightarrow Titles$
$HasAuthor : BookNumbers \rightarrow Authors$
$HasIsbnNumber : BookNumbers \rightarrow IsbnNumbers$
$BorrowedBooks : \mathbb{F}\ BookNumbers$
$MissingBooks : \mathbb{F}\ BookNumbers$
$OnShelveBooks : \mathbb{F}\ BookNumbers$

$BorrowedBooks \cup MissingBooks \cup OnShelveBooks = AllBooks$
$MissingBooks \cap OnShelveBooks = \varnothing$

(c) The state space part [Formal]

Fig. 4: A Comparison of Informal Part and Formal Part after Transformation.

The *initial state part* is generated if the values of objects' properties are assigned in the operational requirements specification. The scheme of *operation state part* is generated based on the type of functions and details of conditions in the operational requirements specification. The *error handing part* can be generated if there exists the exceptions either in pre-conditions or post-conditions are specified. Lastly, the *robust version part* presents that the robust state occurred in case of which operation state part and error handing part by using the logic operations such as $\wedge$ or $\vee$.

5. Conclusion

The proposed conceptual framework in this paper illustrates the way to transform informal requirements specification to formal requirements specification based on Z language. Using the restrictive and simple sentence statements and the concepts about data invariant, state, and operation in specifying informal requirements specification can state the strict system requirements. The generated Z specification shows the formal pattern of requirements specification, therefore, the ambiguity of the system requirements is cut. The computer oriented implementation of proposed conceptual framework could be performed in the future.

6. References

[1] D. C. Ince. An Introduction to Discrete Mathematics, Formal System Specification and Z. 2nd Ed.Walton Street, Oxford : Oxford University Press, (1992).

[2] A. Hall. Realising the Benefits of Formal Methods. *Formal Methods and Software Engineering.* K.Lau and R. Banach (eds.), LNCS 3785 (2005), pp. 1-4.

[3] J. M. Rushby. Automated Formal Methods Enter the Mainstream. *J. UCS*, Vol. 13, No. 5 (2007), pp. 650-660.

[4] A. V. Lamsweerde. Formal Specification: a Roadmap. *Proceedings of the Conference on the Future of Software Engineering (2000)*, pp. 147-159.

[5] K. Futatsugi, J. A. Goguen, J. P. Jouannaud and J. Meseguer: Principles of OBJ2. In: *Proceedings of the ACM Principles of Programming Languages Conference*. (1985), pp. 168-172.

[6] J. V. Guttag, J. J. Horning, and J. M. Wing. The Larch Family of Specification Languages. *IEEE Software.* Vol. 2, No. 2, (1985), pp. 24-36.

[7] C. B. Jones. *Systematic Software Development Using VDM*. Prentice-Hall, (1986).

[8] J. M. Spivey.*The Z Notation: A Reference Manual*. 2nd Ed., New York, Prentice-Hall, (1998).

[9] A. Hall. Using Formal Methods to Develop an ATC Information System. *IEEE Software*. (1996), pp. 66-76.

[10] A. Hall and R. Chapman. Correctness by Construction: Developing a Commercial Secure System. *IEEE Software*. (2002), pp. 18–25.

[11] E. Yourdon. *Modern Systems Analysis*. Prentice-Hall, (1989).

[12] S. Shlaer and S. J. Mellor. *Object lifecycle: Modeling in world in states*. Prentice-Hall, (1992).

[13] R. S. Pressman. *Software Engineering: A Practitioner's Approach*. 4th Ed. New York : McGraw-Hill, (1997).