Z Formal Specification Language - An Overview

Mr. Vishal Ruhela
Graduate Trainee Engineer, HCL Technologies, Noida, India

Abstract

Formal methods provide a much needed solid software engineering foundation for the 'art' of programming computers. Formal specifications can be used to provide an unambiguous and consistent supplement to natural language descriptions and can be rigorously validated and verified leading to the early detection of specification errors. Z is a model oriented formal specification language based on Zermelo-Fränkel axiomatic set theory and first order predicate logic. It is a mathematical specification language, with the help of which natural language requirements can be converted into mathematical form. In this paper an overview of formal method is presented. Z formal specification language is described using small example.

Keywords: *Informal and Formal Specification Language, Model Oriented , Z.*

1. Introduction

With the ever-increasing complexity of computer systems, reliable and effective, design and development of high quality systems that satisfy their requirements is extremely important. In the mission and safety critical system failure can cause cost overrun, loss of lives or even severe economic consequences can arise. So, in such situations, it is necessary that errors are uncovered before software is put into operation. These challenges call for acceptance of proper engineering methods and tools and have motivated the use of formal methods in software engineering.

There are varieties of formal specification languages available to fulfill this goal and one way to achieve this goal is by using Z formal specification language. Z is model oriented formal method based on set theory and first order predicate calculus [1].

In this paper an outline of formal method is described in Section 2.In Section 3 Difference between formal and informal specification language is presented. In Section 4 An overview of Z formal specification language is described. In Section 5 An example to describe Z is presented. Conclusions are presented in Section 6.

2. An Outline

In this section we describe formal method, formal specification language and its different types.

2.1. Formal Method

Formal methods used in developing computer systems are mathematical techniques for portraying system properties. Such formal methods provide structures within which software system can be specified, developed and verified in a systematic, rather than ad hoc, manner [2]. Formal methods can be applied throughout the development of a system to precisely describe a system and involve the use of refinement techniques and proof obligation at each stage to ensure the correctness, completeness and consistency of specification. Formal methods used in developing computer systems are mathematical based techniques for portraying system properties.

Formal methods can be used at a number of levels:

Formal Specification: In computer science, a formal specification is a mathematical description of software or hardware that may be used to develop an implementation. It describes what the system should do, not (necessarily) how the system should do it. Given such a specification, it is possible to use formal verification techniques to demonstrate that a candidate system design is correct with respect to the specification. This process of formal specification is similar to the process of converting a word problem into algebraic notation.

Formal development and verification: Formal development process involves iteratively refining a formal specification to produce the finished system. Formal Methods differ from other specification systems by their heavy emphasis on provability and correctness. By building a system using a formal specification, the designer is actually developing a set of theorems about his system. By proving these theorems correct, the formal methods ensures the correctness of the system. The process of proving or disproving properties of the software system against a formal specification is known as formal verification.

Implementation: Once the model has been specified and verified, it is implemented by converting the specification into code. As the difference between software and hardware design grows narrower, formal methods for developing embedded systems have been developed [2].

2.2. Formal Specification Language

The representation used in formal methods is called a formal specification language. The language is formal in the sense that it has a formal semantics and as a result can be used to express specifications in a clear and unambiguous manner. A formal specification language can be used to specify the task at hand in a clear and concise manner. As formal methods and formal specification language has sound mathematical basis, it provides the means of proving that specification is realizable, complete, consistent and unambiguous. Even the most complex systems can be modeled using relatively simple mathematical objects, such as sets, relations and functions [2].

A formal specification language is usually composed of three primary components or in mathematical term we can say that it consists of two sets, syntax and semantics and a set of relation [2].

The specific notation with which specification is represented is defined by syntactic domain or syntax. Formal techniques can have considerably different semantic domain. Semantics helps to define a universe of objects that will be used to describe the system. Set of relations defines the rules that indicate which objects properly satisfy the specification. Formal specification languages use mathematics as their basis. Most complex systems can be modeled using simple mathematical objects, such as sets, relations and functions. A mathematical statement is unambiguous and precise, which provides a way to give convincing arguments to justify ones solutions, and allows proving that an implementation satisfies the mathematical specification [2].

2.3. Types of Formal Specification Languages

Different types of Formal Specification Languages are:

2.3.1. Model Based Languages

There are a number of different ways to write a precise specification. One approach is model based languages. In it the specification is expressed as a system state model. This state model is constructed using well understood mathematical entities such as sets, relations, sequences and functions. Operations of a system are specified by defining how they affect the state of the system model. Operations are also described by the predicates given in terms of pre and post conditions [3].

The most widely used notations for developing model based languages are Vienna Development Method (VDM) [4], Zed (Z) [1] and B [5].

2.3.2. Algebraic Specification Languages

Process algebras are amenable to algebraic manipulation; however, there are also languages which describe a system solely in terms of its algebraic properties. These algebraic specification languages describe the behavior of a system in terms of axioms that characterize its desired properties. Examples of algebraic specification languages include OBJ [7] and the Common Algebraic Specification Language (CASL) [6]. In mathematical terms algebra (or an algebraic system) consists of (1) a set of symbols denoting values of some type, referred to as the carrier set of the algebra; and (2) a set of operations on the carrier set.

2.3.3. Process oriented Languages

Concurrent systems are described using process oriented formal specification language. A specific implicit model for concurrency is the basis for these languages. In these languages processes are denoted and built up by expressions and elementary expressions, respectively, which describe particularly simple processes. Ex. Communicating Sequential Processes (CSP) [8].

2.3.4. Hybrid Languages

Many systems are built with a combination of analog and digital components. In order to specify and verify such systems it is necessary to use a specification language that encompasses both discrete and continuous mathematics. There has been recent interest in these hybrid languages, such as CHARON [9]. A simple example of a nonlinear hybrid system is that of a temperature controller. The temperature of a room is controlled through a thermostat which continuously senses the temperature and turns the heater on and off.

3. Difference between Informal and Formal Specification language

Requirements specification languages may be classified into two types: formal specification languages and informal specification languages. Informal specification language uses natural language like English for specifying requirements. But they tend to include various deficiencies such as a system specification can contain contradictions, ambiguities, vagueness, and incomplete statements.

Contradictions: Sets of statements that are at variance with each other. For example, one part of a system specification may state that the system must monitor all the temperatures in a chemical reactor while another part, perhaps written by another member of staff, may state that only temperatures occurring within a certain range are to be monitored. Normally, contradictions that occur on the same page of a system specification can be detected easily. However, contradictions are often separated by a large number of pages.

Ambiguities: Statements that can be interpreted in a number of ways. For example, the following statement is ambiguous: The operator identity consists of the operator name and password; the password consists of six digits. It should be displayed on the security VDU and deposited in the login file when an operator logs into the system. In this extract, does the word it refers to the password or the operator identity?

Vagueness often occurs because a system specification is a very bulky document. Achieving a high level of precision consistently is an almost impossible task. It can lead to statements such as the interface to the system used by radar operators should be user-friendly or the virtual interface shall be based on simple overall concepts that are straightforward to understand and use and few in number. A casual perusal of these statements might not detect the underlying lack of any useful information.

Incompleteness: The most frequently occurring problems with system specifications. For example, consider the functional requirement: The system should maintain the hourly level of the reservoir from depth sensors situated in the reservoir. These values should be stored for the past six months. This describes the main data storage part of a system. If one of the commands for the system was: The function of the AVERAGE command is to display on a PC the average water level for a particular sensor between two times. Assuming that no more detail was presented for this command, the details of the command would be seriously incomplete. For example, the description of the command does not include what should happen if a user of a system specifies a time that was more than six months before the current hour.

On the other hand Formal specification languages have a mathematical (usually formal logic) basis and employ a formal notation to model system requirements. The desired properties of a formal specification consistency, completeness, and lack of ambiguity are the objectives of all specification methods. However, the use of formal methods results in a much higher likelihood of achieving these ideals. The formal syntax of a specification language enables requirements or design to be interpreted in only one way, eliminating ambiguity that often occurs when a natural language (e.g., English) or a graphical notation must be interpreted by a reader. The descriptive facilities of set theory and logic notation (enable clear statement of facts (requirements). To be consistent, facts stated in one place in a specification should not be contradicted in another place. Consistency is ensured by mathematically proving that initial

facts can be formally mapped (using inference rules) into later statements within the specification. Completeness is difficult to achieve, even when formal methods are used. Some aspects of a system may be left undefined as the specification is being created; other characteristics may be purposely omitted to allow designers some freedom in choosing an implementation approach; and finally, it is impossible to consider every operational scenario in a large, complex system. Things may simply be omitted by mistake [2].

4. Overview of Z Formal Specification Language

In this section, we briefly describe Z formal specification language. A Z specification for employee maintenance is presented to illustrate the use of language. We then describe how the specification can be strengthened.

4.1. Description of Z Formal Specification Language

The Z language is a model oriented, formal specification language that was proposed by Jean-Raymond Abrail, Steve Schuman and Betrand Meyer in 1977 and it was later further developed at the programming research group at Oxford University [10]. It is based on Zermelo Fränkel axiomatic set theory and first order predicate logic. The Z notation [1], [11] is a strongly typed, mathematical, specification language. It has robust commercially available tool support for checking Z texts for syntax and type errors in much the same way that a compiler checks code in an executable programming language. It cannot be executed, interpreted or compiled into a running program. It allows specification to be decomposed into small pieces called schemas. The schema is the main feature that distinguishes Z from other formal notations. In Z, both static and dynamic aspects of a system can be described using schemas. The Z specification describes the data model, system state and operations of the system. Z specification is useful for those who find the requirements, those who implement programs to meet those requirements, those who test the consequences, and those who write instruction manuals for the system [1].

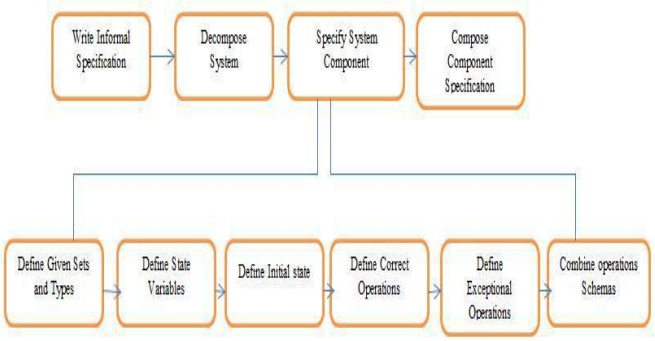

Figure 2.1: Z Process

Z also helps in refinement towards an implementation by mathematically relating the abstract and concrete states. Z is being used by a wide variety of companies for many different applications.

In the Z notation there are two languages [1]:

Mathematical Language

The mathematical language is used to describe various aspects of a design: objects and the relationships between them by using propositional logic, predicate logic, sets, relation and functions.

Schema Language

The schema language is used to structure and compose descriptions: collecting pieces of information, encapsulating them, and naming them for reuse.

4.2. Structure for Z Specification

Schemas are box like structure that introduces variables and specifies the relationship between these variables [1]. A schema is shown below. All declarations are made above the central line and predicates are defined below the central line.

SchemaName

Declarations
Predicates

DECLARATION: The declarations part of the schema will contain:

- a list of variable declarations; and
- references to other schemas (this is called schema inclusion).

PREDICATES: Values of variables are constrained below the central line. The predicate part of a schema contains:

- a list of predicates, separated either by semi-colons or new lines.

The declarations part is separated from the predicate part by the horizontal line.

4.3. Z Conventions [1]

- If any variable name, N, is followed by ' e.g. N', this means that it represents the value of the state variable N after the operation. In Z terminology, N is decorated with a dash.
- If a schema name is decorated with , this introduces the dashed values of all names defined in the specification together with the invariant applying to these values.
- If a variable name is decorated with !, this means that it is an output e.g. message!.
- If a variable is decorated with ?, this means that it is an input e.g. amount?.
- If a schema name is prefixed with the Greek character Xi (X), this means that dashed versions of the variables defined in the named schema are introduced. For all variable names introduced in the schema, the values of corresponding dashed names are the same. That is, the values of state variables are not changed by the operation.
- If a schema name is prefixed with the Greek character Delta (D), this implies that values of one or more state variables will be changed by the operation where that schema is introduced. For all variable names introduced in the named schema, corresponding dashed names are also introduced and may be referenced in operations.

4.4. Benefits and Limitations of Z

4.4.1. Benefits of Z [1]:

- A Z specification forces the software developer to completely analyze the problem domain. (e.g. identify the state space and pre and post conditions for all operations).
- A Z specification forces all major design decisions to be made prior to coding the implementation. Coding should not commence until you are certain about what you should be coding.
- A Z specification is a valuable tool for generating test data, and the conformance testing of completed systems.
- A Z specification allows formal exploration of properties of system.
- The flexibility to model a specification which can directly lead to the code.
- A large class of structural models can be described in Z without higher – order features, and can thus be analyzed efficiently.
- Independent Conditions can be easily added later.

4.4.2. Limitations of Z [12]:

- Z does not provide any support for concurrency.
- It does not provide any concept for timing aspects.
- Sequencing operations is difficult with Z.
- Explicit representation of Non-determinism (how to represent un-determined or unknown parameters ?!)
- No single approach has yet asserted itself as the best starting point for defining reasoning about real-time behavior in Z.

5. An Example

A specification to convert requirements written in natural language to Z formal

specification language method is given below. The specification depicts small operation to add, students details such as rollno, name, class, section, address into school database.

5.1. Specification

[*ROLLNO,NAME,CLASS,SECTION,ADDRESS*]

[*ROLLNO,NAME,CLASS,SECTION,ADDRESS*]

StudentsDetails

$rollno : \mathbb{P}\ ROLLNO$

$name : \mathbb{P}\ NAME$

$class : \mathbb{P}\ CLASS$

$section : \mathbb{P}\ SECTION$

$address: \mathbb{P}\ ADDRESS$

$stuname : ROLLNO \rightarrowtail\!\!\!\!\!\!+\!\!\!\!\!\!\rightarrow NAME$

$stuclass : ROLLNO \rightarrowtail\!\!\!\!\!\!+\!\!\!\!\!\!\rightarrow CLASS$

$stusection : ROLLNO \rightarrowtail\!\!\!\!\!\!+\!\!\!\!\!\!\rightarrow SECTION$

$stuadd: ROLLNO \rightarrowtail\!\!\!\!\!\!+\!\!\!\!\!\!\rightarrow ADDRESS$

$rollno = \text{dom}\ stuname$

$rollno = \text{dom}\ stuclass$

$rollno = \text{dom}\ stusection$

$rollno = \text{dom}\ stuadd$

AddStudentDetails

$\Delta StudentsDetails$

$rollno? : ROLLNO$

$name? : NAME$

$class? : CLASS$

$section? : SECTION$

$address?: ADDRESS$

$rollno? \notin rollno \wedge$

$rollno' = rollno \cup \{ rollno? \}$

$stuname' = stuname \cup \{rollno? \mapsto name?\} \wedge$

$stuclass' = stuclass \cup \{rollno? \mapsto class?\} \wedge$

$stusection' = stusection \cup \{rollno? \mapsto section?\} \wedge$

$stuadd' = stuadd \cup \{rollno? \mapsto address?\}$

STUDENT_REPORT ::= okadded | alreadyPresent

OkAdded

$result : STUDENT\_REPORT$

$result! : STUDENT\_REPORT$

$result! = okadded$

AlreadyPresent

$\Xi StudentDetails$

$rollno? : ROLLNO$

$result : STUDENT\_REPORT$

$result !: STUDENT\_REPORT$

$rollno? \in ROLLNO$

$result! = alreadyPresent$

5.2. Verification

To type check a document clicks the fuzz button in Z/Word tool [13]. The results are displayed in a dialog box.

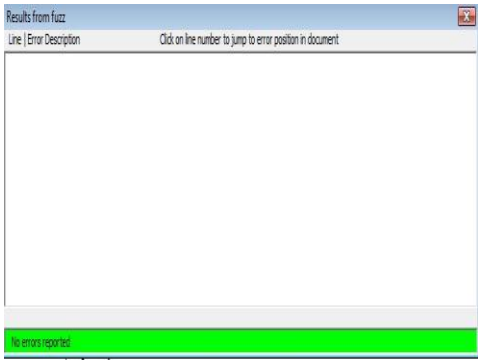

Figure 5.1: Type checking the Z specifications.

6. Conclusions

Following conclusions are drawn:

- Z is one of the numbers of specification languages which are being developed around the world. Z can be used to compactly specify real systems (ATM). Z has various collection of library (Mathematical Toolkit), which supports user to specify the requirements without any ambiguity.
- Large specifications are achievable in Z, using the schema notation for structuring. Also it is possible to produce hierarchical specifications. A part of a system is specified in isolation, and then put into a global context.
- By applying formal method in terms of Z notation, it is observed that it does not require a high level of mathematics rather it requires knowledge of basic set theory and first order logic for the analysis of a complete system.
- Difficulties with Z are cannot do concurrency, Timing aspects, Algorithmic aspects and programming constraints, and Sequencing operations.

6. References

[1] J. Davies and J. Woodcock, "Using Z: Specification, Refinement, and Proof", In Prentice Hall, 1996.

[2] R. Pressman, "Software Engineering- A Practitioner's Approach", McGraw Hill, 5th edition, 2000.

[3] D. Bjorner, Pinnacles of software engineering: 25 years of formal methods‖ , In Annals of Software Engineering, vol. 10, pp. 11–66, 2000.

[4] C. B. Jones, "Systematic Software Development using VDM", In Prentice Hall, 1990.

[5] J.R. Abrial, "The B Book - Assigning Programs to Meanings", Cambridge University Press, 1996.

[6] P. D. MOSSES, CASL Reference Manual: The Complete Documentation the Common Algebraic Specification Language‖ , Lecture Notes in Computer Science, Springer-Verlag, Vol. 2960, 2004.

[7] J. A. Gougen and J. J. Tardo, An introduction to OBJ: a language for writing and testing formal algebraic Specifications. In The IEEE Conference on specifications of Reliable Software. IEEEComputer Society Press, pp. 170-189, 1979.

[8] C. A. R. Hoare, Communicating Sequential Processes‖ , In Prentice Hall, NJ, 1985.

[9] R.Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee, Modular specification of hybrid systems in CHARON. In Hybrid Systems: Computation and Control, Third International Workshop, HSCC,2000.

[10]J.R. Abrial, S. A. Schuman and B. Meyer: "A Specification Language, in On the Construction of Programs", Cambridge University Press, eds. A. M. Macnaghten and R. M. McKeag, 1980.

[11] J.M. Spivey, "The Z Notation, Reference Manual", 2nd edition, Prentice Hall International, 1992.

[12] M. Joseph, ―Formal Techniques in Real-Time and Fault-Tolerant Systems‖ , Lecture Notes in Computer Science 331, pp. 160- 174.

[13] http://sourceforge.net/projects/zwordtools/.