VOL. 10, NO. 3, FEBRUARY 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

FORMAL SPECIFICATION APPROACH IN DESIGNING DATABASE SYSTEM USING Z

Julaily Aida Jusoh[1], Mohd Yazid Md Saman[2] and Mustafa Man[2]
[1]Department of Information Technology, Faculty of Informatics & Computing, Tembila Campus, Universiti Sultan Zainal Abidin, Besut, Terengganu, Malaysia
[2]Pusat Pengajian Informatik & Sains Gunaan, Universiti Malaysia Terengganu, Kuala Terengganu, Terengganu, Malaysia
E-Mail: julaily@unisza.edu.my

ABSTRACT

The requirements for a software system set out what the system should do and define constraints on its operation and implementation. Traditionally, users express the requirement specification of system development by natural language. Natural language is ambiguous, thus the requirement specification statement may result in different apprehension among users, analysts and programmers. Anxieties have been raised by several relevant software developers about the weakness use of natural languages in documenting system requirements. Hence, one of the solutions to solve the problem in capturing user requirements is by using the formal specification approach. Formal specifications is an approach to solve the vagueness and contradiction in natural language descriptions by providing an unambiguous and precise specification. This approach can be validated and verified mathematically leading to the initial detection of specification errors. The use of a formal specification will reduces ambiguity between programmer and end-user while eliminates an errors during software development. If this is done, then we can carried out testing the system aligned to the user requirement specification. This work will discuss on how to implement formal specification approach in designing database system.

Keywords: natural language, software requirement specification, formal specification.

INTRODUCTION

Software specification is the process that beginning by defining the user requirement. User requirement is a statement that expressed in natural language plus diagrams, on what services are required from the system and determine the constraints on the system's operation and development. While system requirement is a guideline of what functions that the system must provide. To reduce ambiguity, both requirements may be written in a structured form of natural language complemented by system models and tables. The statement of the user requirements and a detailed specification regarding the system requirements usually documented in the software requirements specification (SRS). There two cases in documenting user requirement and system requirement either both requirements are integrated into a single description or the user requirements are defined in an introduction of the system requirements specification. But, the detailed of system requirements may be presented in a separate document if there are a bundle of user requirements. In any software process model, the first phase is the requirement analysis. This phase describes on what the system are required to do and the constraints in the operation. At some stage in analysis, data are collected through interviews, on-site observation and questionnaire (Sommerville, 2011), (Pressman & Maxim, 2014).

In a software development process, traditionally, a user may articulate the requirement specification of system by natural language. Natural language description is ideal for human communication, but it causes damage of the quality of requirements (Saeki & Enomoto, 1989) and ambiguous. Thus the requirement specification statement may result in dissimilar perception among users, analysts and programmers (Jochen, 2003). Concerns have been raised by several relevant software developers about the poor use of natural languages in traditional documenting system requirements. One of the solutions to solve the problem in capturing user requirements is by using the formal specification approach (Mohd Saman et.al, 2006), (Mohd Saman et.al, 2007).

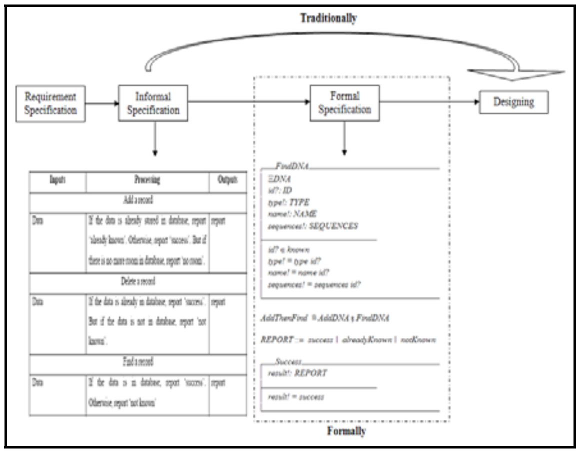

Traditionally
Requirement Specification
Informal Specification
Formal Specification
Designing
Inputs
Processing
Outputs
Add a record
Data
If the data is already stored in database, report 'already known'. Otherwise, report 'success'. But if there is no more room in database, report 'no room'.
report
Delete a record
Data
If the data is already in database, report 'success'. But if the data is not in database, report 'not known'.
report
Find a record
Data
If the data is in database, report 'success'. Otherwise report 'not known'.
report
FindDNA
ΞDNA
id?: ID
type!: TYPE
name!: NAME
sequences!: SEQUENCES
id? ∈ known
type! = type id?
name! = name id?
sequences! = sequences id?
AddThenFind ≙ AddDNA ⨟ FindDNA
REPORT ::= success | alreadyKnown | notKnown
Success
result!: REPORT
result! = success
Formally

Figure-1. The requirement specification process for designing DNA database.

The system must be designed based on the detailed analysis and the user requirements of a system. Normally, informal specification is turned out at requirement specification phase. Then this informal specification generally applied on software requirement specification (Sommerville, 2011), (Pressman & Maxim, 2014). Software requirement specification considered as a complete informal description of system requirements and dependencies at a particular point in time prior to any actual design and development work. Figure-1 shows the proposed process from requirement specification to designing by implementing formal specification after informal specification.

Formal specifications formally defined the vocabulary, syntax and semantics in a mathematical notation (Michael & Jeremy, 2014). Usually both the system requirements and the system design are expressed in detail and carefully analyzed and checked before starts the implementation phase. Developing formal specification of the software usually comes after the system requirements have been specifying but before the detailed systems are design (Jusoh et.al, 2009). One of the main benefits of formal specification is its ability to discover problems and ambiguities in the system requirements. This work will analyse informal specifications for traditional DNA sequence database systems in order to identify the structure of the requirements and transform into formal specifications using Z language. The correctness of these specifications will be verified and validated focuses on requirement analysis of the system development.

FORMAL SPECIFICATION

Formal methods have been used in computer system development for decades. As mentioned in (Man et.al, 2012), formal methods use the ideas and techniques from mathematical and formal logic to increase the design assurance and eliminate defects at the early stage to fulfill user requirements. Formal methods developed complete, consistent and unambiguous specification using set theory and logic notation to create a clear statement of requirements (Saaltink, 1997), (Diller, 1994). In safety-critical systems, a failure rate may be high. Lives may be lost or severe economic consequences can arise when computer software fails. In such situations, it is crucial that errors are exposed before software is put into operation (Spivey, 1989), (Wordsworth, 1996). A formal method radicall eliminate the specification errors and as a consequence serves as the basis for software that has very few errors once the customers begin using it. The primary concepts in formal methods are (Diller, 1994):-

- Data invariant – a condition true throughout the execution of the system that contains a collection of data.
- State – the stored data that a system can access and alters.
- Operation – an action of reads or writes data to a state. An operation associated with two conditions: a pre-condition and a post-condition.

Some methods in formal methods are based on set theory and first order predicate calculus while other methods are based on temporal logic (Moller et.al, 2007). Set theory and logic notation are used to create a complete, consistent and unambiguous statement of formal requirements. Nevertheless, its main benefit is in reducing the number of errors in systems and the main applicable area is critical systems. In this area, the use of formal methods considered as cost effective because more errors are detected before implementation phase (Sommerville, 2011). Formal methods includes formal specification, specification analysis and proof, transformational development and program verification.

In computer science, formal specification approach declared as the specification of a program's properties that defined by a mathematical description. Most languages in this approach used to formally specify programs associated with operational semantics to allow execution of the specification and testing a program's properties. Formal specification describes on what the overall system should do and not how the system should do. It is possible to use formal specification to assist developer develop an accurate system with a few errors at the end of the system development's phase.

Formal specification usually reveals errors and inconsistencies in the informal requirement's specification (Man et.al, 2011). This error detection is probably the most potent argument for developing a formal specification. Fixing errors at this phase is inexpensive compared to modifying a delivered system (Muhammad et.al, 2008). Formal specification can assist in overcoming the mismatch between users and programmers intention by providing an conciseness unambiguous specification in which the intentions of a design can be formulated (Saeki & Enomoto, 1989). This reduces requirement errors as it forces a detailed analysis of the requirements.

Formal specifications use mathematical notation to describe in a precise way the system properties. Formal specification languages consist of Vienna Development Method (VDM), OBJ, Larch, Communicating Sequential Processes (CSP), Lotos, Petri-Nets, Z and B. There are two fundamental approaches in formal specification that have been used to write detailed specifications for the systems (Sommerville, 2011). These fundamental approaches consist of algebraic approach and model-based approach. The language for model-based approach such as B (Wordsworth, 1996), Vienna Development Method (Jones, 1986) and Z (Spivey, 1989) are currently used in a variety of industrial case studies and supported by commercial tools. Z, VDM and B are formal specification language that implemented in different logic and rule. Between that all those languages, Z is chosen as formal specification language to specify DNA sequences database system. This is because Z language has an available support tool, and it becomes acceptable in a software specification lesson and industry. However, the main reason in choosing Z language is because its specification is readable.

In software development lifecycle, formal specification language implemented during requirements analysis, system analysis and designing phase. Normally, specification languages are not directly executed in a form of high level programming language. But most high level programming languages will executed formal specification languages as a complement to develop a system. Formal specification languages describe the system at much higher level compared to a programming language. Indeed, it is considered as errors if a requirement specification is cluttered with unnecessary implementation detail (Man, Mohd Rahim, Jusoh, & Zakaria, 2012).

In this work, most concern is given to model-based approach. Model-based approach modeled the system using mathematical notation such as sets and functions. It exposed the system state which simplifies some types of behavioural specification. Therefore, the system's operations in a model-based specification are defined by pre-conditions and post-conditions on the system state.

Z SPECIFICATION FOR DNA DATABASE SYSTEM

DNA is recognized as Deoxyribonucleic Acid. It is string sequences consisting 4 letter alphabet of nucleotides (bases): A (adenosine), C (cytosine), G (guanine) and T (thymine) (Mohd Saman et.al, 2006). DNA is a nucleic acid molecule that includes the genetic information used in the development and functioning of all living organisms. The DNA segments that hold this genetic information are called genes (A Rahman et,al, 2007).

Over the past century there has been a dramatic increase in the technology of databases. It incorporate both "public" repositories of gene data such as GenBank or the Protein DataBank (PDB), and "private" repositories which used by the work-groups that concerned in gene mapping projects held by biotech industries. These databases are accessible via open online platform likes Web. It is very significant since clients of bioinformatics data are preferred use a range of computer platforms. DNA defined as the proteins which we can trace the genetic information about an organism in it. DNA macromolecules have a fixed structure which can be analyzed by biologists assists by bioinformatics tools and databases. A few acceptable databases among biologists are GenBank from NCBI (National Center for Biotechnology Information), SwissProt from the Swiss Institute of Bioinformatics, BLAST and PIR from the Protein Information Resource. The motivation on utilizing DNA sequence files as a case study is based on the increasing scientific and societal need towards human genome data (Zhang et.al, 2011). In the next few years, various fields including academia, business, and public health will take advantage, of the huge information stored in the DNA sequence database system.

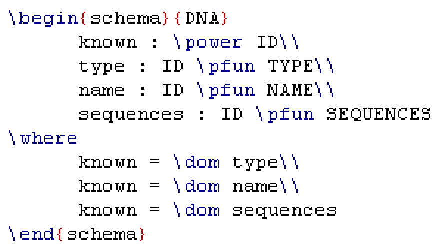
There are three general types of DNA database consists of forensic, genetic genealogy and medical. Those DNA databases determined data that are related with computerized software designed which can be insert, remove, update and retrieve components of the data stored within the system. A simple database might be a single file containing numerous records which includes the similar set of information. For instance, a record regarding a nucleotide sequence database normally contains information such as contact name, a molecule of sequences, type of the molecule description and the scientific name of the source organism from which it was isolated.

The DNA sequences have structural purposes that involved in regulating the use of genetic information. Genetic information and personal data are stored indefinitely on DNA databases. The DNA database system is a system that hold millions of DNA sequences. The DNA database are designed to provide and encourage access within the scientific community to the most up to date and comprehensive DNA sequence information (Jusoh et.al, 2009). Nevertheless, the development in sequence databases and technology in solving a DNA sequences similarity search becomes vital issues nowadays.

This section review on specification of traditional DNA database system. In order to develop a Z specification for DNA database system, a few related data set will determined. These data set are represented by id of DNA, type of DNA, name of DNA and sequences of DNA. The declaration of Z specification for DNA database system are clarified as follows:-

- This specification makes use of the following given sets:
  [ID, TYPE, NAME, SEQUENCES]
- Database of Data : DNA
- The type definition for the responses to the operations is as follows: REPORT ::= successs | alreadyKnown | notKnown
- Note : \Delta means there is a change in state schema
  \Xi means there is no change in state schema

The first step in designing Z specification is describing the state schema of the system as illustrated in Figure-2. It is similar as most schemas which consists of declaration part above the central dividing line and a predicate part below the central line which gives a relationship between the values of the variables.



```
\begin{schema}{DNA}
      known : \power ID\\
      type : ID \pfun TYPE\\
      name : ID \pfun NAME\\
      sequences : ID \pfun SEQUENCES
\where
      known = \dom type\\
      known = \dom name\\
      known = \dom sequences
\end{schema}
```

Figure-2. State schema for DNA databases.

Figure 2 depicted a state schema in latex format. There are the four variables declared in this schema. It represents important annotations as follows:

- known is the set of ID with DNA recorded.

VOL. 10, NO. 3, FEBRUARY 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

- type is a molecule description which applied to certain ID, the system gives the DNA's type associated with the ID.
- name is a organism scientific name, the system will gives the DNA's name associated to the ID that applied by biologist.
- sequences is a DNA sequences, which once biologist key-in the ID, the system gives the DNA's sequences associated with them.

As presented in Figure 2, the predicate part of the schema below the line gives a relationship which is true in every related schemas of the system. In this case, it says that the set known as the domain of the function type, name and sequences - the set of ID to which it can be validly applied. This relationship is a condition that can be relied upon to be true during the execution of the system. This condition also called as an invariant. In this state schema, the invariant allows the value of the variable known to be derived from the value of sequences. The known is a derived component of the state which is possible to specify the system without mentioning known at all. However, ID helps the specifications more readable because an abstract view of the state schema of the DNA databases are described. It can be done without any commitment to signify known explicitly in an implementation. One possible state of the system is the following:

known = {001, 002, 003}
sequences = {001 ÷ TTAGCCGAAAGGT,
002 ÷ AAATGCCTATGCC,
003 ÷ CAGTTTGCAAGGTT}

There are three ID's known to the system. The sequences function as associates a DNA sequence with related ID. In the state schema, there is no limitation on the number of DNA recorded into the database. The second steps in designing Z specification is develop a formal specification of related operations in the database system. An operation schema represents the operation that the system will perform. The first operation is inserting a new discovered DNA data into the system. The Z specification schema for this operation illustrates in a Figure-3.

```
\begin{schema}{AddDNA}
    \Delta DNA\\
    id? : ID\\
    type? : TYPE \\
    name? : NAME \\
    sequences? : SEQUENCES
\where
    id? \notin known\\
\also
   type'=type\cup \{(id?\mapsto type?)\}\\
   name'=name\cup \{(id?\mapsto name?)\}\\
   sequences'=sequences\cup\ {(id?\mapsto sequences?)\}
\end{schema}
```

Figure-3. AddDNA schema.

The declaration of \Delta DNA means that this schema incorporates the state schema which permits as reference to the state variables before and after of this operation. Next, it is followed by the declarations of the four inputs key-in into the operation. As depicted in Figure-3, the names of inputs end with a question mark. AddDNA schema introduces four new variables; known', type', name' and sequences'. These four variables are observations of the state after the change. Each pair of variables is unconditionally constrained to satisfy the invariant in the predicate part of this schema. It hold both condition before and after the operation. In predicate part, a pre-condition for the operation is check the new id to be added must not already be one of those known to the system that presented as id? \notin known. If the pre-condition is satisfied, the next three predicate shown that those three functions are added by mapping the new data related to the given new id.

```
\begin{schema}{DelDNA}
      \Delta DNA\\
      id? : ID
\where
      id? \in known\\
\also
      type' = \{id?\} \ndres type \\
      name' = \{id?\} \ndres name \\
      sequences' = \{id?\} \ndres sequences
\end{schema}
```

Figure-4. DelDNA schema.

Figure-4 illustrated the delete operation to remove an existing DNA data from the database. Predicate id? \in known shows that given id? to be delete must already a member of known in the system. If the pre-condition is satisfied, hence, the next three predicates shown those three functions automatically will be deleted from the database. This situation also can be formulated as follows:

$$\begin{aligned}&DNA \wedge id? \in ID \wedge id? \in known\\ &\Rightarrow (\exists\, known' \colon \mathbb{P}\, ID;\ name' \colon \mathbb{P}\,(ID \times NAME);\\ &\quad sequences' \colon \mathbb{P}\,(ID \times SEQUENCES);\\ &\quad type' \colon \mathbb{P}\,(ID \times TYPE) \bullet\ DelDNA)\end{aligned}$$

The next operation in this database system is to retrieve the data from the database system. Next operation describes in a schema as depicted in a Figure-5.

```
\begin{schema}{FindDNA}
      \Xi DNA\\
      id? : ID\\
      type! : TYPE \\
      name! : NAME \\
      sequences! : SEQUENCES
\where
      id? \in known\\
\also
      type! = type (id?) \\
      name! = name (id?) \\
      sequences! = sequences (id?)
\end{schema}
```

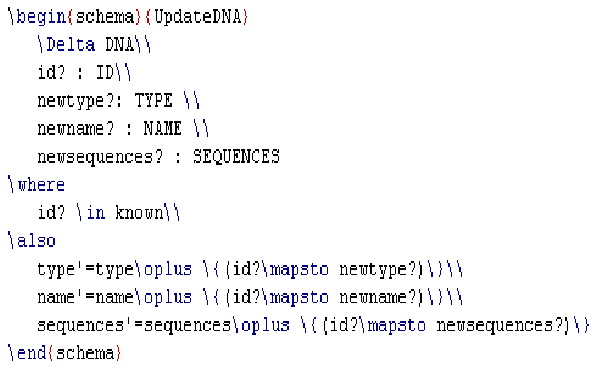
Figure-5. FindDNA schema.

This schema declared two new notations. First notation is \Xi DNA, which indicates that this operation will not cause any changes in the state schema. The second notation is the use of a name which end with an exclamation mark for an output declaration. FindDNA operation takes an id as input and yields the corresponding type, name and sequences as output.

$$\Xi DNA \wedge id? \in ID \wedge type! \in TYPE \wedge$$
$$name! \in NAME \wedge sequences! \in SEQUENCES$$

$$\Rightarrow (id? \in known \Rightarrow id? \in \mathrm{dom}\, type)$$
$$\wedge (id? \in known \wedge type! = type\ id?$$
$$\Rightarrow id? \in \mathrm{dom}\, name)$$
$$\wedge (id? \in known \wedge type! = type\ id?$$
$$\wedge\ name! = name\ id?$$
$$\Rightarrow id? \in \mathrm{dom}\, sequences)$$

The pre-condition for this operation is same as previous operation which is presented as id? \in known. If this is so, the output type!, name! and sequences! are the data of the three functions at argument id? as presented by last three predicates. The last operation this database system is to update the current DNA. This operation is described in a schema as provided in a Figure-6.



```
\begin{schema}{UpdateDNA}
   \Delta DNA\\
   id? : ID\\
   newtype?: TYPE \\
   newname? : NAME \\
   newsequences? : SEQUENCES
\where
   id? \in known\\
\also
   type'=type\oplus \{(id?\mapsto newtype?)\}\\
   name'=name\oplus \{(id?\mapsto newname?)\}\\
   sequences'=sequences\oplus \{(id?\mapsto newsequences?)\}
\end{schema}
```

Figure-6. UpdateDNA sequence schema.

As depicted in a Figure-6, there are four inputs given in this system. The first input, id?, means that all data associated with this id will be updated. The input id? must already be one of those known to the system. If this pre-condition is satisfied, the last three predicates will updates to override the existed data associated to the given id?

CONCLUSIONS

Using formal specification in the software development of an information system will eliminate all ambiguity start from the interpretation of the need/ requirement (the model) and end by elaborating the system which realizes the specification in successive stages. The coherence of the model and the conformity of the final program in relation to this model are guaranteed by mathematical proofs. This work has described the mathematical notation in the state schema of DNA database system and the associated operations which can be performed on it. The related data that used in the system are described in terms of mathematical data types such as sets and functions. One of our future works shall deal with complete and precise specification for DNA sequences pattern scanning.

REFERENCES

A Rahman, M., Mohd Saman, M., Ahmad, A., & M Tap, A. (2007). Automaton Based Filtering in Optimal DNA Sequence Similarity Search. 1st Regional Conference on Computer Science & Technology. Malaysia.

Diller A. (1994). Z: An Introduction to Formal Methods. John Wiley & Sons.

Jochen L. L. (2003). Current Issues in Software Engineering for Natural Language Processing. Proceedings of the HLT-NAACL Workshop on Software Engineering and Architecture of Language Technology Systems, ACM,vol. 8.

Jones C. B. (1986). Systematic Software Development Using VDM. London: Prentice Hall.

Jusoh J., Mohd Saman M. AND Man M. (2009). Formal Validation of DNA Database Using Theorem Proving Technique. International Journal of Computer, the Internet and Management, 21.1-21.5.

Man M., Jusoh J., Mohd Rahim M. and Zakaria M. (2011). Formal Specification Validation For SIDIF Using Theorem Proven. Journal Of Computing.

Man M., Mohd Rahim S., Jusoh J. and Zakaria M. (2012). Mustafa Man, Mohd. Julaily Aida Jusoh, Mohammad Zaidi Zakaria, Designing Multiple Types of Spatial and Non Spatial Databases Integration Model Using Formal Specification Approach. International Journal of Digital Content Technology.

Mohd Saman M., Jusoh J. and Man M. (2006). A Study of Z Formal Specification Language With a Case Study In The Development of Database System. Computer Sciences and Mathematics Symposium Terengganu. Malaysia: UMT.

Mohd Saman M., A Rahman M., Ahmad A. and M Tap A. (2006). A Minimum Cost Process in Searching for a Set of Similar DNA Sequences. 5th WSEAS International Conference on Telecommunications and Informatics, (pp. 348-353). Istanbul, Turkey.

Mohd Saman M., Jusoh J., Ledru Y. and Man M. (2007). On Transformation of Diagrams for DNA Database to Z Specifications. National Conference on Software Engineering & Computer Systems (NaCSES). Malaysia.

Moller M., Ernst-Rudiger Rasch H. and Wehrheim H. (2007). Integrating a Formal Method into a Software Engineering Process with UML and Java. Formal Aspects of Computing.

Muhammad M., Mohd Saman M. Y., Jusoh J. A., Man M., and A Rahman M. N. (2008). Formal Specification of Aho Corasick Algorithm. 4th Malaysian Software Engineering Conference (MySec). Malaysia.

Pressman R. and Maxim B. (2014). Software Engineering: A Practitioner's Approach. McGraw Hill.

Saaltink M. (1997). The Z/EVES System. ZUM (pp. 72-85). LNCS.

Saeki H. H. and Enomoto H. (1989). Software development process from natural language specification. 11th international conference on Software engineering ACM, pp. 64-73.

Sommerville I. (2011). Software Engineering. UK: Addison-Wesley.

Spivey J. M. (1989). An Introduction To Z And Formal Specifications. Software Engineering Journal.

Wordsworth J. (1996). Software Engineering with B. Wokingham: Addison-Wesley.

Michael J. B. and Jeremy L. J. (2014). On Integrating Confidentiality and Functionality in a Formal Method. Formal Aspects of Computing, Springer, DOI10.1007/s00165-013-0285-4, 963-992.

Zhang J. R., Chiodini A. B. and Zhang G. (2011). The impact of next-generation sequencing on genomics. Journal of Genetics and Genomics, vol. 38, 95-109.