Journal of the Nigerian Association of Mathematical Physics
Volume27, (July, 2014), pp 509 – 514
© J. of NAMP

Formal Representation for Ascertaining Language Domain Reliability

Akwukwuma V.V.N and Obi J.C.

Department of Computer Science, University of Benin, P.M.B. 1154. Benin City. Nigeria.

Abstract

Formal specification uses mathematical notation to specify clearly and unambiguously the properties a safety software system should possess. They focus on "What" and not "How". This research paper attempts to define the properties a reliable system ought to possess utilizing Zed notation in specifying these properties wherein interaction within the system was visualized using Unified Modeling Language (UML) sequence diagrams. The findings are: system failures are eliminated as reliability will be upheld to a large extent while time usage invested in ratifying unknown errors will be saved. This research paper has provided a sample representation of formal specification.

Keywords: Formal Specification,MATLAB Schema, UML, Z-Notation.

1.0 Introduction

Language Reliability (LR) is a subset Program reliability is an important factor affecting system reliability. It differs from hardware reliability in that it reflects the design perfection, rather than manufacturing perfection [1]. Reliability is an essential program attribute. Reliability is closely linked with the quality of measurement which is determined by the "consistency" or "repeatability" of program measure [2].

The stability or consistency of scores over time or across raters determines reliability. Reliability pertains to scores not people. Thus, it is highly inadequate to associate reliability with persons. The root causes of program (application) unreliability are found in non- compliance with coding best practices (CBP). This non-compliance can be detected by measuring the static quality attributes of application program reliability criteria. Assessing the static attributes underlying an application's (program) reliability provides an estimate of the level of program risk and the likelihood of potential application failures and defeats the application will experience when placed in operation [2].

System reliability is tied to preciseness in the specifications of system properties eliminating ambiguities and vagueness. Formal specification is an important ingredient in safety critical systems in which failure cannot be tolerated due to the cost associated with it flaws. This research paper attempts to define the properties a reliable system ought to possess utilizing Zed notation in specifying these properties formally, backtracking toward our previous published paper [3].

2.0 Material and Methodology

2.1 Materials

In achieving our objective of determining the objective a reliable system should possess, Z-notation and Unified Modeling Language (UML) serves as the main tools for the methodology of our work.

Z-notation uses mathematical notation to describe in a precise way the properties a software system must possess, without unduly constraining the way in which these properties are achieved [4, 5 and 6]. Formal specification (Mathematical notation or Z) uses mathematical data types to model data in a system and achieve it underlining objectives. These data types are not oriented towards computer representation, but they obey a rich collection of mathematical laws which make it possible to reason effectively about the way a specified system will behave. We use the notation of *predicate logic* to describe abstractly the effect of each operation of our system, again in a way that enables us to reason about their behavior.

The other main ingredient in Z is a way of decomposing a specification into small pieces called *Schemas*. By splitting the specification into schemas, we can present it piece by piece. Each piece can be linked with a commentary which explains informally the significance of the formal mathematics. In Z, schemas are used to describe both static and dynamic aspects of a system [4]. The static aspects includes

Corresponding author: Akwukwuma, E-mail:vakwukwuma@yahoo.com, Tel.: +2348033440003 & 08093088218(O.J.C)

a. the state it can occupy;
b. the invariant (quantity that is unchanged by a set of mathematical operation) relationship that are maintained as the system moves from states to states.

The dynamic aspect Includes:

a. the operation aspect that are possible;
b. the relationship between their input and outputs;
c. the changes of state that happen.

The schema presented in this research paper provided an avenue wherein our formal specification could be presented in fragment enabling us to associate commentary; explain informal the significance of the formal mathematical notation representation.

Unified Modeling Language (UML) is a standard modeling language used for modeling software systems. The focus of UML is on creating simple, well documented and easy to understand software models. The goals of UML are as follows [7]:

a. Provide a simple and easy-to-use expressive visual modeling tool which would be process and language independent
b. Visualize the software with well-defined symbols to represent various elements of a system for developer's unambiguous interpretation of a model written by another developer
c. Construct models of the software system that can easily be implemented with a variety of programming language.
d. Document model of the software system by expressing the requirements of the system during its development and deployment stages.

UML enables system engineers to create a standard blueprint of any system. It provides a number of graphical tools that can be used to visualize a system from different viewpoints. The multiple views (user, structural, behavior, implementation and environment) of the system that is represented by using diagrams together depict the model of the system [8].

2.2 Methodology

This research paper follows through on the methodology proposed, implemented and simulated by [3]; wherein the holistic criteria for determining language domain reliability where framed into a model capable of determining varied language reliability. Genetic algorithm and fuzzy logic were utilized as the methodological tools, the implementation of the system were handled with Matrix Laboratory (MatLab). The system specification properties of the system are proposed with this adjourning research paper owning to the criticality of the system.

The following are some of the basic types in Z{CHAR, STRING, CURRENCY, QUERY, OBJECT, COMPONENTS, BOOLEAN:: = TRUE/FALSE, DATA and OBJECT}

Every system user is authenticated using his username/ID and password on the system

System User

System User_name/ID: seq CHAR
System User_password: seq CHAR
System User: ℙ System User
Access! : Boolean User

System ∈system.access! = accepted) ∨ (system ∉system.access ≠ accepted)

Figure 1: *SystemUser Schema*

There is no frontier to the number of registered system user and each system user can have only one authentication and authorization privilege. Logging on, each system user must register its name as specified in Figure 1.

System list

System: ℙ SYSTEM: ℙ RELIABILITY PROCESSES
System List: SYSTEM → RELIABILITY PROCESESES

Systems = dom list

Figure 2: *LIST Schema*

Figure 2 highlight the *Systemlist* which provide the reliability processes provide by the system.

Register Reliability Processes

Δ System List
System? : SYSTEM
Reliability -Processes? : RELIABILITY PROCESSES
report! : REPORT

(system? ∉ system ∧
System List′ = SystemList U {system? → processes?}∧ report! = ok) ∨
(system? ∉system∧systemlist′ = systemlist∧ Report! = already_known)

Figure 3: *Register-Reliability Schema*

The list braces the facility in registering reliability processes given that the system process does not exist previously. If the process exists previously, a report of 'already known' is returned or vice versa as the case may be as clearly showed in Figure 3.

LocateProcess
Ξ SystemList
Processes? : RELIABILITY PROCESSESS
Systemlist?ℙ RELIABILITY PROCESSES
(systemlist! = {system: systems / list (system) = process? } ∨ (system ∉ systems ∧ report! = not_known)

Figure 4: *Locate process Schema*

The *Locate Process* function obtains a process type as an argument and returns a result responds in line with the schema processes as shown by Figure 4.

AlreadyExistingProcess
Ξ SystemList
system? : SYSTEM
result! : REPORT
process? ∈ Processes
result! = already_known

Figure 5: *AlreadyExistingprocessSchema*

Figure 5, highlights, *AlreadyExistingProcessSchema* which determines a change to the systemlist in terms of new input process.

ProcessNotAvailable
Ξ Processlist
process: ℙ RELIABILITY PROCESS
report! : REPORT
request? : RELIABILITY PROCESS
∀process ∈system.list (process) ≠ Reliability
report! = not_known

Figure 6: *ResponsibilityNotAvailable Schema*

The Figure 6, shows the error which occurs when a system user requests for a process which has not been registered in the systemlist. An error report of 'not known' is returned. The list is initialized at the beginning with no process or reliability process in Figure 7.

Initialize_List
SystemList
PROCESS: ℙ RELIABILITY PROCESSES
systemlist = ∅
reliability process = ∅

Figure 7: *Initialize_List Schema*

2.3 System Design and Unified Modeling Language (UML)

Software design immediately follows the requirements engineering phase in a software process. Software design is the translation of the requirement specification into useful patterns for implementation. UML sequence diagram shows the interaction between classes (or object) in the system for each use case. The interaction represents the order of messages that are exchanged between classes to accomplish a purpose. Figure 8 and Figure 9 specify these interactions.

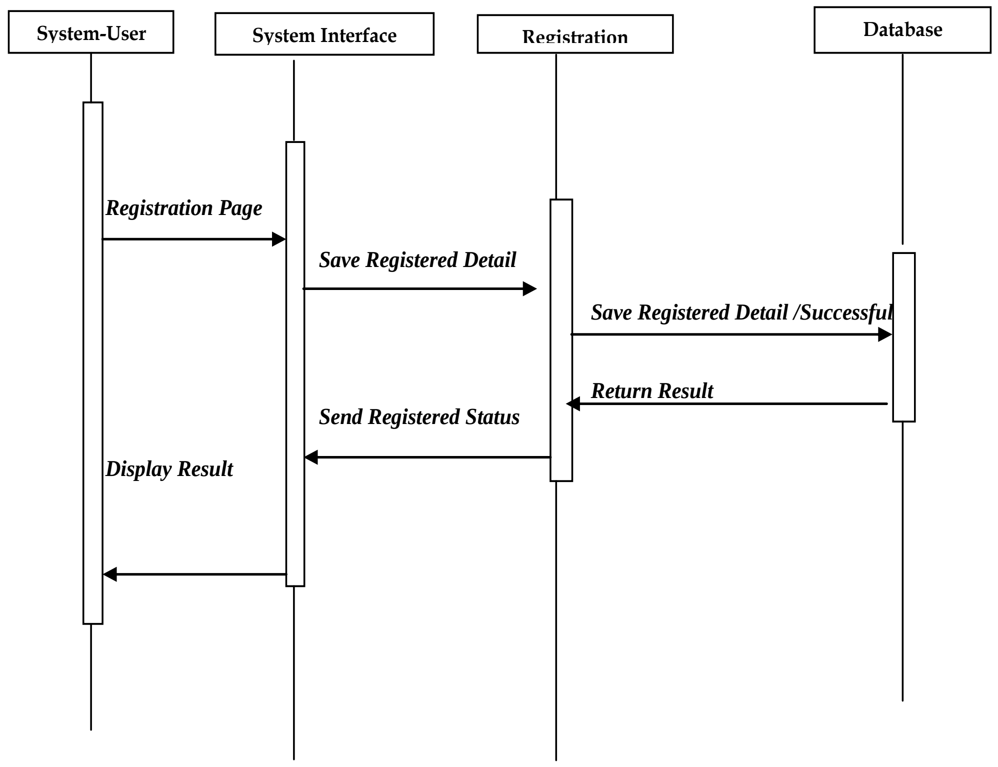

Figure 8: *Registration User Sequence Diagram*
Figure 8, models the sequence of steps involved in the registration of a system user. The order of appearance of the arrows indicates the order of the actions while the arrow direction indicates the direction of flow of events / results.

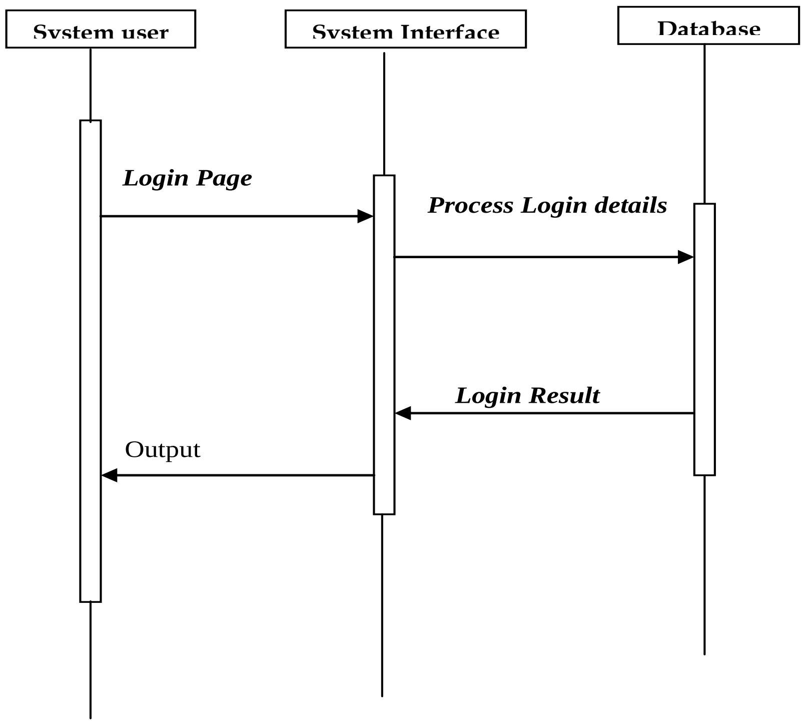

Figure 9: *Login User Sequence Diagram*

Figure 9, models the sequence of steps involved in the log in System user case scenario. The order of appearance of the arrows indicates the order of the actions while the arrow direction indicates the direction of flow of events/results.

3.0 Findings

Based on the ease at which the users get information through this new system, the following are revealed:

a. Prevent system failure as reliability will be upheld to a large extent.

b. Save time otherwise investigated in ratifying unknown errors.

4.0 Conclusion

Formal specification is the bedrock of safety critical systems, which uncovers ambiguities and unwanted errors from system requirement. In Nigeria and African as a whole this approach has not be implemented for most safety critical system opening the avenue for system failure with huge implication. This research paper focuses on providing a sample representation of formal specification. The various segment of the system was specified utilizing UML. The results of the finding were listed assiduously. Formal specification is an avenue for safety critical system which must be explored in as much safety is involved.

5.0 References

[1] Jiantao P. (1999), "Software Reliability", retrieved online from http://ece.cmu.edu/koopman/Des_a 99/sw-reliability/

[2] CISA: Certified Information System Auditor (2011), "CISA Review Manual 2010", Chapter 5, Pp. 305.

[3] Obi and Akwukwuma (2013), "Assessment of Programming Language Reliability Utilizing Soft-Computing", International Journal on Computational Sciences & Applications (IJCSA) Vol.3, No.3 pp. 25 -35.

[4] Spivey J. M. (1998), "The Z Notation: A Reference Manual", Oxford, United Kingdom.

[5] Sannella D., (1988), "A Survey of formal software development methods", appeared in Software Engineering: A European Perspective, A. McGettrick and R. Thayer (eds.), IEEE Computer Society Press, pp 281-297, 1993.

[6] Spivey J. M. (1992), "The Z Notation: A Reference Manual, 2nd Edition", Prentice Hall International (UK) limited, United Kingdom.

[7] Chris M. (2000), “Enterprise Modeling With UML: Designing Successful Software through Business Analyses, Addison Wesley.

[8] Philippe K. (2000), “Rationale Unified Process: An Introduction: Second Edition, Addison Wesley