Journal of Software Engineering and Applications, 2017, 10, 721-733
http://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

An Augmented Framework for Formal Analysis of Safety Critical Systems

Monika Singh, V. K. Jain

College of Engineering & Technology (FET), Mody University of Science & Technology, Laxmangarh, India
Email: Dhariwal.monika@gmail.com

How to cite this paper: Singh, M. and Jain, V.K. (2017) An Augmented Framework for Formal Analysis of Safety Critical Systems. *Journal of Software Engineering and Applications*, **10**, 721-733.
https://doi.org/10.4236/jsea.2017.108039

Received: May 15, 2017
Accepted: July 8, 2017
Published: July 11, 2017

Abstract

This paper presents an augmented framework for analyzing Safety Critical Systems (SCSs) formally. Due to high risk of failure, development process of SCSs is required more attention. Model driven approaches are the one of ways to develop SCSs for accomplishing critical and complex function what SCSs are supposed to do. Two model driven approaches: Unified Modeling Language (UML) and Formal Methods are combined in proposed framework which enables the analysis, designing and testing safety properties of SCSs more rigorously in order to reduce the ambiguities and enhance the correctness and completeness of SCSs. A real time case study has been discussed in order to validate the proposed framework.

Keywords

Unified Modeling Language, Formal Methods, Z Notation, Safety Critical System

1. Introduction

Embedded softwares are found in almost every field of human life such as in medical equipment, air traffic control systems, car airbag, braking systems, nuclear reactor control and cooling systems, aerospace on-board systems, etc. From last decades, these systems are supposed to do and control more complex functions. A minor fault in such systems may cause severe consequences such as loss of human lives, environmental damage or high economic losses. Such systems whose focus is kept on safety are known as Safety Critical Systems [1]. The deed of complex function of safety critical system spiked the growth of model based techniques for development of such system. These techniques are preferred to apply at early stage of software development process [2]. The idea behind these techniques is to develop and analyze a model of the system instead of the assessment of the final system implementation. The model-driven techniques

(MDT) [3] vary in the degree of mathematics used. For example, there is MDT which emphasizes on the graphical representation instead of defining semantics formally such as Unified Modeling Language [4]. On the other hand, there are techniques such as Formal Methods [5] that use rigorous mathematics for defining the semantics. The formal methods are used to develop the system with a high degree of confidence to prevent the failures and enhance the correctness of system behaviour. Generally, the model-driven development (MDD) techniques are used to capture the main features of the system. Moreover, in order to achieve the safety of system, we need to make sure that, even the occurrence of fault in some system components or some unfavorable conditions in the environment, the system will not be put into an unsafe state. This will increase the complexity of system by using the fault tolerance and failure detection mechanisms. Therefore, we require an integrated approach to develop system rigorously which will consider the system integrity along with handling the abstraction, complexity and provide verification by proof. In this thesis, we rely on Z Notation [6] formalism. Since Z notation is an orphan of model-oriented formal methods empire that will start from abstract model and consecutive refinements towards the final model via revised or updated schema for correctness. Revise or updated schema allows us to assure the system safety requirements by stepwise enhancing their representation in the system model along with providing the proof for system properties at various level of abstraction. Figure 1 presents the augmented framework which is wrapping safety properties along with safety analysis is required which includes safety properties, incorporating as the results of safety analysis, transformed into formal models for verification, facilitates formal development and verification of safety-critical systems via pre-defined patterns.

Rest of the paper is organized as follow: Section 2 presents the methodology and

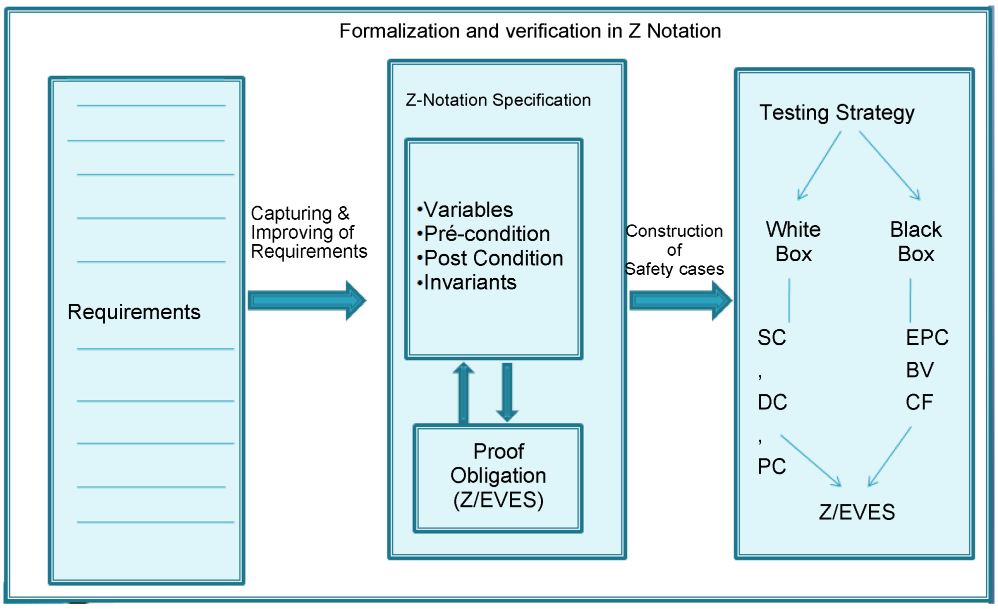


Figure 1. Formal model for developing safety critical systems.

components of augmented framework. Section 3 presents the Formal Model of SCSs. Section 4 represents the simulation and section 5 presents the conclusion.

2. Methodology and Components

The components of this holistic approach are: Unified Modeling Language and Z Notation [5]; a Formal Method and described briefly as follow.

2.1. Unified Modeling Language (UML)

UML [6] is a modeling technique that combines object oriented methods and concepts. It enhances the analysis and design of software system by allowing more cohesive relationships b/w objects. It has been observed that graphical representation of model is easily accessible and understandable to the user. The primary gap between the developer and the user has been easily fulfilled by the graphical description.UML composed of nine diagrams: Use case diagram, class diagram, sequence diagram, state diagram, activity diagram, interaction diagram, component diagram, deployment diagram and package diagram. Graphical representation always gives a better understanding of proposed system.

2.2. Z Notation

Z notation [5] is the oldest formal specification language introduced in late 1970's and developed through the 1080's with the collaboration of two leading brand names; Oxford University with industrial partner IBM and Inmos. The next milestone in the journey of z notation came as a Reference manual in 1989 given by Mile specy. Z uses the standard such as American National Standards Institute (ANSI), British Standard Institute (BSI) and International Organization for standardization (ISO) for writing any draft in Z notation for formal semantics. The Z notation is a model oriented approach based on first order predicate logic and Zermelo-Fraenkel (ZF) set theory used for specifying behavior of abstract data types and sequential programs.

The proposed framework is presented in **Figure 1** which helps in:

1) Capturing and improving the readability of requirements of Safety Critical Systems.

2) Model and verify the correctness of Safety Critical Systems.

3) Construct and test the Safety Critical Cases.

2.3. Methodology

The requirements are captured by using graphical modeling language *i.e.* Unified Modeling language (UML). Once the requirements are captured, Stereotypes are used in order to improve their readability. Formal model had been developed for Use case, Class and Sequence diagram of UNL in order to depict various systems' level properties. For example, Formal Model of use case diagram ensures that the functional requirements are complete, consistent and unambiguous. The Formalization of Class Diagram provide us the correct design specification and formal transformation of testing criteria's helps us to assure that the test specifications are complete and consistent.

Figure 1 presents an overview of the proposed approach. The following steps are associated with this approach: 1) capturing and improving readability of informally defined requirements of safety critical systems, 2) formalization of various aspects of critical systems and formal verification of correctness—by construction as well as 3) validate the required safety properties are met.

3. Formal Analysis of Safety Critical Systems (SCSs)

This section is divided into three segments.

3.1. Capturing and Improving the Readability of Requirements of Safety Critical Systems

The requirements are usually written in natural language. To get a better understanding of system functionality, graphical languages such a Unified Modeling Language (UML) are used. Graphical representation always gives a better understanding of proposed system. The UML—use case diagram defines the behaviour of a system *i.e.* the functionality of the system. Therefore one can get better understanding of system behaviour by making use case diagram of the system which further forms the root of Software Requirement Specification (SRS). Although UML has numerous good attributes yet not accepted for designing the safety critical system alone. One of the reasons is lack of preciousness in semantic used in graphical model. To improve the readability of requirements, Extension Mechanism [7] such as Stereotype, Tagged values and Constraints are used. The UML Extension Mechanisms are used to extend UML by:

1) Adding new model elements, 2) Creating new properties and 3) specifying new semantics.

Extensive Mechanism:

- Stereotype
- Tagged Value
- Constraint

In the context of this paper, Stereotype is used to sever the purpose. A stereotype is a model element that denotes additional values, additional constraint and optionally a new graphical representation. Moreover, a stereotype allows us to attach a new semantic meaning to a model element. The two type of sterotypes are used in this paper:

- **“Include”—**Include is used to extract use case fragments that are *duplicated* in multiple use cases. The included use case cannot stand alone and the original use case is not complete without the included one.
- **“Extend”—**Extend is used when a use case adds steps to another first class use case.

3.2. Model and Verify the Correctness of Safety Critical Systems

The second phase of software Development process is the design phase. In this thesis, formal transformation of model is done by refinement in the design phase. The class diagram forms the root for system structure and sequence diagram forms the basis for system behaviour respectively. According to propose approach, the system development begins with an abstract system model that is further elaborated gradually by providing implementation details in a number of model transformation steps. This approach provides us support to handle the system complexity and to develop the system correct-by-construction. In addition, refinement processes are iterative in nature and feedback is provided by verification. Thus, contradicting, inconsistent and incomplete requirements specifications, design specification can be discovered.

Since class diagram is the basis for system structure and helps in designing modules of a system, but due to component of UML family, lack preciousness in semantic. To address this problem, we construct formal model for class diagram and sequence diagram in order to capture static and dynamic aspect of the system respectively. It guides the developer starting from the informal representation of class and sequence diagram (with UML) to building the corresponding parts of system module via formal modeling and verification in Z Notation and accompany toolsets. In the scope of this approach, we propose the automated Theorem Prover for demonstrating that formal system models are themselves are well defined. Moreover, in order to assure that there is no logical inconsistency and no model element contains any infeasible mathematical definition, simulation of Z specification has been done with Z/EVES [8] tool.

3.3. Construct and Test the Safety Critical Cases

Testing [9] plays an important role for checking the correctness of system implementations. To test system, test cases are formed and system behavior has been observed during execution. Based on test execution, the decision is made for the correctly functioning of the system. However, the criterion for the correctness of test cases has been specified in the system specification. A specification prescribes "What" part of the system *i.e.* the function that a system supposes to do and accordingly forms the foundation for testing criteria. As system specifications are documented in natural language (informal), which is generally incomplete and ambiguous in nature, due to this many problems may occur in testing processes such as incompleteness, ambiguous and inconsistency in test specifications. With an unclear specification, it is next to impossible to predict how the implemented system will behave; consequently testing will be difficult as it is not clear what to test. This become more severs specifically in case of Safety Critical System. We propose an approach to rigorous construction of structured test cases by formalization of test-specification. In other words, formal transformation of testing criteria [10] such as white box and black box is done by refinement pattern. Formal model of each testing criteria such as statement coverage (SC), path coverage (PC), decision coverage (DC), boundary value analysis (BV), equivalence partition class (EPC) and cause & effect is formed. This approach helps the tester to completeness and correctness of test specification in automated environment by Theorem Prover toolset. Z/EVES serve the purpose of simulation in automated environment. Moreover, formal methods are rich in mathematics axioms, supporting the argument that all the model elements definitions are consistent and feasible.

4. Simulation Results

This section is divided into two parts: section 4.1 presents the formal model for safety Critical systems and section 4.2 represents the verification of results with automated Theorem Prover.

4.1. Formal Model

Figure 2 shows the formal model for requirements and design of SCSs by formal transforming the use case, class diagram, and sequence diagram into Z notation element; Schema—a notion for structuring the formal specification written in Z notation. Moreover the formal model for testing specification is depicted in Figure 3.

The testing strategies are broadly divided into two categories: White box testing and Black box testing. White box testing focuses on internal structure of

Class Diagram

Name :ℙ*Name*
Tyoe :*seq char*
Attribute:*seq char*
Attritype :*seq char*

Type ∈ {*base class*,*sub class*}
Attritype ∈ {*public*,*private*,*protected*}

Use Case Diagram

Actor :*seq char*
Relationship :ℙℕ
Event Id :ℙℕ

Relationship Id =*dom relationship*
Event Id =*dom Event*

URelationship

Actor :*seq char*
Rel Type :*seq char*
UC Id : ℙℕ

Rel Type ∈ {□*extend*□,□*include*□;*extend* ∩ *include* =∅
UC Id =*dom Use Case*

Sequence diagram

Objectname :ℙ *Objectname*
Link :*seq char*
MessType:*seq char*

Objectname ∈ *dom objectname*
MessType ∈ {*simple*,*synchronous*,*asynchronous*,*timeout*}
Link ∈ {*Horizontal*,*Vertical*}

Figure 2. Formal model for system design.

_SC

$decinput\,!: \mathbb{P}_1 INPUT$

$decst\,?: STATEMENT$

$decinput\ 0, decinput\ 1: INPUT\text{-}PART$

$decinput = \{i: INPUT \mid decst \in path\ i\}$

$<decinput\ 0, decinput\ 1> partitions\ decinput$

$Dom\ value = decinput$

_DC

ΔSC

$\forall d{:}dec\ \square(test\text{-}data \cap d\ \square decinput\ 0 \neq \varnothing) \wedge$

$(test\text{-}data \cap d \square decinput\ 1) \neq \varnothing$

_PC

ΔSC

ΔDC

$\forall i, j \in \mathbb{N}, (path\ i \cap path\ j = \varnothing) \wedge$

$(path\ i \cup path\ j = test\text{-}data)$

_EPC

$tstdat: TEST\_DATA$

$cls: CLASS$

$\forall i, j \in \mathbb{N}, tst1, tst\ 2 \in TEST\_DATA \wedge$

$tst\ 1 \cap tst\ 2 = \varnothing \wedge$

$\forall i \in \mathbb{N}, cls1, cls2 \in CLASS \wedge$

$cls1 \cap cls2 = \varnothing \wedge$

$\cup cls\ i = CLASS$

BV

$\Delta\ EPC$

$min, max, nominal \in TEST\text{-}DATA \wedge$

$jst\text{-}abv\text{-}min, jst\text{-}blw\text{-}max \in TEST\text{-}DATA$

Figure 3. Formal aspect of testing criteria's.

software artifact. One of the ways to test internal structure is to use either of following scenarios: Statement coverage (SC), decision coverage (DC) or path coverage (PC). Black box testing focuses on the input and output irrespective of internal structure. The famous black box testing criteria's are: Equivalence partitions class (EPC), Boundary Value Analysis (BV), Cause & Effect (C&E). The formal model for all these testing criteria's are formed with Z notation.

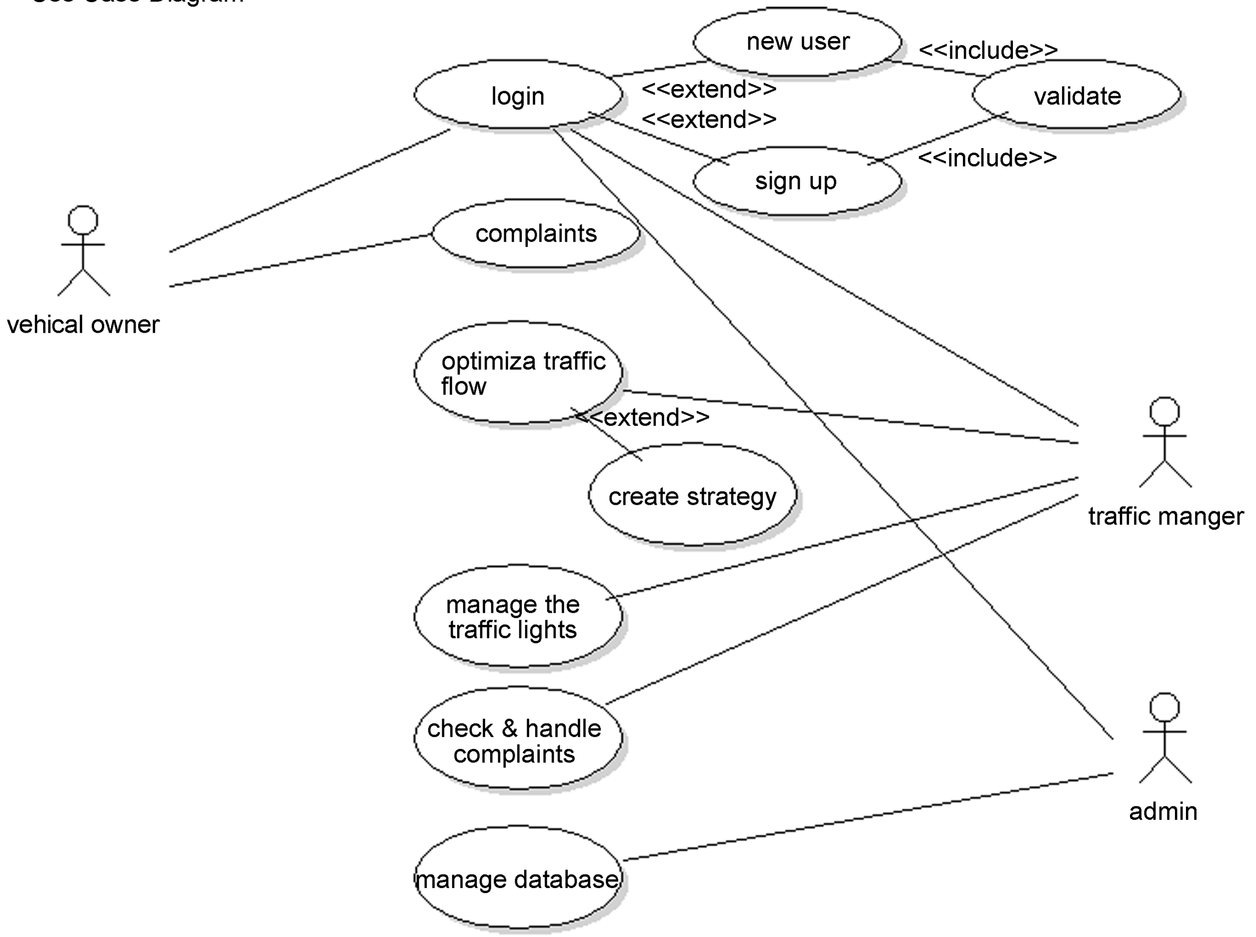
To validate the proposed framework, case study of Road Traffic Management System has been discussed and formal model is shown in Figure 4. RTMS make use of real-time data acquired from the road network in order to reduce traffic congestion and accidents, and to save energy and preserve the environment. The Road Traffic management System (RTMS) has three active actors *i.e.* Admin, Vehicle Owner, Traffic police. Traffic police maintains the information which is provided by the users (Admin, vehicle owners).

The UML model of RTMS consists of Use Case diagram, Class Diagram and Sequence Diagram in Figure 5.

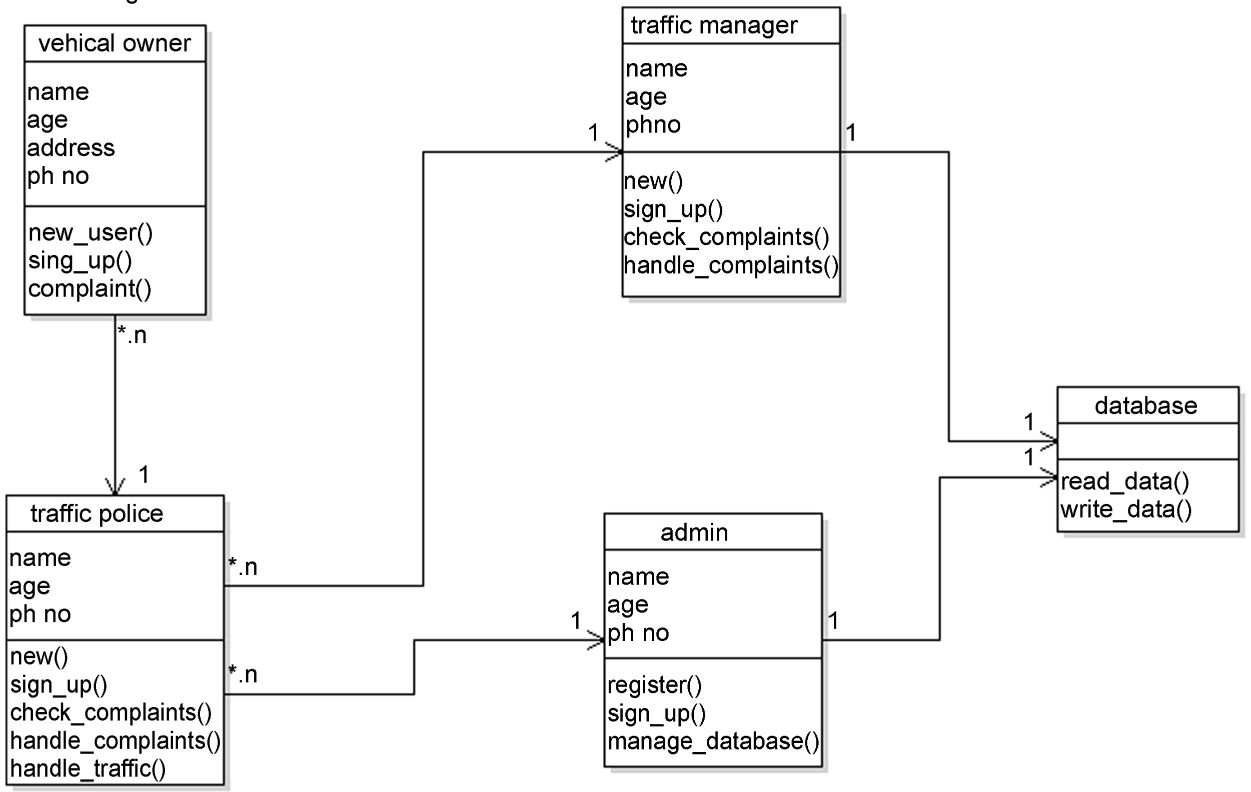
```
Traffic Police______________________________________________
| Tpolice :ℙName
| EmpId :Name□ seqchar
|__________________________
| Tpolice =dom EmpId
|___________________________________________________________

_EmpLogin___________________________________________________
| Password :Tpolice □Word
| EmpId :ℙTpolice
|__________________________
| EmpId =dom password
|___________________________________________________________

_EmpSignin__________________________________________________
| ΔEmpLogin
| Tp? : Tpolice
| P?:Word
|__________________________
| P? = password tp?
| password′  =password
|___________________________________________________________

_VehicleOwner______________________________________________
| Vowner: ℙ Name
| Age: ℙℕ
| Phno: Seqchar
| registVowner: Name □Seqchar
|__________________________
| Vowner = dom registVowner
|___________________________________________________________
```

Figure 4. Formal model for RTMS elements.

Use Case Diagram



Class Diagram



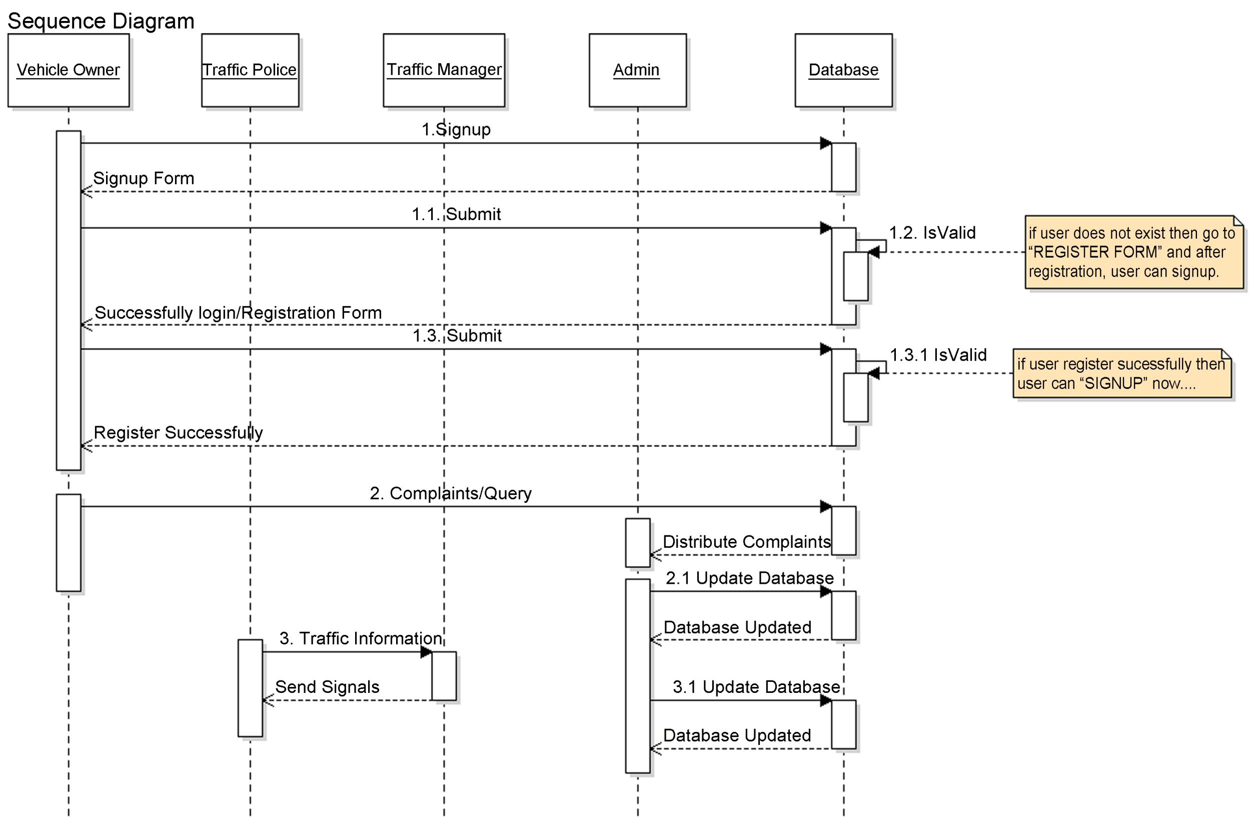


Figure 5. UML model for RTMS.

4.2. Simulation

To ensure the correctness and completeness of specification, design draft and test specification; the formal model is further verified on automated Theorem Prover *i.e.* Z/EVES in Figure 6.

The testing criteria's for construction of safety test case is formally verified in Figure 7 and Figure 8.

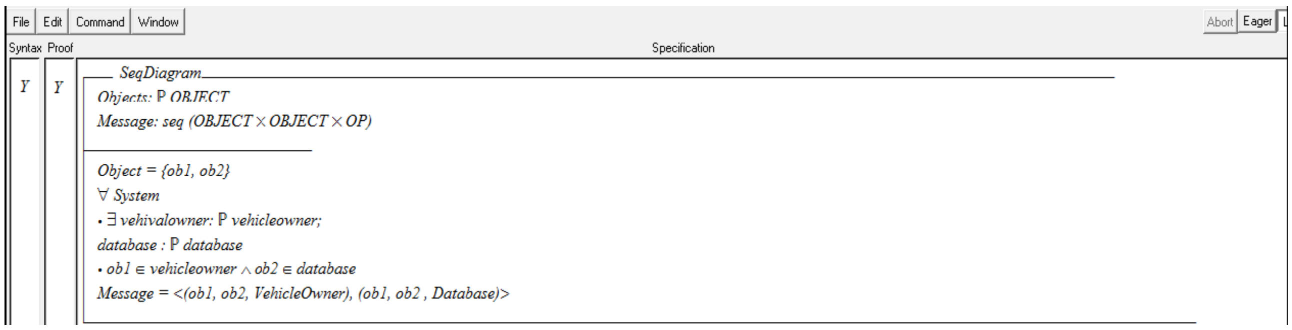
The RTMS elements are further verified on Z/EVES in order to enrich the validation of proposed framework which is shown in Figure 9.

The Table 1 presents the overall model analysis results on automated Theorem Prover and the parameters are: Syntax & Type checking, Domain checking and Proof by Reduction.

5. Conclusions

The proposed framework is a complete, formal, framework which can be used to develop the safety-critical system, with a combination of graphical modeling language. This framework poses the following characteristics:

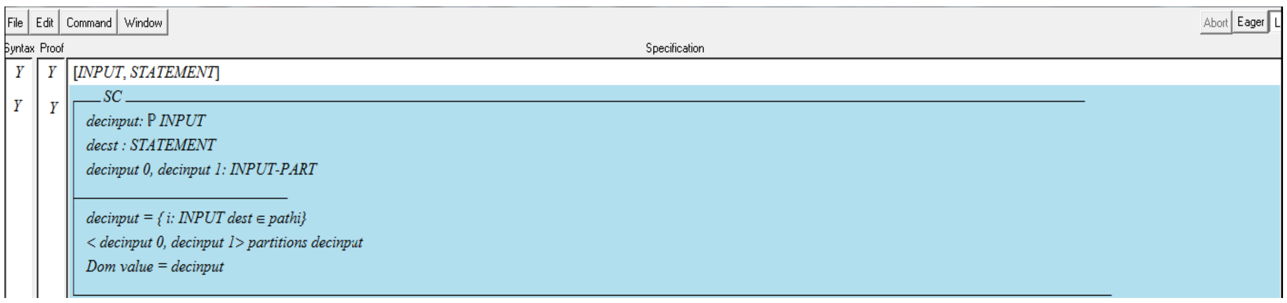
- *Complete*—The proposed method covers all the phases of the software development process *i.e.* starting from specification to design followed by verification and validation.
- *Formal— This method uses rigorous mathematics for* specification, verification and validation. Consequently, provides high level of confidence as compared to conventional manual and informal methods.



SeqDiagram
Objects: ℙ OBJECT
Message: seq (OBJECT × OBJECT × OP)

Object = {ob1, ob2}
∀ System
• ∃ vehivalowner: ℙ vehicleowner;
database : ℙ database
• ob1 ∊ vehicleowner ∧ ob2 ∊ database
Message = <(ob1, ob2, VehicleOwner), (ob1, ob2 , Database)>

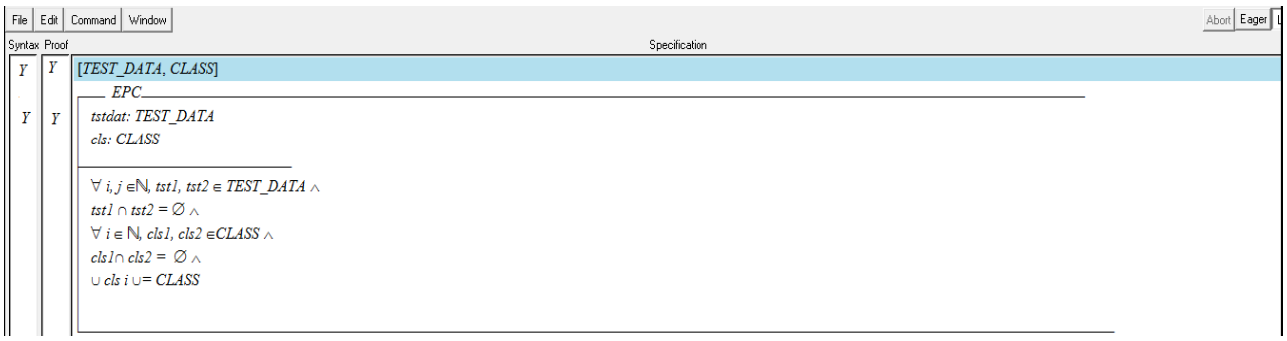
Figure 6. Verification of Z specification on Z/EVES.



[INPUT, STATEMENT]
SC
decinput: ℙ INPUT
decst : STATEMENT
decinput 0, decinput 1: INPUT-PART

decinput = { i: INPUT dest ∊ pathi}
< decinput 0, decinput 1> partitions decinput
Dom value = decinput

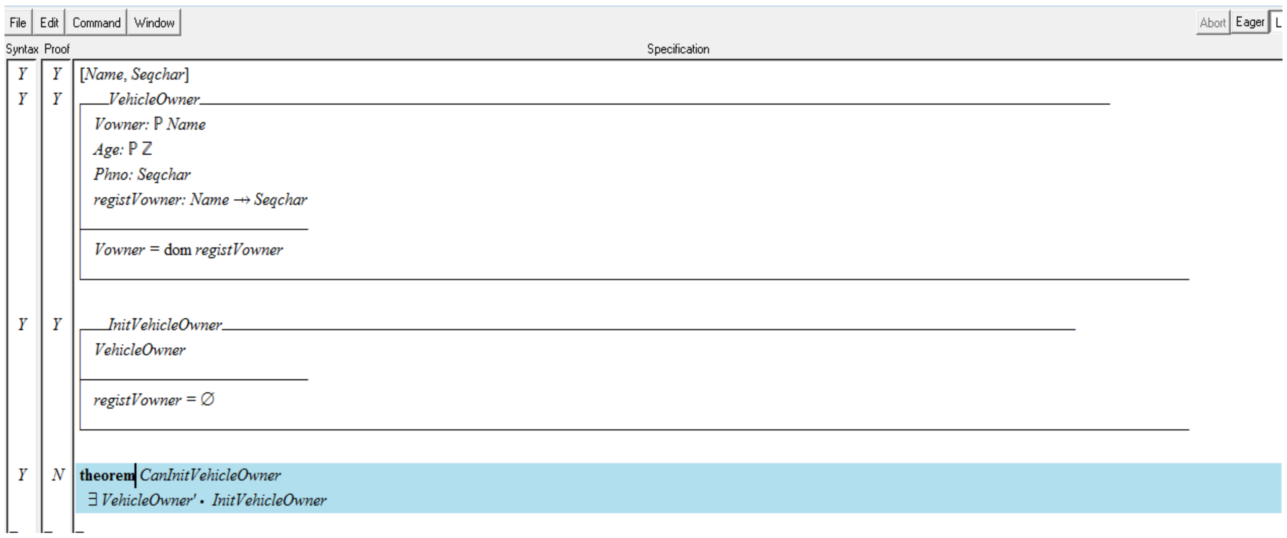
Figure 7. Z/EVES result for Statement Coverage (SC).



[TEST_DATA, CLASS]
EPC
tstdat: TEST_DATA
cls: CLASS

∀ i, j ∊ℕ, tst1, tst2 ∊ TEST_DATA ∧
tst1 ∩ tst2 = ∅ ∧
∀ i ∊ ℕ, cls1, cls2 ∊CLASS ∧
cls1∩ cls2 = ∅ ∧
∪ cls i ∪= CLASS

Figure 8. Z/EVES result for Equivalence Partition Class (EPC).



[Name, Seqchar]
VehicleOwner
Vowner: ℙ Name
Age: ℙ ℤ
Phno: Seqchar
registVowner: Name ⇸ Seqchar

Vowner = dom registVowner

InitVehicleOwner
VehicleOwner

registVowner = ∅

theorem CanInitVehicleOwner
∃ VehicleOwner′ • InitVehicleOwner

Figure 9. Z/EVES model analysis for RTMS elements.

Table 1. Overall model analysis results on automated theorem prover.

SCSs Elements	Verified Via	Syntax & Type Checking	Domain Checking	Proof by Reduction
Requirements Capturing	Use case Diagram	Y	Y	Y
Design Specification	Class Diagram	Y	Y	Y
Construct and Test safety cases	Sequence Diagram	Y	Y	Y

- ***Framework***—Framework presents a 2-*step verification and validation approach*, helps in detecting errors in early phases of development process.
- ***User-friendly***— *Us*ing graphical modeling languages initially to formal transformation, this methodology assists various level of abstraction, which helps different users to understand the system behaviour with ease.
- ***Scalable***—The approach is proficient in verifying the system (safety properties) of industrial size by allowing inductive reasoning and model checking together.
- Facilitate the safety assurance process of formally developed safety-critical systems, by an established link between formal specification and safety cases.
- Enhance the processes of Requirements Elicitation, Formal modeling and Verification.
- Facilitate the system certification by showing how to incorporate safety requirements into Formal model.
- Allows using the verification results of formal models as the evidence for construction of safety cases.

References

[1] Gowen, L.D. (1994) Specifying and Verifying Safety-Critical Software Systems. IEEE Seventh Symposium on Computer-Based Medical Systems, 235-240. https://doi.org/10.1109/cbms.1994.316018

[2] Dunn, W.R. (2002) Practical Design of Safety-Critical Computer Systems. Reliability Press, Solvang, USA.

[3] Hebig, R. (2014) On the Need to Study the Impact of Model Driven Engineering on Software Processes. In Proceedings of 2014 International Conference on Software and System Process, Nanjing, 26-28 May 2014, 164-168. https://doi.org/10.1145/2600821.2600846

[4] Booch, G., Rumbaugh, J. and Jacobson, I. (1999) The Unified Modeling Language User Guide. Addison-Wesley, Boston.

[5] Monin, J.-F. (2003) Understanding Formal Methods, Springer, Berlin. https://doi.org/10.1007/978-1-4471-0043-0

[6] Spivey, J.M. (2001) The Z Notation: A Reference Manual. 2nd Edition, Prentice Hall, Upper Saddle River.

[7] Rosenblum, D.S. (2005) Lightweight Extension Mechanisms for UML. Lecture Notes Advanced Analysis and Design (GS02/4022), London. http://www0.cs.ucl.ac.uk/teaching/syllabi/2006-07/ug/4022.htm

[8] Saaltink, M. (1999) The Z/EVES 2.0 User's Guide, Technical Report TR-99-5493-06a. ORA Canada, One Nicholas Street, Suite 1208. Ottawa, Ontario K1N 7B7

CANADA.

[9] Myers, G.J. (2004) The Art of Software Testing. 2nd Edition, John Wiley & Sons, New York.

[10] Jorgensen, P.C. (2013) Software Testing: A Craftsman's Approach. 4th Edition, Auerbach Publications, Boca Raton.