Open access Journal International Journal of Emerging Trends in Science and Technology

Control Model for Analyzing Erroneous Human Behavior a System Safety Using Model Checker with Mutation Testing Approach

Authors
Manisha Umak[1], Prof. U. A. Jogalekar[2]
[1]Pune University, Smt. Kashibai Navale college of Engineering, Vadgaon (BK), Pune
Email: *Manisha.umak059@email.com*
[2]Pune University, Smt. Kashibai Navale college of Engineering, Vadgaon (BK), Pune
Email: *uajogalekar@sinhgad.edu*

Abstract
Safety critical system always provides the automatic system machine where automation testing become an important part for safety. Automation testing is one way to change the way to operate Safety critical system. This is achieved by changing the nature of the tasks that the operators perform. Safety standard assures the little about design and verification of operating procedures. Safety in coma system is maintained by using model checker and mutation testing model. The model checker is used for validating informal hazard analysis result. The coma monitoring system verifies the erroneous human behavior in coma patient system using model checker in which mutation testing method analyze the human behavior interface with the system and generate the invalid actions. The state diagram of coma patient recognition system use for identify whether the system is in safe state or in unsafe state. Testing is important in order to achieve sufficiently high software quality. Test-suite derived from the specification can only be as good as the specification itself.

Keywords: *Model Checking, Task analysis, System Safety, Human Automation Interaction (HAI), coma, coma Patient Monitoring, Mutation testing, mutants, specification.*

1. Introduction

Automation testing provides the way to operate critical system is in safe state. The complexity of both software and the testing process itself yields the desire for automation. One direction to address this issue is model-based testing. There, test-cases are used to determine whether an implementation is a refinement of an abstract model. Many model-checker based test-case generation methods adhere to this idea. On the other hand, if a requirements specification is available, then testing should concentrate on showing that the implementation is correct with regard to the specification. This approach is integrating the automated test-case generation and specification analysis. This approach is based on mutation of an abstract model and the specification. It is shown that analysis of the mutants that are used for test-case generation can reveal interesting information about the specification For Example. How thoroughly is the specification tested? And How much of the possible behaviors are covered by the specification?

Erroneous human behavior wherever the human operator doesn't follow the normative procedures for interacting with a system, is commonly related to failures in Human Automation interaction surroundings. Human operator will create following mistakes.

- Omission of activities.
- Erroneous repetition of activity.
- Erroneous execution of activity.

Formal systems area unit required to spot human behaviors on system states and handle the errors to create the system safe for the users of the system [1].

Human error are generated and tested on state model to find the unsafe states in the existing system In the proposed solution we will use Mutation based method to generate the Human error test scenarios and used this scenarios to test the state models of the system to find the erroneous behavior. Mutation based test case generation method have been used in many software testing environment method to automate test case generation jobs. Here also, for the case of human errors test suite generation we will use mutation test case generation. Advantage is that mutation test automation will increase the test case coverage of FSM models and it will able to identify more unsafe states compared to existing solution.

2. Literature Survey

2.1 Verifying and Evaluating Human -Automation Interaction

P. Curzon et. al. [6] - Failures in complex systems controlled by human operators can be difficult to anticipate because of unexpected interactions between the elements that compose the system, including human-automation interaction (HAI). HAI analyses would benefit from techniques that support investigating the possible combinations of system conditions and HAIs that might result in failures.

M. L. Bolton et al [2][- Formal Verification Model: Formal verification of program correctness hinges on the use of mathematical logic. A program is a mathematical object with well-defined, although possibly complex and intuitively unfathomable, behavior. Mathematical logic can be used to describe precisely what constitutes correct behavior. This makes it possible to contemplate mathematically establishing that the program behavior conforms to the correctness specification. In most early work this involved constructing a formal proof of correctness. In contradistinction, model checking avoids proofs.

M. L. Bolton et al -Temporal Logics: In view of the difficulties in trying to construct program proofs it seemed like there ought to be a better way. The way was inspired by the use of Temporal Logic (TL), formalism for describing change over time. If a program can be specified in TL, it can be realized as a finite state system. This suggested the idea of model checking-to check if a finite state graph is a model of a TL specification.

2.2 Models for types and levels of Automation

R. Parasuraman et. al. [10] - Technical developments in computer hardware and software now make it possible to introduce automation into virtually all aspects of human machine systems.

- Information Acquisition: Information is gathered and extract the exact information that is required.
- Information Analysis: Data which is required is fully understand and analyze all the details.
- Decision and Action Selection: According to the data and activity need to be performed, the decision and action selection should be done.
- Action Implementation: Appropriate action execution will be done in Action Implementation,

2.3 Task Analytic Models To Visualize Model Checker Counterexamples

M. L. Bolton and E. J. Bass [4]- Model checking may be a form of machine-controlled formal verification that searches a system model's entire state house so as to mathematically prove that the system will or doesn't meet desired properties. Associate degree output of most model checkers may be a counterexample associate degree execution trace illustrating precisely however a specification was profanes. In most of the analysis environments, this output may be a list of the model variables and their values at every step within the execution trace.

A. Blandford, R. Butterworth et. al. [8]- have developed a language for modeling human task behavior and automatic methodology that interpret instantiated models into proper system model enforced within the language of Symbolic Analysis Laboratory(SAL).

2.4 Enhanced Operator Function Model: Syntax and Formal Semantics

L. De Moura et. al. [9] - introduce the hierarchy where in goals decompose into lower level activities and ultimately atomic actions. Further, conditions specify constraints below those activities and actions will execute. Logical operator won't to management what number activities or actions can execute and what temporal relationship exists between them.

2.5 A Knowledge -Based Monitor that Facilitate Incremental Knowledge-Based Development

E. J. Bass, S. T. Ernst-Fortin et. al. [7] - Being able to incrementally outline and take a tool at information basis for intelligent systems is fascinating. However, as a lot of information is superimposed, the information engineer should make sure that unwanted interactions between the present and extra information don't occur. One knowledge- based observance system, Hazard Monitor (HM), provides the power to feature information incrementally. HM's design includes tailorable parts enable the information interactions. Metric linear unit conjointly includes knowledge-base development tools to facilitate initial and progressive knowledge-base development. This paper describes HM's design and information structures and its information-base development tools that facilitate the knowledge engineering method.

2.7 Using a Multi-method Approach to formalize Human to Automation Interaction

Human to Automation Interaction: E. J. Bass et. al. [5] - Human communication processes, including human-human communication and human automation interaction, are important to the operation of safety critical systems but have contributed to failures in domains including aviation, medical. The use of task behavior as part of a larger, formal system model is potentially useful for analyzing such safety-critical systems as the potential ramifications of human behaviors can be verified in relation to other aspects of the system.

Human to Human Communication: E. J. Bass et. al.- are extending EOFM as part of a multi-method approach where analyses via theorem proving and model checking are linked through a top-level XML description of human task behavior. Our model of cooperative systems functioning within specified roles.

2.8 Phenotypical Erroneous Human Behavior

M. L. Bolton, E. J. Bass et. al. [3] - Model-driven design and analysis techniques provide engineers with formal methods tools and techniques capable of evaluating how human behavior can contribute to system failures. This paper presents a unique technique for mechanically generating task analytic models encompassing each normative and human behavior from normative task models. The generated inaccurate behavior is capable of replicating Hollnagel's zero order phenotypes of inaccurate action for omissions, jumps, repetitions, and intrusions. Multiple makeup acts will occur in sequence, therefore providing the generation of upper order phenotypes.

3. Proposed System

While considering the previous paper, which identify erroneous human behavior and avoid that transaction. It also replaces that value with its proper value but yet it does not provide the proper solution that will solve in proposed system with some modification. We are going to apply this proposed solution on coma medication system data. Existing system drawback that will analyze in given proposed system:

A) Scalability: Mutation based testing method increase the system scalability by increasing the state space for identifying human behavior.

B) Does not identify higher order failure: It does not identify complicated erroneous human behavior. In this paper defined the modification that will help to identify complicated human erroneous activity.

However, our new method is capable of generating these types of higher order failures without considering all of the complex combinations of extraneous actions that would be required to generate similarly ordered erroneous behaviors using the technique from. While mutation testing model checker method presented in this paper could be used to explore higher order erroneous behaviors based on attentional failures.

In this paper we are combining the human behavior with mutation testing method which will increase the system scalability by identifying more complex error from the system.

Mutation Testing Method

Mutation testing is a structural testing method aimed at improving the accuracy of the test suit, and estimating the number of faults present in systems under test.

The process, given program P and test suite T, is as follows:

1. We systematically apply mutations to the program P to obtain a sequence P_1, P_2 ,... P_n of mutants of P. Each mutant is derived by applying a single mutation operation to P.
2. We run the test suite T on each of the mutants, T is said to kill mutant P_i if it detects an error.
3. If we kill k out of n mutants the adequacy of T is measured by the quotient k=n. T is mutation adequate if k = n.

The main benefit of mutation testing approach is that it can be almost completely automated. Structural test suites are directed at identifying defects in the code. One goal of mutation testing is to assess or improve the efficiency of test suites in discovering defects [12].

In proposed solution we use the model checker with mutation testing model which increase the state space for identifying the erroneous human behavior which will cause to generate all invalid actions from the system. Formal Model and model checker is a software tool which will help to identify erroneous human behavior. In this we generate the following algorithm:

Algorithm

MODELCHECKER (INPUT, OUTPUT): Preparing the sample input program X.

INPUT: No. of events E_i:={E_1,E_2,…….E_n} to be passed such as E_1:= Set Dose, E_2:= Set Delay, E_n:= Set Limit etc.

OUTPUT: Generate all types of high order errors “Err” from
complex system.
Err := {Err_1,Err_2…………..Err_n}

Step1: Calculate the Equivalent mutants M_t:={X_1,X_2,……X_n} from input program X and Determine transaction states set S:= {Tx_1, Tx_2,……Tx_n}.
Mutants are calculate by using function: M_t := split(X).

Step2: Determine unexecutable transition from existing EOFM.
Unexecutable transition using Existing EOFM= UnExec(Tx_i)/S

Step3: Determine unexecutable transition from mutation Testing method.

Unexecutable transition= UnExec(Tx_i)/S
Provide more scalable result by using mutation based method.

Step4: Generate valid transition test set ‘T’ that satisfies all properties.

Step5: For all valid transition ‘T’:
Compute Mutation score:= Valid mutants(V_m) /total(T).

In Model checker with mutation testing algorithm where we consider that we have system of coma patient. In this system doctor identify the sequence of actions based on their knowledge and that actions is to be passed to the system as input. So number of events should be passed as input to the system and all invalid human actions will generate as output which identify the all actions which are harmful for system. For performing this action we apply mutation testing method

In first step of algorithm calculate the mutants from the program mean we split the program into parts. then in second step identify the all invalid actions performed by user using existing method. . In step 3 generate erroneous behavior by using mutation testing method and then we calculate mutation score which is count as original valid transaction upon total actions performed by user

Architecture

Complex safety critical applications like Automated Medication systems for Coma patients involve both human operators and automated devices. In this environment, human operator can make some error in operation of systems which will affect the safety of patients. In this project, we propose a verification mechanism which automatically generated the erroneous behaviors of humans and test it on the formal models of the system to evaluate the security. With this verification system, human errors can be identified and system can be made more robust to failures.

Figure 1 shows the overview for the functions defined in coma patient recognition system. In figure 1, input is passing the EOFM (Enhanced operator function model) language used for model the operator as input/output system. Inputs may come from several sources including: the human device interface, mission goals, environment, and other human operators. Output variables are human actions. The system consists of 3 modules

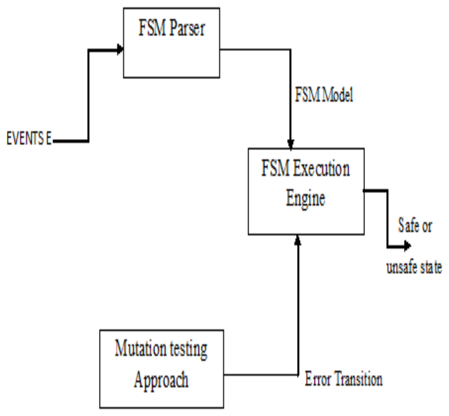


Figure 1: System Architecture

- FSM Parser: This module will create the FSM (Finite State Model) model from the EOFM language specification which consists of state and transition.
- FSM Execution Engine: This module will execute the FSM state machine and generate all possible output states and also summarizes the number of safe state and number of unsafe states.
- Mutation testing Approach: This module will generate the different combination of erroneous human behaviors.

Final outcome shows the system is in safe state or not and if system is not in safe state then we translate that value and convert unsafe state into the safe state.

4. Implementation Details

The implementation of this project is run on jdk 1.6 platform with the coding language in java. For this project, Netbeans IDE 6.9.1 use as a tool in for supporting the java language. Specifically its graphical user representation is done using swings in java. This execute on the Pentium processor with the feature of 1 GB RAM, 20 GB Hard Disk and SVGA Monitor.

5. Result

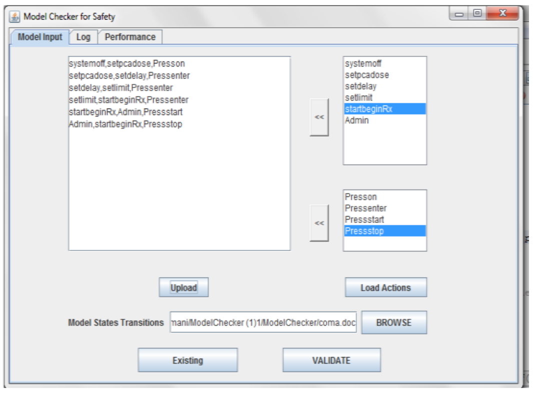
In the proposed system we are using the mutation based testing approach to improve scalability and to find out the complex error from the automated system. The main aim of our project is to maintain the human interaction with the system.

Following snapshot show the graphical user interface of our system where we some actions in two list box.

In list1: system off, set dose, set delay, set limit, admin etc

In list2: press on, press enter, press off, press stop etc

These actions are provided to admin for generating the sequence of flow. Admin selects this action in proper execution sequence. And it pass as a input by clicking on upload button. On main screen we have two options existing and validate where existing button shows the result of existing system and validate button shows the result of proposed method.



```
systemoff,setpcadose,Presson
setpcadose,setdelay,Pressenter
setdelay,setlimit,Pressenter
setlimit,startbeginRx,Pressenter
startbeginRx,Admin,Pressstart
Admin,startbeginRx,Pressstop
```

systemoff
setpcadose
setdelay
setlimit
startbeginRx
Admin

Presson
Pressenter
Pressstart
Pressstop

Model States Transitions: mani/ModelChecker (1)1/ModelChecker/coma.doc

Figure 2: Main gui of the system

When admin click on validate or existing button then it will generate the flow sequence diagram as shown below.

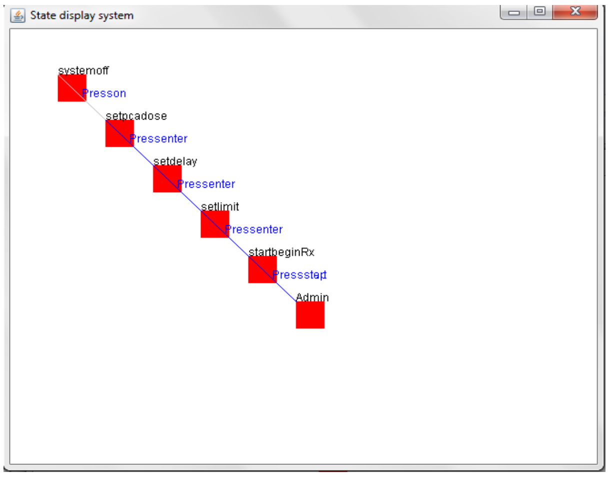
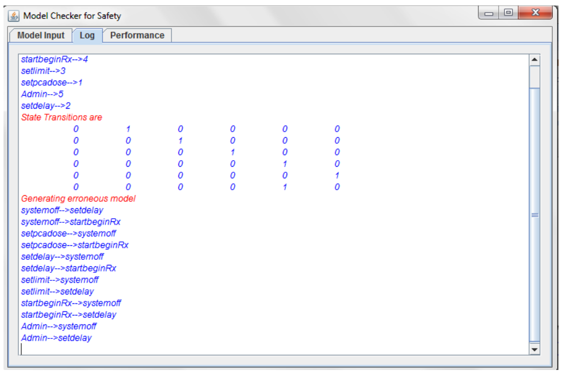


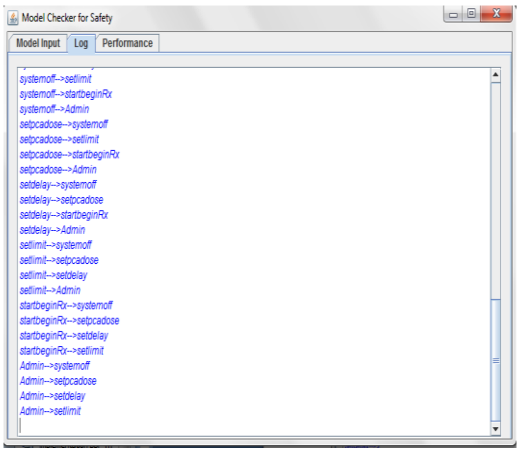
Figure 3: Flow sequence of activity

When we click on the validate button it will generate the state transition table and erroneous actions as the output using the using the mutation testing approach.



```
startbeginRx-->4
setlimit-->3
setpcadose-->1
Admin-->5
setdelay-->2
State Transitions are
          0         1         0         0         0         0
          0         0         1         0         0         0
          0         0         0         1         0         0
          0         0         0         0         1         0
          0         0         0         0         0         1
          0         0         0         0         1         0
Generating erroneous model
systemoff-->setdelay
systemoff-->startbeginRx
setpcadose-->systemoff
setpcadose-->startbeginRx
setdelay-->systemoff
setdelay-->startbeginRx
setlimit-->systemoff
setlimit-->setdelay
startbeginRx-->systemoff
startbeginRx-->setdelay
Admin-->systemoff
Admin-->setdelay
```

Figure 4(1): state transition table and erroneous activities.



```
systemoff-->setlimit
systemoff-->startbeginRx
systemoff-->Admin
setpcadose-->systemoff
setpcadose-->setlimit
setpcadose-->startbeginRx
setpcadose-->Admin
setdelay-->systemoff
setdelay-->setpcadose
setdelay-->startbeginRx
setdelay-->Admin
setlimit-->systemoff
setlimit-->setpcadose
setlimit-->setdelay
setlimit-->Admin
startbeginRx-->systemoff
startbeginRx-->setpcadose
startbeginRx-->setdelay
startbeginRx-->setlimit
Admin-->systemoff
Admin-->setpcadose
Admin-->setdelay
Admin-->setlimit
```

Figure 4(2): state transition table and erroneous activities

In our project we are using the EOFM (Enhanced operator Function Model) method which will provide the formal methods to identify the incomplete transition state and that will generate the erroneous transition within the automated system. But it has some limitations that it does not find complex errors and also it find errors from limited state space. So the state space is increment by using mutation testing approach.

The comparison between this two methods shown in following graph where red rectangle shows proposed system result and blue circle shows existing system result. Fig.3 shows the result using EOFM method and mutation based method. In all plots, the number of visited states is reported on the left *y*-axis and number of erroneous transaction is reported on the right *x*-axis.

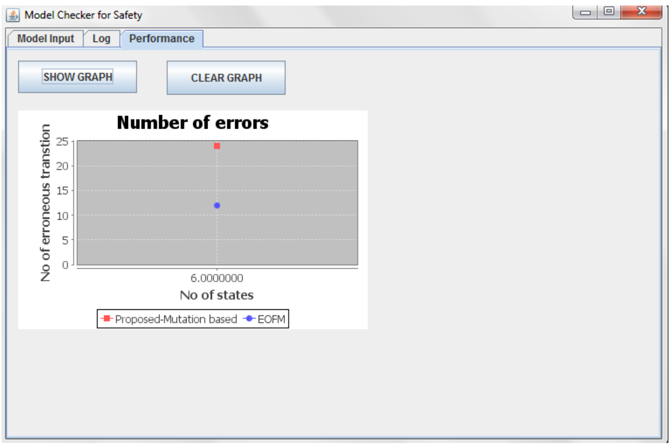


fIg.5: Plot of the verification results (number of erroneous transition and the number of visited states)

Acknowledgement

The authors would like to thank the researchers as well as publishers for making their resources available and teachers for their guidance. We are thankful to the authorities of Savitribai Phule University, Pune and concerned members of cPGCON2015 conference, for their valuable guidelines and support. We are also thankful to the reviewer for their valuable suggestions. We also thank the college authorities for providing the required infrastructure and support. Finally, we would like to extend a heartfelt gratitude to friends and family members.

References

1. Matthew L. Bolton and Ellen J. Bass, *"Generating Erroneous Human Behavior From Strategic Knowledge in Task Models and Evaluating Its Impact on System Safety With Model Checking"*, IEEE Transaction On Systems, Man, And Cybernetics: Systems, Vol. 43, No. 6, November 2013.
2. M. L. Bolton, E. J. Bass, and R. I. Siminiceanu, "*Using formal verification to evaluate human-automation interaction: A review,"*, IEEE Trans. Syst., Man, Cybern. Syst., vol. 43, no. 3, pp. 488-503, May 2013
3. M. L. Bolton, E. J. Bass, and R. I. Siminiceanu, *"Generating Phenotypical erroneous human behavior to evaluate human-automation interaction using model checking,"* Int. J. Human-Comput. Stud., vol. 70, no. 11, pp. 888-906, 2012. 5, pp. 961-976, Sep. 2011.
4. M. L. Bolton and E. J. Bass,*"Using task analytic models to visualize model checker counterexamples,"* in Proc. IEEE Int. Conf. Syst. Man, Cybern., Oct. 2010, pp. 2069-2074.
5. E. J. Bass, M. L. Bolton, K. Feigh, D. Griffith, E. Gunter, W. Mansky, and J. Rushby, *"Toward a multi-method approach to formalizing human automation interaction and human-human communications,"* in Proc. IEEE Int. Conf. Syst., Man, Cybern., Oct. 2011, pp. 1817-1824.
6. P. Curzon, R. Ruksenas, and A. Blandford,*" An approach to formal verification of human-computer interaction,"* Formal Aspects Comput., vol. 19, no. 4, pp. 513-550, 2007.
7. E. J. Bass, S. T. Ernst-Fortin, R. L. Small, and J. Hogans, *"Architecture and development environment of a knowledge-based monitor that facilitate incremental knowledge-based development ,"* IEEE Trans. Syst.,Man, Cybern. A, Syst. Humans, vol. 34, no. 4, pp. 441-449, Jul. 2004.
8. A. Blandford, R. Butterworth, and P. Curzon, *"Models of interactive systems: A case study on programmable user modelling,"* Int. J. Human-Comput. Stud., vol. 60, no. 2, pp. 149-200, 2004.
9. L. De Moura, S. Owre, and N. Shankar, "The SAL language manual," Computer Science Laboratory, SRI International, Menlo Park, Tech. Rep. CSL-01-01, 2003.
10. R. Parasuraman, T. Sheridan, and C. Wickens,*"A model for types and levels of human interaction with automation,"* IEEE Trans. Syst., Man,Cybern. A, Syst., Humans, vol. 30, no. 3, pp. 286-297, May 2000.
11. P. Ammann, P. E. Black, and W. Ding, "Model Checkers in Software Testing. Technical report, National Institute of Standards and Technology, 2002.
12. D. Javaux, "A method for predicting errors when interacting with finite state systems. How implicit learning shapes the user's knowledge of a system," *Reliab. Eng. Syst. Safety*, vol. 75, no. 2, pp. 147–165, 2002.
13. P. Ammann, W. Ding, and D. Xu. Using a Model Checker to Test Safety Properties. In Proceedings of the 7th International Conference on Engineering of Complex Computer Systems, pages 212–221, June 2001.
14. P. Curzon and A. Blandford, "From a formal user model to design rules," in *Design, Specification and Verification of Interactive Systems*. Berlin, Germany: Springer, 2002, pp. 1–15.
15. J. M. Wing, "A specifier's introduction to formal methods," *Computer*, vol. 23, no. 9, pp. 8–23, Sep. 1990.