International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2017 IJSRCSEIT | Volume 2 | Issue 4 | ISSN : 2456-3307

Model-based Integration and System Test Automation for Software Systems

***G. Lavanya, S. Vasundra**

Department of Computer Science JNTU College of Engineering, Anantapurumau, Andhra Pradesh, India

ABSTRACT

The importance of software system is growing rapidly, traditional development system are not suitable to meet their requirements. New cost-efficient tools are needed for software packages. This paper introduces an automatic test generation technique, known as Model-based Integration and System Test Automation (MISTA). This integrated practical and security testing of various software package systems . MISTA generates test code that can be executed immediatetly. It uses a high-level Petri web to capture each control and data-related needs for practical testing, access management testing, or penetration testing with threat models. Once generating test cases from the test model, MISTA converts the test cases into workable test code by mapping model-level components into implementation-level. MISTA has enforced test generators for varied test coverage criteria of test models, code generators for various programming and scripting languages, and test execution environments like Java, C, C++, C#, HTML-Selenium IDE. The effectiveness is evaluated in terms of access-control fault detection rate using mutation analysis of access control implementation.

Keywords : Functional Testing, Model-Based Testing, Security Testing, Software Assurance.

I. INTRODUCTION

The widespread application of web and mobile computing has considerably increasing the dependence on software enabled systems. This dependence raises vital issues concerning computer code dependableness and security as a result of a computer code failure will result in fatal consequences. However, computer code testing may be a labour-intensive activity, which regularly accounts for five hundred or additional of the computer code development prices[11]. To enhance testing productivity and scale back prices, it's extremely fascinating to automatize test generation and execution.

It additionally facilitates fast, economical verification of demand changes and bug fixes, and minimizes human errors. In this present a tool supported technique referred to as Model-based Integration and System check Automation (MISTA), for integrated testing of system functions, access management policies, and security threats[1][2]. It can be also define as Model-Implentation Description. It consists of a test model and Model-Implementation Mapping(MIM). It uses Predicate-Transition (PrT) nets as associate degree expressive formalism for building useful and security test models.

PrT nets are high-level Petri nets, a well-studied formal methodology for modeling and verification of computer code systems[3]. Previous work has additionally demonstrared that PrT nets square measure capable of specifying access management policies and security threats. Because test models specified by PrT nets will capture each knowledge and management flows of test requirements, MISTA will generate complete model-based test cases, as well as specific test inputs and test databases(expected results). Note that model-based test cases are not yet feasible with the System Under Test(SUT), because test models are abstract descriptions of SUT's behaviors. MISTA provides an expressive way for describing the relations between the model-level parts and therefore the implementation level constructs within the target language or test surroundings thus automatically transform the model-level tests into feasible code[4]. After providing test cases from the test model

according to given criteria, test model converts the test scripts into feasible test code by comparing model-level elements into implementation level builds the test code and test trees.

The remaining paper is organized as follows. In section 2 includes the related work of different methodologies. The section 3 introduces the proposed system. The section 4 gives experimental results. The section 5 concludes the paper.

II. RELATED WORK

Dianxiang Xu [6],proposed threat models and practical models are both spoken to by PrT nets. The fundamental linguistic distinction between them is that assault moves in danger models are named after attack .

Zhu and He [8] have projected a strategy for testing abnormal state Petri nets. The philosophy comprises of four testing procedures: move arranged testing, state-situated testing, information flow-arranged testing, and specification-situated test.

The work on test script modify was done by Grechanik[5],in which they immediately find changes between GUI objects and locate test script statements that reference the modified GUI objects.Their tool gives the warnings that enable testers to fix errors in test scripts manually.

Mc Dermott [7] has additionally proposed to model framework and system assaults with customary place and move nets, and make entrance tests as per assault nets. No system was given to produce security tests from attack nets.

T. Mouelhi [9],[10] proposed a concentrated on the testing of part consent assignments and client part assignments in RBAC, where clients, parts, and authorization guidelines are predefined. It additionally naturally produces executable access control tests from the test models

III. PROPOSED WORK

MISTA provides an expressive way for describing the relations between the model-level elements and the implementation-level constructs in the target language or test environment so it automatically transform the model-level tests into executable code. The input to MISTA is called a Model-Implementation Description(MID) consists of model and a Model-Iplementation Mapping(MIM). MISTA uses a high-level petri net to imprison both control and data-related requirements for functional testing, access control testing or penetration testing with threat models. After test cases, MISTA test models converts the test cases into executable test code by mapping model-level elements into implementation-level and contructs the test code and test trees.

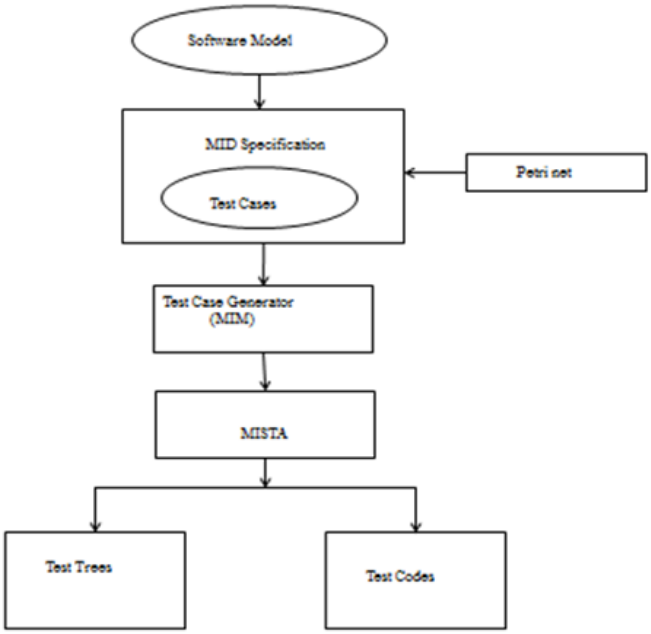

Figure 1: System Architecture of MISTA(Model-based Integrtion and System test Automation For Software Systems).

This section presents the enhanced test scripts model. It consists of three parts,

1. **MID Specification**

MODEL-IMPLEMENTATION ESCRIPTIONS(MID), as the front-end input language for MISTA, lay the foundation of the automated test generation technique in the approach. A MID specification consists of a test model (PrTnet) and a MIM description. The former does not use the implementation details of the SUT, whereas the latter relies on the test model as well as the SUT. First present PrT nets and MIM, and then describe with the examples like Bank account transaction, block Game, Cruise control and self-test.

2. PrT Nets for Test Modeling

A PrT(Predicate Transition) net consists of places (data and conditions), transitions(activities), normal and bidirectional arcs between places and transitions (input and output conditions of activities), inhibitor arcs from places to transitions (negative input conditions), and initial markings (states). A transition can be associated with a guard condition.

A PrT net N is a tuple < P,T,F,I,L,φ,> where the elements are defined as follows.

P - a finite set of places (also called predicates).

T - a finite set of transitions.

F - a finite set of normal arcs from places to transitions and from transitions to places, ie.

F ⊆ P x T ∪ T x P, .

I - a finite set of inhibitor arcs from places to transitions.

L - a labeling function on arcs F ∪ I. L(f) is the label for arc f ∈ F ∪ I When the label of an arc is not specified, the default label is a no-argument < >.

Φ - a guard function on T . The guard condition of transition t, φ(t), is a first-order logical formula, which can evaluate true or false.

Multiple initial markings (states) are often related to identical internet structure. Suppose,

$$M_o^k = \bigcup_{p \in P} M_0^k(p) \quad \text{-------- Eq1}$$

Equation 1, Defines Mk0(p) is that the set of tokens residing in P. A token in p can be a tuple of ground terms<X_1,…….,X_n>. we tend to denote it as p(X1,…….,Xn). For a zero-argument token tend to denote it as p. The tokens in an initial marking represent take a look at information or system settings (e.g., choices and preferences).

consider handcart system is an example, token product and token amount represent the quality. A transition could also be related to a listing of variables as formal parameters. These variables usually seem within the connected arc labels.

Fig.1 shows a straight forward PrT internet, wherever holding, clear, on, and handempty square measure places (circles) and stack(x, y) may be a transition (a rectangle). The guard condition of stack(x, y) is x!=y (it is encircled in brackets). An arrow (e.g., from holding to stack) represents a standard arc; a line phase with a tiny low circle (e.g., from handempty to stack) represents a matter arc.

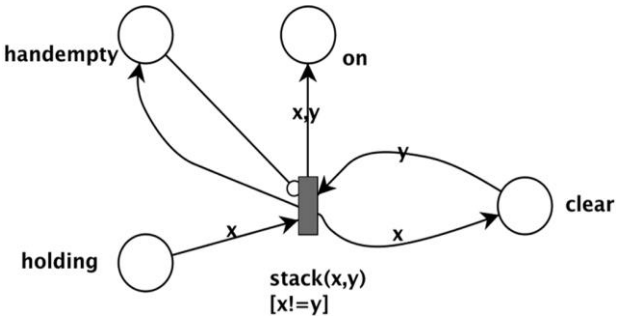

Fig : 1 A simple PrT(Predicate Transition) net.

3. Model-Implementation Mapping

MIM description is used to generate code by mapping the elements in a test model to the implementation constructs based on the System Under Test(SUT) programming interface. The generated code can be executed with the SUT.

A MIM specification consists of 7-tuple , it can measure as follows.

$$MIM=<ID,f_0,f_c,f_a,f_m,f_s,h> \quad \text{---------- Eq2,}$$

1) ID is that the identity of the SUT take a look at against the test model.

2)$f_0 : O_M \longrightarrow O_I$ - the object function that maps the objects in the test model to the objects in the SUT. Given an object x in the test model, $f_0(x)$ is an object in the SUT.

3)$f_c : T \longrightarrow CODE_I$ - the element (or method) mapping function that maps transitions (component calls) with in the PrT net to code blocks (test operations) in the SUT.

4) $f_a : P \longrightarrow CODE_I$ - the accessor function that maps the places in the PrT net to code blocks (called accessor) in the SUT. An accessoris typically a sequence of assertions that scan and check system states.

5) $f_m : t \longrightarrow CODE_I$ - the mutator function that maps the places in the PrT net to code blocks (called mutators) in the SUT. A mutator may be a piece of code which will modification system states.

6) $l_s \subseteq P$ - a list of places in the PrT net that are implemented as system settings in the SUT. These places are referred to as setting predicates.

7) h - the helper code function that defines user-provided code to be enclosed in the test code.

IV. EXPERIMENTAL RESULTS

The results of our experiments are summarized based on two Java applications are Banking account (BA) and Login Validate (LV) with different parameters as

shown in below table. In the below table Define T is the total number of transitions, P is the total number of places (they reflect the complexity of test models), TC is the number of test cases, LOC is the number lines of code generated, M is the total number of mutants, K is number of mutants killed by the test and FDR is the fault detection rate (number of mutants killed).

	Models		Tests		Mutation Analysis		
	T	P	TC	LOC	M	K	FDR
BA	73	27	207	3086	243	233	95.9%
LV	126	30	179	4680	914	914	100%

Table 1 : Results for Banking account and Login Validate

For BA, 207 test cases in 3,086 lines of non-comment code were generated. They killed 233 out of 243 mutants, with an overall detection rate of 95.9%. The 10 remaining mutants not killed by the tests having some adding-rule operator but can never cause any security issues because the functional precondition of the activity in the added rule is not satisfiable. In LV,179 tests in 4,680 line of code were generated. They killed all of the 914 mutants.

Figure1, shows total number of pass and fail tests in application. when the testing process advances, it gets more time consuming to discover extra faults since increasing and more tests should be made and executed.

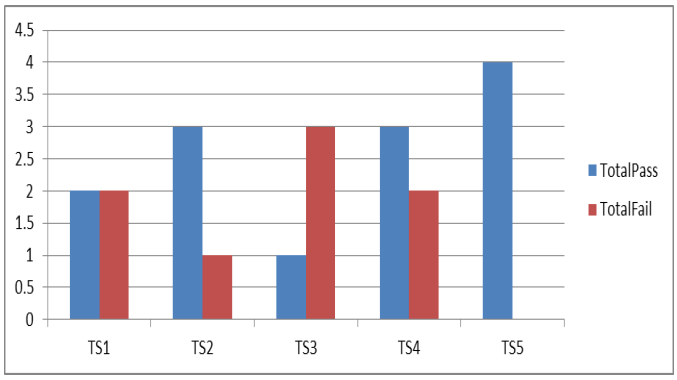

Figure 1 : Total number of pass and fail test cases

Figure 2 shows cost effectiveness of test cases. When number of failed test cases are more than passed test cases, cost increases. When number of failed test cases are less than passed test cases, cost decreases.

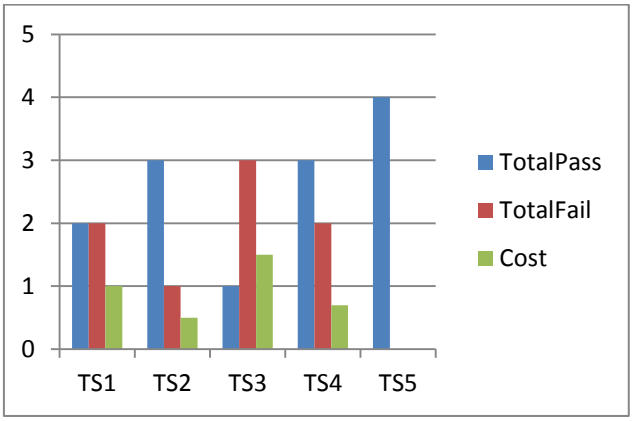

Figure 2 : Cost effectiveness of test cases.

IV. CONCLUSION

The main contribution of this paper introduces a technique for integrated model-based testing of system functions, access management policies, and security threats. This technique will generate executable tests with reference to a spread of coverage criteria of test models represented by Predicate Transition nets. It additionally supports variety of programming languages, and test execution framework. It is simple to introduce a new test generator, target language, or test execution environment. The methodologies proposed address the effects on testing scope and profitability and minimize cost and time of testing.

V. REFERENCES

[1]. M. Utting and B. Legeard, "Practical Model-Based Testing: A Tools Approach. San Francisco", CA, USA: Morgan Kaufmann, 2006.

[2]. Michael Grottke, Dong Seong Kim, Rajesh Mansharamani, Manoj Nambiar, Roberto Natella, Kishor S. Trivedi, "Recovery From Software Failures Caused By Mandelbugs", IEEE 2015

[3]. K. Jensen, Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. New York, NY, USA: Springer-Verlag, 1992, vol. 26.

[4]. T. Murata, "Petri nets: Properties, investigation and applications," Proc. IEEE, vol. 77, no. 4, pp. 541-580, Apr. 1989.

[5]. M. Grechanik, Q. Xie, and C. Fu, "Maintaining and Evolving GUI-directed Test Scripts," in

Proceedings of the International Conference on Software Engineering (ICSE'09), 2009, pp. 408-418.

[6]. Dianxiang Xu, Senior Member, IEEE, Michael Kent, Lijo Thomas,TejeddineMouelhi, and Yves Le Traon, Automated Model-Based Testing of Role-Based Access Control Using Predicate/Transition Nets, Transactions on Computers, Vol. 64, NOo. 9, September 2015.

[7]. K. H. Mortensen, Automatic code generation method based on coloured Petri net models applied on an access control system, in Application and Theory of PetrNets.NewYork,NY,USA: Springer-Verlag, 2000, pp. 367-386.

[8]. Zhu and X. He, A methodology for testing high-level Petri nets, Inf. Softw. Technol., vol. 44, pp. 473-489, 2002.

[9]. Pretschner, Y. L. Traon, and T. Mouelhi, Model-based tests for access control policies, in Proc. 1st Int. Conf. Software Testing Verification and Validation (ICST'08), Lillehamer, Norway, Apr. 2008.

[10]. Briand, L.C, Di Penta, M., Labiche, Y. Assessing and improving state based class testing: A series of experiments, IEEE Trans. on Software Engineering, vol. 30, no. 11, pp. 770-793, Nov. 2004.

[11]. Dr. S. Vasundra , K.Hussenvalli1, Software System to Model Preferences of Multiple Users", International Journal of Advance Research in Computer and Communication Engineering,", Volume3, Issue 11, November 2014.