ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 3, Special Issue 2, March 2016

Automated Test Code Generation Technique to Improve Software Quality

S.Sujitha#1, V.M.Priyadharshini*2,

1M.E (Software Engineering), Anna University, Trichy
sujithacse1992@gmail.com
2Assistant Professor-CSE/IT dept., Anna University, Trichy
priyadharshinivm@gmail.com

Abstract— The application of internet and mobile computing has increased our dependence on software systems. The major concerns for software systems are reliability and security. To increase security for the system, Model based Integration and System Test Automation (MISTA) has been used. It uses high level petrinet model to get data requirements from functional testing, access control testing and penetration testing. First MISTA generates test cases from test model and then converts test cases into executable test code. The MIM specification converts the elements of the test model to the implementation level constructs. MISTA implements test generators for different test coverage criteria and code generators for different programming languages such as Java, C, C++, HTML etc. MISTA has been applied to both functional and security testing of software system and it also used to analyse the time dependency between the fault detection and correction process.

Keywords— Petrinet; functional testing; security testing; software assurance; model based testing

1. Introduction

Software quality assurance is important for software testing. It aims at finding errors by executing a program with test cases. Automated software testing has been used because it will reduce cost and improve testing productivity. The software testing is a labor intensive activity and it is expensive. For improving testing productivity and reduce costs, it is important to automate test generation and execution. It also generates quick, efficient and verification of requirement changes and bug fixes and reduces the human error. The major means for improving the quality of the software system is to test the system to find potential failures.

Model based testing (MBT) generates and executes test cases using behavior models by System under test (SUT) [2]. It can be very effective in fault detection. The main work of our approach is to gather and integrate functional and security testing. The access control tests can be executed from test models and it also produces the interactions done in access control model [3]. By applying the fault detection technique in MBT, the access control model uses the mutation method. Mutation method is used for analyse the effectiveness for the techniques of software testing. During test execution the failure can be detected is a called mutant.

In existing technique, MBT technique focuses only on functional testing. It will apply directly to functional testing not for security testing. The MBT technique doesn't completely generate the automatic test execution because of two reasons: i) the model generates test are not complete because the parameters cannot be specified directly. ii) Doesn't immediately executes the test obtained from test model because the test model use different programming languages.

In our proposed technique, we use a new tool technique called Model based Integration and System Test Automation (MISTA) for generating test code from a Model Implementation Description (MID). It will integrate the functional and security testing. MID contains the Model Implementation Mapping (MIM) and test model. The test models are functional model, access control model and security model. It uses the high level petrinet model for verifying the software system. Test models designed by the petrinet can integrate both data and control flow of test models. MISTA can generate automatic model based test cases, including the test inputs and expected test results. MISTA shows the relations of test models and implementation level for the test environment. It will automatically help to generate test code from test model.

The remainder of this paper is structured as follows: Background and related works are explained in Section II; Automated test code generation and its architecture can be presented in Section III; Generation of model based tests and example is described in Section IV; Conclusion and future work for the technique is discussed in Section V.

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 3, Special Issue 2, March 2016

2. Background

We use MISTA tool for functional and security testing. D.Xu et al.[4] proposed a Threat Model-Implementation (TMID) approach to automated generation of security tests by using formal threat models that can be denoted as Predicate/Transition nets. This model generates attack paths. A formal threat driven approach of security threats was described by D.Xu et al.[5] that acts as the mediator between security goals and applications of the security features. Y.L.Traon et al.[6] provides a test driven methodology and policy decision point to analyse the flexibility of the system. The security policy modification can helps to changes in the test code for flexibility. The property is defined as the degree of coupling in between access control logic and business logic in the system. H.Zhu et al.[7] presents a new method for test the software system depend on high level petrinets. For that approach use four testing technique called state oriented testing, flow oriented testing, transition oriented testing and specification oriented testing. All techniques are used a set of schemas for analyse and generate a testing results and various coverage criteria. J.Desel et al.[8] discussed a new concept called cause effect graphing for generates the test cases and the test code. The high level petrinet acts as a intermediate level. The policy used for generating test from the finite state model was proposed by A.masood et al. [9] and evaluated the Role Based Access Control (RBAC) policy. The test suite generated from this model is suitable for fault detection. To avoid fault and increase security policy W.Mallouli et al.[10] described a technique called extended finite state machine.

To generate test cases Alexander et al.[11] defined the model based approach. Jacques et al.[12] discussed the approach to add security testing with the functional testing by using language expression in model based approach. In a modular technology, H.Huang et al.[13] specified and verified the security policies of the system used the colored petrinet process. The security policy of the module is considered as very flexible. Mortensen [14] specified the colored petrinet model to analyse the access control system to generate the test code. The main characteristics are that model is focus only on access control model, but not for the intermediation between the access control model and functional model. In our approach, the access control model integrates with the functional model. The security testing used the attack trees requires mostly the manual work for convert the attack tree into security test. In our work, to find the fault occur in software using model based testing.

3. Automated Code Generation

3.1 Architecture

. Fig.1 shows the architecture of MISTA. The input to the MISTA is MID specification, which includes the test model and the MIM specification. The test model denotes the petrinet model that contains functional model, access control model, and the threat model. The functional model specifies the function in the system, the access control model describes the constraints on the system and the threat model shows the security policy in the system. The MIM specification converts the test model to implementation constraints. MISTA uses different languages such as C, C++, HTML etc to generate test code. It supports a various coverage criteria for test case generation. MISTA is also very effective in the software fault detection.

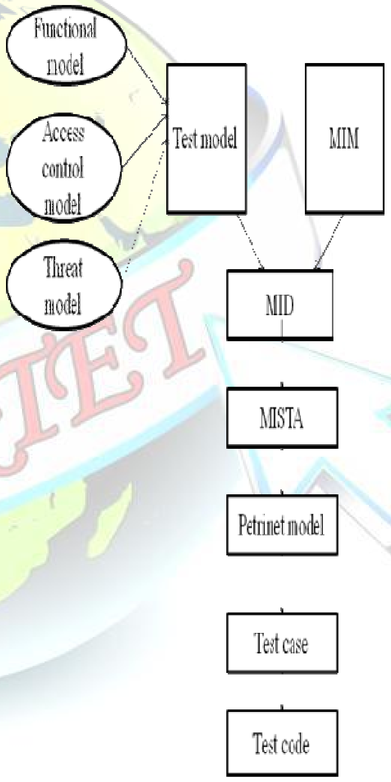


Fig.1: Architecture diagram of MISTA

3.2 Petrinet model

PrT nets are also known as high-level Petri nets. The previous work has also explained that PrT nets are able to specifying access control methods and security threats. Because the test models specified by PrT nets can integrate both data and control flows of test requirements. MISTA can generate automatic model based test cases, including the test inputs and the expected test results. The generation of test model from the PrT net reduces the derivation of valid and invalid test cases. The petrinet is the directed bipartite graph, in which the nodes represent places and transitions. The directed arcs describe in the petrinet model in which places describes are pre or post conditions for the transitions. It is also known as place/transition net. It is one of the developed mathematical modeling languages for the specification of distributed systems [1].

A petrinet has five tuples : <R, S, T, U, L1>

Where, R - finite set of places
S - finite set of transitions
T - finite set of normal arc
U - finite set of inhibitor arc
L1 - set of initial markings

3.3 Model Implementation Mapping (MIM)

The MIM specification mapping the inputs of the test model to the implementation level execution. The goal of model-based testing is to check whether an implementation of a software system relates to the model of that system. The requirements and the test models checks by the MIM specification in the implementation stage. By automatically generate the test cases and expected results from the specification of the system, it requires a formal specification. In some cases the formal specification is also executable by the expected results obtained by executing the specified specification with the test inputs. The setting predicate denoted an input condition of the component that should be configured correctly before called by the component. For example, the test generation component of MISTA requires that the coverage criterion be set before it is invoked. This can be occurring by calling the mutator function of the predicate coverage.

The MIM specification obtain the elements from the petrinet model< R, S, T, U, L1> used in the target language to the programming languages. MIM consists of identity of the system, list of hidden predicates in the test model and the mapping elements. The helper code 'h' is used to generate the test code. It contains the header, setup methods and generated code.

3.4 Model Implementation Description (MID)

The input to the MISTA is called a Model-Implementation Description (MID) and it consists of a test model and a Model-Implementation Mapping (MIM). MID is the front end language for MISTA, and gives the basics for the automated test generation approach. The MIM specification mapping the input of the test model to the implementation level constructs. For the MID technique, the test code can be generated by MISTA for the target languages such as Java, C#, C, C++, HTML etc based on the various coverage criterion of the test model such as reachability coverage, state coverage, transition coverage, depth coverage, and goal coverage. During the development stage, we have applied MISTA to the functional testing to find many problems occur in the system. This MISTA technique has proved that it is very effective in fault detection of the systems.

3.5 Test Code

The target language used in the transition tree is used to generate the test code. MISTA generates the test code from the abstract test from MIM specification [20]. The test code generated in the Selenium IDE can be automatically executed. Helper code denotes to the test code that helps to the tester to generate the executable test code. Test code generation is to convert the transition tree to generate the test code according to the MIM specification and the helper code. The system under test immediately generates and executes the test code. The generated test code is in the form of different target languages from a given input transition tree. Various languages can acts as a input to generate the test code. For example, Jfcunit is an extension for JUnit for GUI testing of the Java programs.

4. Generation of Model Based Tests

The test case can be automatically generated from the model by using model based tests. To analyse software testing the MBT is execute the design from models[17]. Test models represent the behaviour and functions of system and also express the test environment. The existing model based tools focus less on the automatic test code generation. The method calling and execution can be done in the execution of a program [16]. Model Based Testing (MBT) generates test cases and verified the relations between the models and system under test (SUT) by make use of test models of a system under test. The modeling process of MBT helps clarify test requirements. Access control policies are well constraints on system functionality, whereas security attacks are often unexpected behaviors that violate security requirements such as confidentiality, integrity, and availability. Threat modeling can acts as efficient for security testing because threat models describes how security threats attack or damage the system .

4.1 Fault Detection

The software faults defines a defect occur in the system. MBT can be considered as effective in fault detection in the software system because of the automatic generation and execution of many test models. The successful testing is it will recover all testing error in the software. First we will assume priority to the valid test cases and we select the valid test cases based upon their priority value [18]. The code coverage value is also based upon the priority value. The interactions occur in the test makes complexity and increase the fault detection.

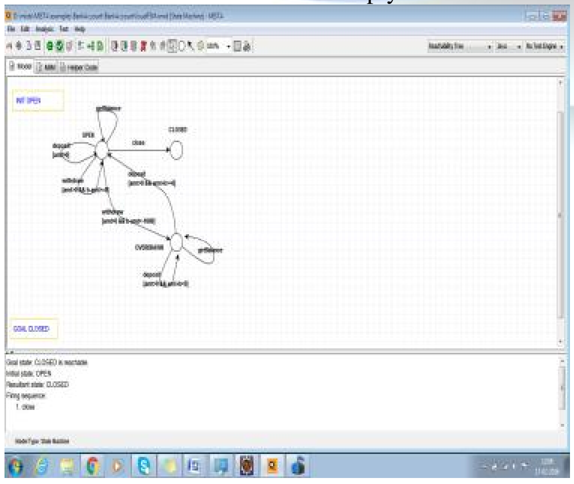
Fault correction is also a difficult and a time dependency process [19]. The performance can be calculated based on the fault detection and correction in the test code. The evaluation method is used to increase the reliability. The fault detection can be calculated by,

$$f(d) = \int_0^t \gamma d(s)\,dt$$

4.2 Running Example

The running example called the bank account is in the form of petrinet model. Three operations can be done: open, close and overdrawn. It is implementation independent. The banking system consists of bank account and bank details. Bank details denote the customer will create, access and close their accounts. Bank account can be used to know the balance amount, deposit and their withdrawal. Bank account can be used to analyse the open and close operations. Many states are reached from the initial goal state. It can be written as {withdraw [amt>0&&b_amt<0]},{ withdraw[amt>0&&b_amt>=0]}.

The time taken for code generation for bank account is 0.5 seconds. The breadth first search strategy can be used and it reaches the maximum search depth is 100. The hidden event or condition can be analysed in MIM specification. For example, { ontable (3), ontable(6), clear(2), clear(5), on(2,3) , on(5,6)} denotes the account 2 and 3are in the table, account 5 is on block 6, account 2 and 5 are clear and the account become empty.



Goal state: CLOSED is reachable
Initial state: OPEN
Resultant state: CLOSED
Firing sequence:
1. close

Fig. 2: Bank account example

5. Conclusion

The systems have presented a technique for automated generation and execution of functional and security tests from a test model including with the mapping from the elements of the model to the implementation constructs. The mapping makes it feasible to convert the model level tests into the executable form of test code. MISTA technique is very efficient and effective for generating test cases and test code. The main advantage is that the technique can integrates system functions, access control policies and security model. This technique can generate executable test code and detect the fault occur in the system. Due to the technical architecture, this technique is easy to introduce a new test generator, target language and test execution environment. The possible approach is to use PrT nets for associating the transitions with time intervals same as in Time petrinets. In future work, introduce notations for real world systems and compare the fault detection with various coverage criteria.

References

[1] T. Murata, “Petri nets: Properties, analysis and applications,” Proc. IEEE, vol. 77, no. 4, pp. 541–580, Apr. 1989.
[2] D. Xu, “A tool for automated test code generation from high-level Petrinets,” in Proc. 32nd Int. Conf. Applicat. and Theory of Petri Nets and Concurrency (Petri Nets 2011), LNCS 6709, Springer-Verlag, Berlin, Heidelberg, Germany, Newcastle, U.K., Jun. 2011, pp. 308–317.
[3] D. Xu, L. Thomas, M. Kent, T. Mouelhi, and Y. Le Traon, “A model-based approach to automated testing of access control policies,” in Proc. 17th ACM Symp. Access Control Models and Technologies (SACMAT'12), Newark, NJ, USA, Jun. 2012.
[4] D. Xu, M. Tu, M. Sanford, L. Thomas, D. Woodraska, and W. Xu, “Automated security test generation with formal threat models,” IEEE Trans. Depend. Secure Comput., vol. 9, no. 4, pp. 525–539, Jul./Aug. 2012.
[5] D. Xu and K. E. Nygard, “Threat-driven modeling and verification of secure software using aspect-oriented Petri nets,” IEEE Trans. Softw Eng., vol. 32, no. 4, pp. 265–278, Apr. 2006.
[6] Y. L. Traon, T. Mouelhi, A. Pretschner, and B. Baudry, “Test-driven assessment of access control in legacy applications,” in Proc. 1st IEEE Int. Conf. Software, Testing, Verification and Validation (ICST'08), Norway, 2008, pp. 238–247.
[7]] H. Zhu and X. He, “A methodology for testing high-level Petri nets,” Inf. Softw. Technol., vol. 44, pp. 473–489, 2002.
[8] J. Desel, A. Oberweis, T. Zimmer, and G. Zimmermann, “Validation of information system models: Petri nets and test case generation,” Proc. SMC'97, pp. 3401–3406, 1997.
[9] A. Masood, R. Bhatti, A. Ghafoor, and A. Mathur, “Scalable and effective test generation for role-based access control systems,” IEEE Trans. Softw. Eng., vol. 35, no. 5, pp. 654–668, 2009.
[10] W. Mallouli, J.M. Orset, A. Cavalli, N. Cuppens, and F. Cuppens, “A formal approach for testing security rules,” in Proc. 12th ACM

Symp. Access Control Models and Technologies, 2007, pp. 127–132.

[11] A. 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.

[12] J. Julliand, P. A. Masson, and R. Tissot, “Generating security tests in addition to functional tests,” in Proc. 3rd Int. Workshop Automation of Software Test, 2008, pp. 41–44.

[13] H. Huang and H. Kirchner, “Formal specification and verification of modular security policy based on colored Petri nets,” IEEE Trans. Depend. Secure Comput., vol. 8, no. 6, pp. 852–865, Nov./Dec. 2011.

[14] K. H. Mortensen, “Automatic code generation method based on coloured Petri net models applied on an access control system,” in Application and Theory of Petri Nets. New York, NY, USA: Springer-Verlag, 2000, pp. 367–386.

[15] A. Marback, H. Do, K. He, S. Kondamarri, and D. Xu, “A threat model-based approach to security testing,” in Software: Practice and Experience, Expanded Version of the AST'09Workshop Paper, Feb. 2013, vol. 43, pp. 241–258.

[16] L. Wang, W. Wong, and D. Xu, “A threat model driven approach for security testing,” in Proc. 3rd Int. Workshop Software Eng. for Secure Syst. (SESS'07), May 2007.

[17] C.Chang, X.Huang, D.Xu, Y,Lai, “An Uml behavior diagram based automatic testing approach,” in IEEE 37th Annual comp.software and appl.conference Workshops 2013.

[18] S.Chaturvedi, A.Kulothungan, “Improving fault detection capability using coverage based analysis,” in IOSR Journal of Computer Engineering (IOSR-JCE) e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 16, Issue 2, Ver. VI (Mar-Apr. 2014), PP 22-30.

[19] Y.P.Wu, Q.P.Hu, M.Xie, S.H.Ng, “Modeling and analysis of software fault detection and correction process by considering time dependency,” in IEEE Trans. on Reliability, Vol. 56, No. 4, Dec 2007.

[20] D.Xu, Weifeng Xu, Manghui Tu, “Automated generation of integration test sequences from logical contracts,” in IEEE 38th Annual International Computers, Software and Applications Conference Workshops 2014.