Asian Journal of Information Technology 15 (16): 2846-2850, 2016
ISSN: 1682-3915

A Strategic Aproach with Automated Safe Composable Model Testing Using Petrinets for Safety Critical System

[1]Smitha and [2]Sankar Ram
[1]Department of Computer Science and Engineering,
Velammal Engineering College, Anna University, Tamil Nadu, India
[2]Department of Computer Science and Engineering,
RMK College of Engineering and Tech, Anna University, Tamil Nadu, India

Abstract: Testing with vast requirements is a challenging task especially with interoperability system. Traditional testing approaches to safety critical distributed system has to be rethought as most of the approaches are based on testing from extensive requirements which is very time consuming. The idea proposed here is to design an automated safe composable testing model with compatible and composable components by designing with petrinets and hence into test sequences. Petrinets addresses issues concerning parallel system with automatic conversion and hence reducing time and cost.

Key words: Petrinets, test sequence, composable, safe, component

INTRODUCTION

Software testing plays a crucial role in the development of quality software. Testing cost occupy about 30-50% of software development (Beizer, 1990). Automated test case generation has a strong impact on testing process (Anand *et al*., 2013) which reduces the cost of testing effort. Testing of safety critical system provides many challenges as there should be zero tolerance in propogation of errors. The key aspect of testing include minimizing time and maximizing the chance of identifying more defects. Model Driven Software Development (MDD) is a new paradigm which reduces effort and time in software development process and improves product quality. Model driven testing (Utting and Legeard, 2006; Mussa *et al*., 2009) reduces testing time for safety critical system by testing composable components which are compatible, instead of testing with requirements. Component C1 can be considered as a quadraple of:

$$C1 = (F, I, P, Ta)$$

Where:

C1 = The component
F = The set of function in the component
I = The set of input/output
P = The set of parameter passed to the component
Ta = The activation time (Smitha and Sankarram, 2014)

$$C = <(f1, f2, ..., fi), I/O, (p1, p2, ..., pj), (Ta)>$$

Component based model reduces the requirements for extensive development process that have to be followed for safety system. Composability is the ability to select and assemble, combine and recombine, configure and reconfigure simulations reusable components (Rozenburg and Engelfriet, 1998). Two components are said to be compatible when they are able to work in the same environment. Components, therefore, need to be tested to determine if they are safe in the context into which they will be deployed. Each reuse of a component will require a reassessment of the suitability and safety of the component and if previous testing of the component did not cover all the circumstances of the new context, further testing must be performed (Wilkinson *et al*., 2014). The software components are self-checked based on the criticality of the information.

The key research question raised here is:

- How to simulate petrinets of a component model
- How test sequences can be generated from a composable model
- How to automate test sequences and generate code from composable model

The answer to first research question is the development of petrinets from components. Petri nets are state-transition systems that extend a class of nets called elementary nets.

Corresponding Author: Smitha, Department of Computer Science and Engineering, Velammal Engineering College, Anna University, Tamil Nadu, India

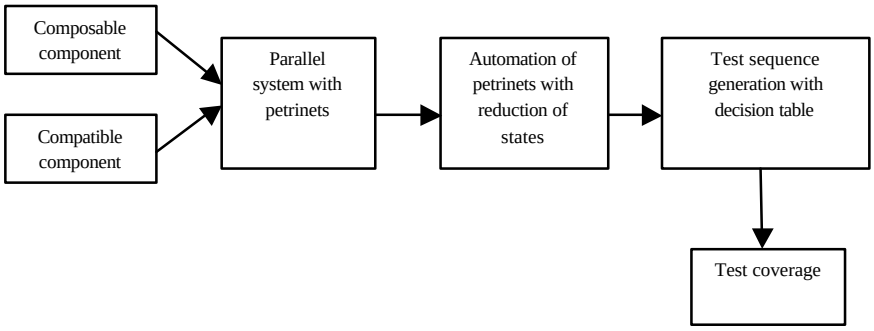

Fig. 1: Safe Composable Testing Model (SCTMP)

Definition 1: A petrinet is a triple where:

- P and T are disjoint finite sets of places and transitions, respectively
- F N (P X T U T X P) is a set of arcs (or flow relations)

Definition 2: Given a net N = (P, T, F), a configuration is a set C so that C⊆P (Rozenburg and Engelfriet, 1998). The 1987 volume contains the most comprehensive bibliography of petrinets (Drees *et al*., 1986) listing 2074 entries. Model based testing (Ravi and Kailash, 2008; Linzhang *et al*., 2004) takes UML (Unified Modeling Language) which is a proclaimed standard by OMG and finite state machine facilitates the construction of behavioral models early in the development of the lifecycle, thus exposing ambiguities in the specification and design.

The answer to second and third research question of how to automate test generation specifies the use of a tool named CPN which automate the test sequence from Petrinets. For a software system specified as a state machine, a test generator may attempt to generate test cases that execute all the state transitions of the state-machine model (Braione *et al*., 2014). Bertolino (2003) articulated that test case generation is a most challenging and an extensively researched activity. A test sequence is a high level test where a sequence of tasks or operations is directly generated from a high level behavioral model according to a particular test objective (Farooq *et al*., 2008). Test generation can be done either manually or automatically. Recent research focus is towards automation of test generation because of the time constraints for testing. Subsequently coding is generated from petrinets with coverage analysis of 100%.

MATERIALS AND METHODS

Safe Composable Testing Model with Petrinet (SCTMP): In order to reduce testing time and to increase the efficiency of safety critical system, SCTMP framework is developed as in Fig. 1. Parallel system is designed with Petrinets which is composable and compatible and thus safety is ensured through reachability, boundedness and liveness property. Automated test sequence is generated with decision table and coverage analysis is done with function coverage.

Example: anti collision device for train protection system on indian railways considering distributed links: The Anti Collision Device (ACD) is a form of Automatic Train Protection invented by and used on Indian Railways. The ACD Network is a Train Collision Prevention system invented by Rajaram Bojji and patented by Konkan Railway Corporation Limited (A Public Sector Undertaking of Ministry of Railways, Government of India). This example specifies a signaling device when two trains are on the same track to avoid accident through GPS (Global Positioning System) and MicroController. The ACD (AntiCollisionDevice), built as a first-of-its-kind system in the world, works with the help of Global Positioning Systems (GPS) installed trains, railway stations and at key points on the tracks. The ACDs constantly read the exact position of the trains and communicate with each other through the GPS. The GPS data is fed to the microcontroller. If two trains happen to be moving on the same track within a Distance (D) of 3 km of each other, the microcontroller on receiving the GPS data and after identifying that both trains are on the same track activates the automatic braking system making the trains to come to a halt. Figure 2 explains anti collision device with multiple trains running parallel on different tracks namely Track 1-3 with Trains T1-T6.

Design of pertrinets: Petrinets are designed for parallel system for anti collision device with trains T1-T4 representing objects. These objects when receiving GPS signal and calculating Distance (D) with the ability to stop the trains on opposite track will be in safe state otherwise it is unsafe as in Fig. 3.

Algorithm for SCTMP

Algorithm for verification: Identify the components from requirement specification:

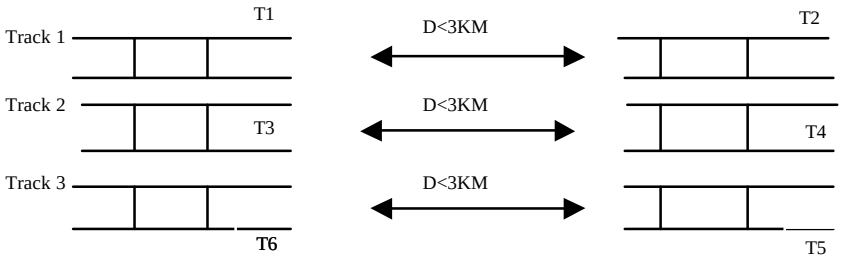

Fig. 2: ACD demonstrated as parallel system

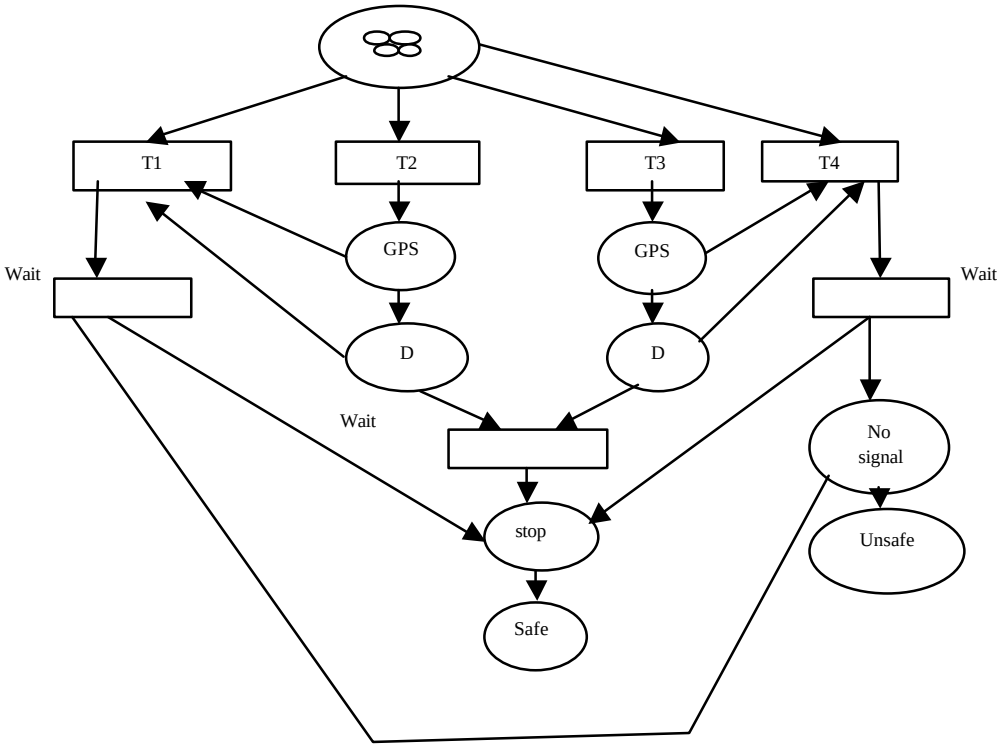

Fig. 3: Petrinet model

with common functionalities related to hardware and software
Int i, j, a[], b[], testcase [], n
//I,j represents the iteration through components
//n is the number of components
For (i = 1 to n)
J=2
Input the no. Of components n
Compare (A[i] with b[j]) where a[i] and b[j] are components
If (a[i] and b[j]) is dependent related to interfaces, o/p, control, function call
Compose (a[i] and b[j])
Check for compatible(a[i] and b[j]) with external environment
Form a network with (a[i] and b[j])
Repeat the above process until n
Let s1,s2....sn represents the states of components
S[n+1]=safe,s[n+2]=unsafe
Int Time t=0;
If (t) connectwithcondition(s1, s2)
Generate Petrinets with safe and unsafe state
Testcase[50];
for each testcase[i]; treetraversal(a[i])
If (a[i] = safe) then verify=1;count++
Else
Verify = 0
If (count >=40) then
Validate (s1, s2, ..., sn)

Algorithm for Validation(reducing time for testing)
Automate(s1,s2....sn)
Automate test sequence(s1,s2...sn)
Generate code coverage
The above algorithm specifies the identification of components from requirements and forming a network with compatible components. The network is designed for petrinet with properties being checked for safe and unsafe state. Automation is done for speeding up the process with test sequence generation and subsequently testing for code coverage.

RESULTS AND DISCUSSION

Generation of test sequence: Test sequence generation represents the dynamic behavior of objects with message passing as illustrated in Fig. 4.

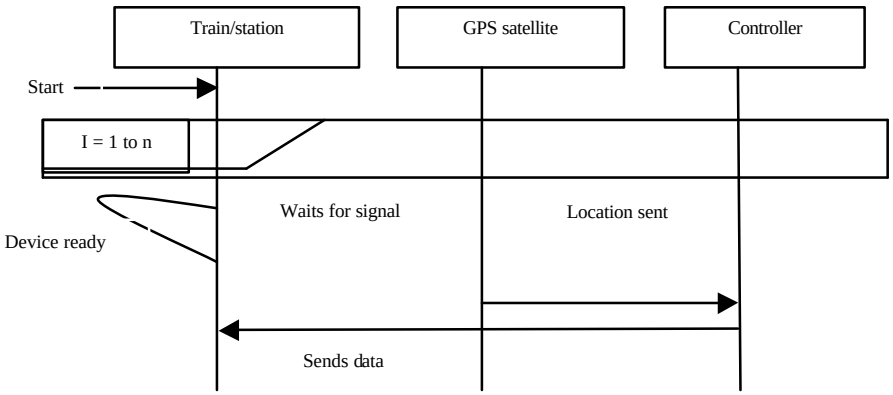

Fig. 4: Test sequence for ACD

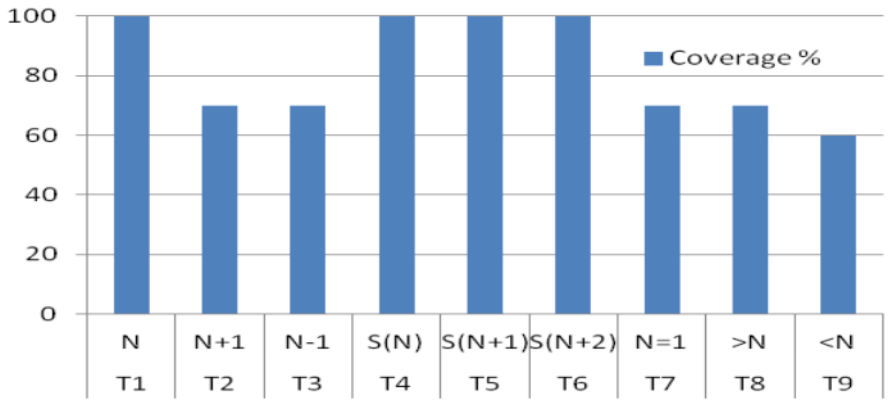

Fig. 5: Graph analysis of coverage

Table 1: Test cases with coverage

Test case ID	Data	Coverage (%)
T1	N	100
T2	N+1	70
T3	N-1	70
T4	S(N)	100
T5	S(N+1)	100
T6	S(N+2)	100
T7	N=1	70
T8	>N	70
T9	<N	60

Coverage analysis: Coverage analysis with function coverage as in Table 1 account for nearly 70-100% of code reachable. The graph in Fig. 5 emphasizes the coverage percentage.

CONCLUSION

Testing with composed components for safety critical system provides better results and makes automation task much easier for designing with petrinets. The task is very difficult to achieve as the design with pertrinets is very complex and vast.

RECOMMENDATIONS

Future research can be focused on reducing the size of petrinets so time and cost can be saved.

REFERENCES

Anand, S., E.K. Burke, T.Y. Chen, J. Clark and M.B. Cohen *et al.*, 2013. An orchestrated survey of methodologies for automated software test case generation. J. Syst. Software, 86: 1978-2001.

Beizer, B., 1990. Software Testing Techniques. 2nd Edn., Van Nostrand Reinhold, New York, ISBN: 0-442-20672-0, Pages: 550.

Bertolino, A., 2003. Software testing research and practice. Proceedings of the 10th International Workshop on Abstract State Machines, March 3-7, Taormina, Italy, 244-262.

Braione, P., G. Denaro, A. Mattavelli, M. Vivanti and A. Muhammad, 2014. Software testing with code-based test generators: Data and lessons learned from a case study with an industrial software component. Software Q. J., 22: 311-333.

Drees, S., D. Gomm, H. Plunnecke, W. Reisig and R. Walter, 1986. Bibliography of Petri Nets. In: Applications and Theory in Petri Nets, Grzegorz, R. (Ed.). Springer, Berlin, Germany, ISBN:978-3-540-18086-9, pp: 309-451.

Farooq, U., C.P. Lam and H. Li, 2008. Towards automated test sequence generation. Proceedings of the 19th Australian Conference on Software Engineering, March 26-28, IEEE Computer Society, Washington, USA., pp: 441-450.

Linzhang, W., Y. Jiesong, Y. Xiaofeng, H. Jun, L. Xuandong and Z. Guoliang, 2004. Generating test cases from UML activity diagram based on gray-box method. Proceedings of the 11th Asia-Pacific Software Engineering Conference, Nov. 30-Dec. 3, IEEE Computer Society, pp: 284-291.

Mussa, M., S. Ouchani, A.W. Sammane and L.A. Hamou, 2009. A survey of model-driven testing techniques. Proceedings of the 2009 9th International Conference on Quality Software, August 24-25, 2009, IEEE, Montreal, Quebec, Canada, ISBN:978-1-4244-5912-4, pp: 167-172.

Ravi, G. and K.P.C. Kailash, 2008. Model-based automated test case generation. SETLabs Briefings, 6: 39-46.

Rozenburg, G. and J. Engelfriet, 1998. Basic Models-Advances in Petrinets. In: Lecture Notes in Computer Science, Reisig, W. and G. Rozenberg, (Eds.). Springer, Berlin, Germany, pp: 12-121.

Smitha, P. and N. Sankarram, 2014. A framework for safe composable testing model for multiple applications testing environment. J. Theoretical Appl. Inf. Technol., 63: 292-297.

Utting, M. and B. Legeard, 2006. Practical Model-Based Testing, a Tools Approach. 1st Edn., Morgan Kaufmann Publishers, USA., INBS:9780123725011, Pages: 433.

Wilkinson, T., M. Butler and J. Colley, 2014. A Systematic Approach to Requirements Driven Test Generation for Safety Critical Systems. In: Model-Based Safety and Assessment, Frank, O. and A. Rauzy (Eds.). Springer, Berlin, Germany, ISBN:978-3-319-12213-7, pp: 43-56.