International Journal of Current Advanced Research
ISSN: O: 2319-6475, ISSN: P: 2319 – 6505, Impact Factor: SJIF: 5.995
Available Online at www.journalijcar.org
Volume 6; Issue 5; May 2017; Page No. 3708-3712
DOI: http://dx.doi.org/10.24327/ijcar.2017.3712.0351

Research Article

AN ANALYSIS WITH PETRINET AND TIMED AUTOMATA WITH AUTOMATION IN MOBILE APPLICATIONS

Smitha.P.S and N.Sankar Ram*

Department of Computer Science and Engineering, Velammal Engineering College, Anna University, Tamil Nadu, India

A R T I C L E   I N F O

Article History:

Received 17th February, 2017
Received in revised form 4th March, 2017
Accepted 25th April, 2017
Published online 28th May, 2017

Key words:

Coverage, Automation, Model, Petrinet, Timed Automata

A B S T R A C T

Model based testing is the current research trend in developing abstract model for modern application. Although many application is tested with different models, there is no concrete complete methodology from verification to validation in reducing cost and time for ensuring effectiveness in testing. This paper suggest a concrete approach in developing a complete model with two model based approaches of petrinets and timed automata with automation and analysis of test coverage.

INTRODUCTION

Software testing plays a crucial role in the testing of application to ensure defect free application. Defects are deviation away from product specification .Reduction of defects and efficient methodology for coverage of test cases has been the research topic for discussion over years and still conclusive results are not obtained. The research questions include

1. How to ensure with a methodology which assist verification to validation in guaranteeing sufficient coverage
2. Automation of software testing reduces time and cost and how to automate with model based testing
3. Which model can be taken for analysis and how the coverage is analyzed.

The answer to the above approaches lies in the development of a model with components of having common interfaces, functionality, inputs and outputs with compatible environment from the requirements manual. The vast requirement manual can be composed into functionality and dependencies between components can be established. The output obtained is the Composable and the compatible model. Composability is a system design principle that deals with the inter-relationships of components. A highly composable system provides recombinant components that can be selected and assembled in various combinations to satisfy specific user requirements. In information systems, the essential features

That make a component composable are that it be

- self-contained (modular): it can be deployed independently – note that it may cooperate with other components, but dependent components are replaceable
- stateless: it treats each request as an independent transaction, unrelated to any previous request. Stateless is just one technique; managed state and transactional systems can also be composable, but with greater difficulty.

It is widely believed that composable systems are more trustworthy than non-composable systems because it is easier to evaluate their individual parts [1]. A composable system behavioural properties can be checked by initiating a finite automation and Petrinets. Safety and liveness properties are checked. Model checking and testing are two areas with a similar goal: to verify that a system satisfies a property [5] [6]. Model-based testing (MBT) relies on models of a system under test and/or its environment to derive test cases for the system [7]. Though object oriented programs are helpful in programming large systems, testing of such systems requires much more effort and time [8]. Creating a finite Automation involves states and transition between states. Petri nets was developed from the work of Carl Adam Petri in 1962 who in his doctoral thesis ``Kommunikation mit Automaten," [3][4] [Communication with automata], gave the theory of

Corresponding author:* **N .Sankar Ram
Department of Computer Science and Engineering, Velammal Engineering College, Anna University , Tamil Nadu ,India

communication between asynchronous components of a computer system. His dissertation was a theoretical development of the basic concepts and was particularly concerned with relationships between events from where Petri nets were developed. Petrinets- a graphical and mathematical modeling tool. Petrinets are a promising tool for describing and studying information processing systems that are characterized as being concurrent, synchronous, distributed, parallel, deterministic and/or stochastic [9]. Finite state machines and petrinets are one among the various conceptual and computational models that have been widely used in analyzing diverse web service research areas [10]. Due to the high number and diversity of users, new testing approaches are necessary to reduce the occurrence of faults and ensure better quality in mobile applications [11]. Petrinets and Timed automata are constructed with automation and formal verification done to ensure all states are reachable in mobile safety application. Automatically generating effective test suites promises a significant impact on testing practice by promoting extensively tested software within reasonable effort and cost bounds [12]. Sequence diagram is generated from the automated model which generates test cases and ultimately the coverage of test cases.

The test cases coverage illustrates petrinets gain edge over timed automata in a set of applications. Apart from introduction in Section I, the rest of the paper is organised as follows. Section 2 describes process for verification and validation and section 3 describes composable model and section 4 describes model with petrinets and timed Automata and section 5 describes results and conclusion.

Process for Verification and validation

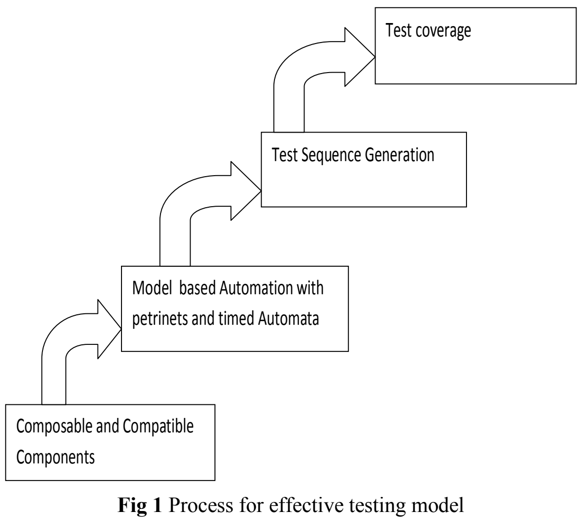

Test coverage
Test Sequence Generation
Model based Automation with petrinets and timed Automata
Composable and Compatible Components

Fig 1 Process for effective testing model

For effective testing process there needs to be a complete scenario which specifies the component to be composed from requirements manual and model being generated with automated tool and test sequences generated which are checked for test coverage as demonstrated in fig 1

Composable Model for Mobile Safety Application

Fig 2 represents a domain model with composable requirements with common inputs and interfaces and output from one composable node to another being grouped into the above models. The requirements can be done by grouping the requirements under each composable node which are critical to the working of software.

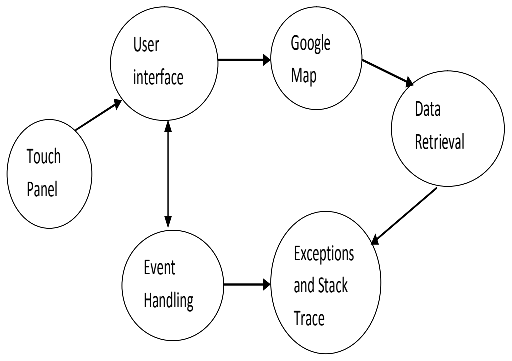


Fig 2 A composable model

Models with Petrinets and Finite State Machine

Generation of Extended Finite State Machine (EFSM) EFSM

is an extension of the original FSM. The expressiveness power of EFSM makes it capable of modeling system specification that include variables and operations based on variable values. In an FSM, the transition is associated with a set of inputs and a set of output functions, whereas in an EFSM model, the transition will be fired if the predicate conditions are all satisfied, moving the machine from the current state to the next state and performing the specified data operations.

An EFSM is 5-tuple = (S, I, O, T, V), such that

• S is a finite set of states, • I is a set of inputs symbols, • O is a set of output symbols, • T is a set of transitions, • V is a set of variables, and
State changes: The transition t in the set T is a 6-tuple:
t = T(st, ´ st, it, ot, Pt, At) where,
• st is the current state,
•´ st is the next state,
• it is the input,
• ot is the output,
• Pt(~v) is predicates on the current variable values, and
• At(~v) is the action on variable values.

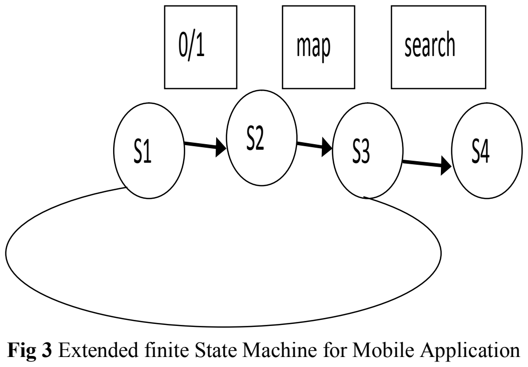


Fig 3 Extended finite State Machine for Mobile Application

Fig 3 represents the flow of states from S1 to state S2 on the input from the user interface screen of for giving input(1) and for not giving(0). From state S2 to S3 based on the input from the user interface to goes for mapping and then from S3 to S4 on searching with the database. The transitions are from S1 to S3 on giving the input from the user interface screen to search for items in the database.

Safety checking

Safety checking can be done with reachability and livveness property based on the input given. If all states are reachable and live and it leads to an end state then that input is reachable and leading to an safe state otherwise it leads to an unsafe state.

Model checking for Extended finite State Machine

```
import net.s1.AbstractListener;
import net.s1.Action;
import net.s1.FsmModel;
import net.s1.RandomTester;
import net.s1.Tester;
import net.s1.Transition;
import net.s1.VerboseListener;
import net.s1.coverage.CoverageMetric;
import net.s1.coverage.TransitionCoverage;

/** Simple example of a finite state machine (FSM) for testing.
 */
public class FSM implements FsmModel
{
  private int state = 0;  // 0..2

  public FSM()
  {
    state = 0;
  }
  public String getState()
  {
    return String.valueOf(state);
  }

  public void reset(boolean testing)
  {
    state = 0;
  }
  public boolean action1Guard() { return state == 2; }
  public @Action void action1()
  {
    //   System.out.println("action0: " + state + " --> 0");
    state = 0;
  }
  public boolean action1Guard() { return state == 2; }
  public @Action void action1()
  {
    //   System.out.println("action1: " + state + " --> 1");
    state = 1;
  }

  public boolean action2Guard() { return state == 0; }
  public @Action void action2()
  {
    //   System.out.println("action2: " + state + " --> 2");
    state = 2;
  }

  public boolean actionNoneGuard() { return state != 1; }
  public @Action void actionNone()
  {
    // leave state the same.
    //     System.out.println("actionNone: " + state + " --> " + state);
  }
  public static void main(String args[])
  {
    // create our model and a test generation algorithm
    Tester tester = new RandomTester(new FSM());

    // build the complete FSM graph for our model, just to ensure
    // that we get accurate model coverage metrics.
    tester.buildGraph();

    // set up our favourite coverage metric
    CoverageMetric trCoverage = new TransitionCoverage();
    tester.addCoverageMetric(trCoverage);

    // ask to print the generated tests
    tester.addListener("v1",                    new V1Listener(tester.getModel()));

    // generate a small test suite of 20 steps (covers 4/5 transitions)
    tester.generate(50);

    tester.getModel().printMessage(trCoverage.getName() + " was "
        + trCoverage.toString());
  }
}
```

Transition coverage

```
done (0, actionNone, 0)
done (0, action2, 2)
done (2, action1, 1)
done Forced reset(true)
done (0, action2, 2)
done (2, action0, 0)
done (0, action2, 2)
done (2, action0, 0)
done (0, actionNone, 0)
done (0, actionNone, 0)
done (0, actionNone, 0)
done (0, action2, 2)
done (2, action0, 0)
done Random reset(true)
done (0, actionNone, 0)
done (0, actionNone, 0)
done (0, action2, 2)
done (2, action0, 0)
done (0, action2, 2)
done Random reset(true)
Transition Coverage was 4/5
```

The source code generated for the above state transi
tion diagram represents the complete test coverage details.

Generation of Petrinet Model

Petri net constitutes places and transitions; the places to which a transition ends called output places and the places from which a transition starts are called the inputs places to the transitions. Places may contain a number of marks, called tokens. The distribution of tokens over the places represents a configuration of the net called the marking. A Petri net may fire whenever there are sufficient number of tokens at each the input places and, firing implies that these tokens consumed and one token is placed each of the output places. Petri nets can be nondeterministic, i.e., when multiple transitions are enabled simultaneously, any one of them may fire and a firing is atomic, i.e., a single non-interruptible event. Since the behaviour of firing is nondeterministic and there may be present multiple tokens anywhere in the Petri net or even in the same place so, Petri net is suited for modelling concurrent behaviour of distributed systems. To analyse the dynamic behaviour of a Petri net modelled systems in reference with states and state changes, each place may hold none or positive integral number of tokens. The condition associated with place is true or false is indicated by the presence or absence of a token at that place. A Petri net is formally defined[2] as a 5-tuple N = (P, T, I, O, M0), where (1) P = {p 1 , p 2 , …, p m } is a finite set of places; (2) T = {t 1 , t 2 , …, t n } is a finite set of transitions, P ∪ T ≠ ∅, and P ∩ T = ∅; [4] (3) I: P × T → N is an input function that defines directed arcs from places to transitions, where N is a set of nonnegative integers; (4) O: T × P → N is an output function that defines directed arcs from transitions to places; and (5) M0 : P → N is the initial marking.

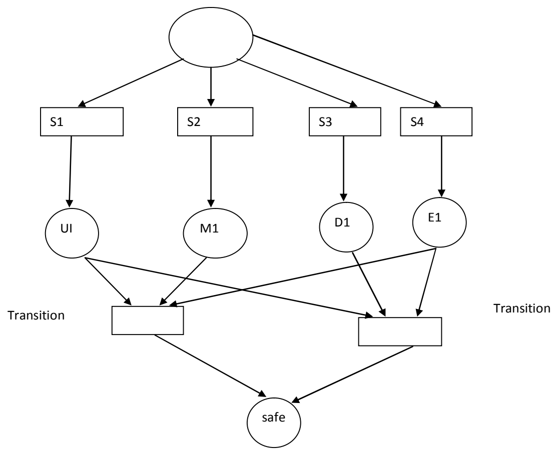


Fig 3 A petrinet Model

UI-User Interface
M1-Maps
D1-Data Retrieval
E1-Exceptions

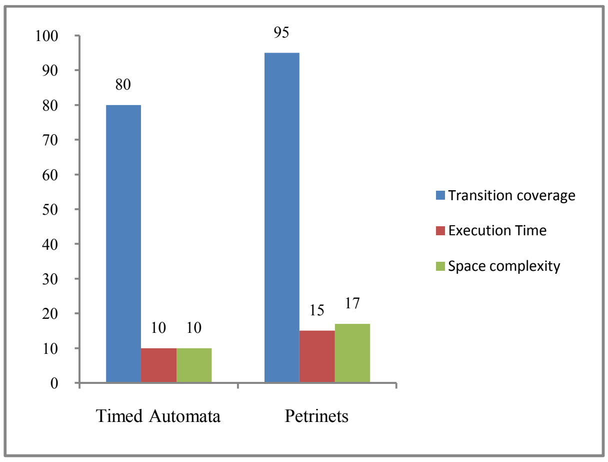
RESULTS

Petrinets

Time complexity = $\sum Pi(Ttt+Tpt)$
Where Ttt is the time for token transition
Tpt is the time for real transition
Space complexity=$\sum Pi(Sst+Sss+Sst)$

Where Sst is the storage token for process tokens
Sss is the storage token for states and their details
Sst is the storage token for transition and their details
Timed Automata

Time complexity = $\sum Pi(Ttt)$
Where Ttt is the time for token transition
Space Complexity=$\sum Pi(Sss+Sst)$
Where Sss is the storage token for states and their details
Sst is the storage token for transition and their details



CONCLUSION

Although many models exist for model based testing, petrinets and timed automata are relevant to many modern applications and use of automated tools largely reduce the cost and time for testing. Test coverage being done on both model shows petrinets have an edge over timed automata in coverage of test cases.

References

1. Peter G. Neumann (2004). 'Principled Assuredly Trustworthy Composable Architectures' (Report).
2. http://bluehawk.monmouth.edu/~jwang/Ch024.pdf.
3. James L. Peterson, Petri Net Theory and the Modelling of Systems, Prentice-Hall, Englewood Cliffs, New Jersey, (April 1981), 290 pages. Translated into Russian and Japanese. ISBN 0-13-661983
4. James L. Peterson, An Introduction to Petri Nets, Proceedings of the National Electronics Conference, Volume 32, (October 1978).
5. Gaudel, M.-C., Lassaigne, R., Magniez, F. and de Rougemont, M. (2013) Some Approximations in Model Checking and Testing. arXiv: 1304.5199.
6. Soliman, D.; Thramboulidis, K.; Frey, G.: Transformation of Function Block Diagrams to UPPAAL Timed Automata for the Verification of Safety Applications. *Annual Reviews in Control* 36 (2012), pp. 338-345.
7. Utting, M., Pretschner, A. and Legeard, B. (2012) A Taxonomy of Model-Based Testing Approaches. *Software Testing, Verification and Reliability*, 22, 297-312.
8. Vipin Kumar K S, Sheena Mathew, (2015), Model Based Distributed Testing of Object Oriented Programs, Elsevier, pp 859-866
9. Tadoa Muarata(1989), Petrinets: Properties, Analysis and Application, Proceedings of IEEE, vol 77,no.4
10. Thirumaran.M, Dhavachelvan.P, Aishwarya.D, K.Rajakumarid (2013), Conventional Usage of finite state machine over petrinet in webservice change management framework, Elsevier, pp 99-109
11. Guilherme de Cleva Farto, Andre Takeshi Endo, Evaluating the Model-Based Testing Approach in the Context of Mobile Applications, Elsevier, pp 3-21
12. Pietro Braione, Giovanni Denaro, Andrea Mattavelli, Mattia Vivanti, Ali Muhammad(2013), “Software

testing with code-based test generators: data and lessons learned from a case study with an industrial software component", *software quality journal, springer*

13. Smitha P.S and Sankar Ram.N 2014 "A Framework for Safe Composable Testing Model For Multiple application Testing environment", *J. of Theoretical and Applied Information InformationTechnology*. Vol 63(2): 292-297

How to cite this article:

Smitha .P.S and N.Sankar Ram (2017) ' An Analysis With Petrinet And Timed Automata With Automation In Mobile Applications', *International Journal of Current Advanced Research,* 06(05), pp. 3708-3712.
DOI: http://dx.doi.org/10.24327/ijcar.2017.3712.0351

****