Journal of Robotics, Networking and Artificial Life
Vol. **8**(**2**); September (2021), *pp.* 112–116
DOI: https://doi.org/10.2991/jrnal.k.210713.009; ISSN 2405-9021; eISSN 2352-6386
https://www.atlantis-press.com/journals/jrnal

Research Article

Simulation and Regression Testing Technique for Software Formal Specifications Based on Extended Place/Transition Net with Attributed Tokens

Tomohiko Takagi[1,*], Ryo Kurozumi[2]

[1]*Department of Engineering and Design, Faculty of Engineering and Design, Kagawa University, 2217-20 Hayashi-cho, Takamatsu-shi, Kagawa 761-0396, Japan*
[2]*Division of Reliability-based Information Systems Engineering, Graduate School of Engineering, Kagawa University, 2217-20 Hayashi-cho, Takamatsu-shi, Kagawa 761-0396, Japan*

ARTICLE INFO

Article History
Received 25 November 2020
Accepted 11 May 2021

Keywords
Formal specifications
place/transition net
VDM
simulation
regression testing

ABSTRACT

We propose a technique of simulation and regression testing for Extended Place/transition Net with Attributed Tokens (EPNAT) models, and then show a prototype tool to partially support it. In the technique, the information about a current marking, current values of variables, and current fireable transitions is indicated to assist engineers in finding faults and selecting next transitions to be fired for the simulation. Also, good execution traces in the simulation are recorded as test cases for the regression testing. When an EPNAT model is modified, the test cases can be applied to it in order to reveal regression failures. A preliminary experiment using simple software requirements has been carried out to discuss the effectiveness of the proposed technique.

1. INTRODUCTION

Formal specifications that represent the way to realize software requirements in unambiguous and executable forms play an important role in the development of high-quality software. Extended Place/transition Net with Attributed Tokens (EPNAT) [1] is a formal specification description language for modeling the expected behavior of state transition-based software that consists of multiple objects, such as modules and subsystems. In an EPNAT model, each attributed token corresponds to an object and has variables to characterize the object, which are called attributes. The firing of transitions leads to the increase, decrease and move of attributed tokens, the change of values of attributes and global variables, and so on. The behavior is constrained by invariants, pre-conditions, post-conditions, and type constraints, which are collectively called constraints. Engineers need to understand such complex aspects of the EPNAT model when constructing, validating and refining it. EPNAT is a new language, and thus systematic techniques to address this problem need to be developed.

In this paper, we propose a technique of simulation and regression testing for EPNAT models, and then show a prototype tool to partially support it. In the technique, the information about a current marking (that is, a current distribution of attributed tokens, including current values of attributes), current values of global variables, and current fireable transitions is indicated to assist engineers in finding faults and selecting next transitions to be fired for the simulation. Also, good execution traces in the simulation are recorded as test cases for the regression testing. When an EPNAT model is modified, the test cases can be applied to it to reveal regression failures. This technique has been developed only for EPNAT. A preliminary experiment using simple software requirements has been carried out to discuss the effectiveness of our technique.

EPNAT models can be converted into VDM++ specifications that can be supported by existing tools [2]. They also enable engineers to perform interactive execution and regression testing. The main differences between our technique and the existing tools are that (i) our technique focuses on more abstract aspects of software specifications, (ii) our technique emphasizes the understanding of software specifications, and (iii) our technique cannot support test-first [3].

This paper is organized as follows. Section 2 describes our technique of simulation and regression testing for EPNAT models, and then Section 3 illustrates our prototype tool. Section 4 gives an experimental result to discuss the effectiveness of our technique. Finally, we show conclusion and future work in Section 5.

2. TECHNIQUE OVERVIEW

In this section, we propose the technique of simulation and regression testing for EPNAT models.

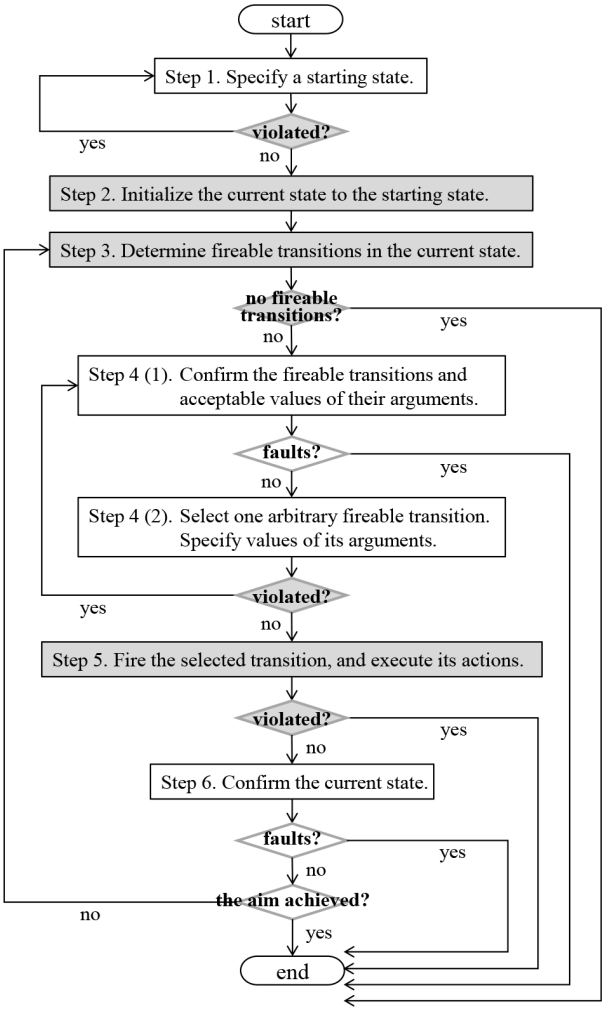
*Corresponding author. Email: takagi@eng.kagawa-u.ac.jp

2.1. Simulation of EPNAT Models

The simulation is intended to help engineers to construct, validate, refine, and understand EPNAT models, and it is applicable to both completed EPNAT models and EPNAT models under construction. The simulation of a given EPNAT model is performed according to the following procedure. The overview of the procedure is shown in Figure 1. Step 2, 3, and 5 can be automatically executed, and should be supported by a tool to save time and effort.

Step 1. An engineer specifies a starting state (that is, a state to start the simulation) according to the aim of the simulation. In this study, a state is characterized by a marking and values of all the global variables in an EPNAT model. Note that a marking in an EPNAT model includes values of attributes. The starting state does not necessarily have to be an initial state (that is, a state established just after software is invoked) or reachable state (that is, a state that can be reached from the initial state).

Step 2. If the starting state violates any type constraints and invariants, this procedure returns to Step 1. Otherwise, the current state of the given EPNAT model is initialized to the starting state.



* *The shaded parts can be automatically executed.*

Figure 1 | Procedure of simulation (overview).

Step 3. The pre-conditions of all the transitions are evaluated to determine fireable transitions in the current state. If there are no fireable transitions, the simulation is terminated.

Step 4. The engineer confirms the determined fireable transitions and acceptable values of their arguments. If he/she finds any faults, the simulation will be stopped. Otherwise, the engineer selects one arbitrary fireable transition and specifies values of its arguments according to the aim of the simulation.

Step 5. If the specified values of the arguments violate any type constraints, invariants and pre-condition, this procedure returns to Step 4. Otherwise, the selected transition is fired, and then its actions are executed by using the specified values of arguments. As a result of them, the current state is changed. If any invariants, and the post-condition of the fired transition are violated due to some sort of failures included in the given EPNAT model, the simulation will be stopped.

Step 6. The engineer confirms the current state. If he/she finds any faults, or if he/she achieves the aim of the simulation, the simulation will be terminated. Otherwise, this procedure returns to Step 3.

2.2. Regression Testing of EPNAT Models

A good execution trace in the simulation is recorded as a test case for the regression testing of future EPNAT models. In this context, the word "good" means that (a) the execution trace does not include the occurrence of faults, (b) the execution trace accelerates the growth of test coverage, (c) the execution trace corresponds to the typical use of software, and (d) the execution trace is useful to cover fault-prone parts [4]. The item (a) is particularly essential for a test case.

When an EPNAT model is modified, the test cases are applied to it to reveal regression failures. Some model-based coverage criteria [5,6] can be introduced to evaluate the effectiveness of the regression testing. Additionally, EPNAT models can be converted into VDM++ specifications [1], and thus common code-based coverage criteria [7] also can be introduced. In general, multiple test cases need to be executed to satisfy a coverage criterion. Also, regression testing is repeatedly performed, and thus it should be automated by a tool to save time and effort.

Figure 2 shows the abstracted structure of a test case for an EPNAT model. It is written in BNF, and parentheses are used to represent an optional element. A test case is a sequence of successive state transitions of arbitrary length. Each transition is identified by an event, and can have values of arguments. Each transition should follow fireable transitions that include its event. Fireable transitions are determined by a state, and thus should follow a state in a test case. Each state is characterized by a marking and values of all the global variables. If an invalid value is given to an argument of a transition to test a constraint, the following state should be a violation state, that is, a state in which the constraint has been violated. A marking is expressed as a sequence of attributed tokens in each place, and an attributed token is characterized by object type and values of attributes. Each place can hold one or more attributed tokens, but

the place and its attributed tokens need to have the same object type. Therefore, the object type of each attributed token is important information in the regression testing of EPNAT models.

The flow of the regression testing is similar to the one of the simulation. A starting state and transitions correspond to test data (also called test input), and are used in Step 1 and 4, respectively. On the other hand, fireable transitions and states excepting the starting state correspond to expected output, and are used in Step 4 and 6, respectively. When there are no differences between expected output and test output through the execution of a given test case, it is concluded that the test case has successfully passed. Otherwise, an engineer needs to find and fix a regression failure, or update the test case so as to reflect the latest true software specification. After that, the engineer should perform confirmation testing, that is, apply the failed test case to the fixed EPNAT model, or apply the updated test case to the EPNAT model.

3. PROTOTYPE TOOL

This section shows our prototype tool to partially support the simulation and regression testing of EPNAT models. As shown in Figure 3, the prototype tool consists of (A) EPNAT model editor, (B) VDM++ specification editor, and (C) EPNAT simulator. (A) and (B) have been developed in our previous study [1], and (C) has been developed in this study. (A) allows an engineer not only to construct his/her EPNAT model using GUI, but also to automatically convert the model into a VDM++ specification. The VDM++ specification can be further developed by using (B). When an EPNAT model on (A) needs to be simulated, (C) can be invoked from (A). (C) interacts with an existing tool called VDMJ [2] to execute a given EPNAT model.

An EPNAT model under simulation is visualized on the right pane of (C). Fireable transitions in the current state are highlighted in green. Therefore, an engineer will be able to easily confirm them, and select a next transition to proceed with his/her simulation. Also, the current state and the simplified execution trace (described as a sequence of fired transitions) are indicated on the left pane. When an engineer selects a fireable transition and specifies its values of arguments on the right pane, (C) automatically executes the firing of the selected transition, executes its actions, evaluates constraints, and then updates the current state, the execution trace, and the graphical image of the EPNAT model. An engineer can stop and reset the current simulation at any time. When an engineer thinks that the execution trace is good for a test case in regression testing, he/she can save it.

An engineer can start automated regression testing by loading a test case and an EPNAT model to be tested.

4. EXPERIMENT

To discuss the effectiveness of the proposed technique, we have carried out a preliminary experiment using software requirements of a simple load balancer [1]. In the experiment, the simulation on our prototype tool is compared with Scenario-Based Reading (SBR), that is, a review technique [8] to find faults in normal software specifications by using expected usage scenarios. The experiment consists of the following four steps:

Step 1. In accordance with the software requirements, we correctly create software Specifications written in a Natural language (SN) and an EPNAT model.

```
<test-case> ::= <state> (<fireable-transitions> <state-transitions>)
<state-transitions> ::= <transition> <state> (<fireable-transitions> <state-transitions>)
<fireable-transitions> ::= <event> (<fireable-transitions>)
<transition> ::= <event> (<values-of-arguments>)
<state> ::= <marking> <values-of-global-variables> | <violation-state>
<marking> ::= <place> (<attributed-tokens-in-the-place>) (<marking>)
<attributed-tokens-in-the-place> ::= <attributed-token> (<attributed-tokens-in-the-place>)
<attributed-token> ::= <object-type> <values-of-attributes>
```

Figure 2 | Abstracted structure of a test case for an EPNAT model (written in BNF).

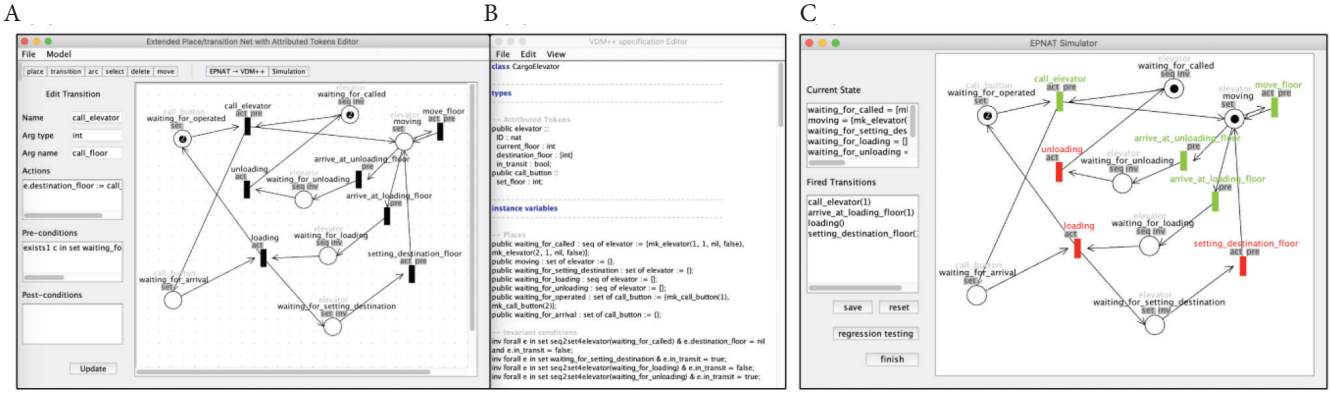


Figure 3 | Screen shots of our prototype tool. (A) EPNAT model editor. (B) VDM++ specification editor. (C) EPNAT simulator.

Table 1 | Experimental results

Technique	Detection rate				Ave. no.
	f_1	f_2	f_3	Ave.	
SBR	1/5	2/5	0/5	0.2	0.6
Simulation	4/5	4/5	1/5	0.6	1.8

Step 2. We insert three faults f_1, f_2 and f_3 into SN, and create Faulty SN (FSN). Additionally, we insert the three faults into the EPNAT model, and create a faulty EPNAT model.

Step 3. In accordance with the software requirements, we correctly create three expected usage scenarios s_1, s_2 and s_3 for SN. They are written in a natural language. s_1 is useful to reveal f_2, and s_3 is useful to reveal f_1. Additionally, we convert the three expected usage scenarios into three simplified execution traces for the EPNAT model.

Step 4. Five students (three in the master's course and two in the undergraduate course at our university) read the software requirements. After that, each of them performs SBR, that is, tries to find the three faults in FSN by using the three expected usage scenarios within 20 min. Additionally, each of them performs the simulation, that is, tries to find the three faults in the faulty EPNAT model by using the three simplified execution traces within 20 min.

Table 1 shows the results of this experiment. The simulation has greater detection rate of each fault than SBR. The average number of faults detection per one student is 1.8 on simulation, and is 0.6 on SBR. After the experiment, we interviewed each of the students and got the following comments:

- The EPNAT model, rather than SN, was or might be useful to understand the software (three students).
- It was easier to find faults in the faulty EPNAT model than in FSN (five students).
- The simulation is useful for understanding the EPNAT model (four students). The simulation is useful for testing and validating, rather than for understanding (another student).
- There is room to improve the user interface of the prototype tool (two students).

The results of the experiment and of the interview indicate the effectiveness of our technique. However, note that the simulation follows SBR in Step 4, and thus the students may have developed the understanding of the given faulty software specifications before starting the simulation. Further evaluation will be needed in future study.

5. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a technique of simulation and regression testing for EPNAT models, and then have shown a prototype tool to partially support it. In the technique, the information about a current marking (including current values of attributes), current values of global variables, and current fireable transitions is indicated to assist engineers in finding faults and selecting next transitions to be fired for the simulation. Also, good execution traces in the simulation are recorded as test cases for the regression testing. When an EPNAT model is modified, the test cases can be applied to it to reveal regression failures. This technique has been developed only for EPNAT. We have carried out a preliminary experiment using simple software requirements. The results of the experiment and of the interview indicate the effectiveness of the technique, but further evaluation will be needed in future study.

When a failure that was caused in the earlier stage of software development is found in the later stage (typically, the processes of system testing and acceptance testing), the cost to fix it generally tends to become higher. Therefore, it is important to find and fix failures in the earlier stage, and it is expected that our technique can be applied to address this problem. However, our tool is under development, and the functions to support our technique are not completely implemented at present. Also, the following matters need to be tackled in future work in order to improve the technique:

- An engineer will often need to confirm that the set of reachable states in an EPNAT model under construction includes all the indispensable states to satisfy given software requirements and also the set does not include any invalid states. However, test cases are manually created in our technique and tool, and thus it will be difficult for most engineers to do such task systematically at small cost. Model checking [9] may be useful to address this problem.
- After an engineer has made a modification on an EPNAT model, he/she will often need to maintain some existing test cases, that is, to update some existing test cases so as to reflect the latest true software specification. A technique and tool to systematically support it should be constructed.
- Coverage criteria are useful to create good execution traces in simulation and to select good test cases in regression testing. Engineers will need guidelines for the effective use of coverage criteria, and will need a tool to automatically suggest execution traces and test cases according to coverage criteria.

Based on the above, we will develop the tool to support our extended technique, and apply it to pilot projects to evaluate its effectiveness.

CONFLICTS OF INTEREST

The authors declare they have no conflicts of interest.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Number JP17K00103.

REFERENCES

[1] T. Takagi, R. Kurozumi, Software modeling technique and its prototype tool for behavior of multiple objects using extended place/transition nets with attributed tokens, J. Robot. Netw. Artif. Life 7 (2020), 194–198.

[2] J. Fitzgerald, P.G. Larsen, P. Mukherjee, N. Plat, M. Verhoef, Validated designs for object-oriented systems, Springer-Verlag, London, 2005.
[3] K. Beck, Test-driven development: by example, Addison-Wesley Professional, Boston, MA, United States, 2002.
[4] J.A. Whittaker, J. Arbon, J. Carollo, How google tests software, Addison-Wesley Professional, Boston, MA, United States, 2012.
[5] T. Takagi, R. Kurozumi, T. Katayama, State transition tuple coverage criterion for extended place/transition net-based testing, Proceedings of the 2019 IEEE Pacific Rim International Symposium on Dependable Computing (PRDC), IEEE, Kyoto, Japan, 2019, pp. 29–30.
[6] T. Takagi, N. Oyaizu, Z. Furukawa, Concurrent N-switch coverage criterion for generating test cases from place/transition nets, Proceedings of the 2010 IEEE/ACIS 9th International Conference on Computer and Information Science, IEEE, Yamagata, Japan, 2010, pp. 782–787.
[7] B. Beizer, Software testing techniques, 2nd ed., Van Nostrand Reinhold, New York, NY, United States, 1990.
[8] T. Thelin, P. Runeson, C. Wohlin, An experimental comparison of usage-based and checklist-based reading, IEEE Trans. Softw. Eng. 29 (2003), 687–704.
[9] E.M. Clarke, O. Grumberg, D.A. Peled, Model checking, MIT Press, Cambridge, MA, United States, 1999.

AUTHORS INTRODUCTION

Dr. Tomohiko Takagi

He received the B.S., M.S. and PhD degrees from Kagawa University in 2002, 2004 and 2007, respectively. He became an Assistant Professor in 2008, and a lecturer in 2013 in the Faculty of Engineering at Kagawa University. Since 2018 he has been an Associate Professor in the Faculty of Engineering and Design at Kagawa University. His research interests are in software engineering, particularly software testing.

Mr. Ryo Kurozumi

He received the B.S. degree from Kagawa University in 2019. He is a master's student in the Graduate School of Engineering at Kagawa University. His research interests are in software engineering, particularly software design.