Simulation and Regression Testing for Behavior of Software Models Based on Extended Place/Transition Net with Attributed Tokens

Tomohiko Takagi
Faculty of Engineering and Design, Kagawa University
2217-20 Hayashi-cho, Takamatsu-shi, Kagawa 761-0396, Japan

Ryo Kurozumi
Graduate School of Engineering, Kagawa University
2217-20 Hayashi-cho, Takamatsu-shi, Kagawa 761-0396, Japan
E-mail: takagi@eng.kagawa-u.ac.jp, s19g457@stu.kagawa-u.ac.jp

Abstract

We propose a technique of simulation and regression testing for Extended Place/transition Net with Attributed Tokens (EPNAT) models, and then show an early prototype tool to partially support it. In the technique, the information about a current marking (a current distribution of attributed tokens, including current values of attributes), current values of global variables, and current fireable transitions is indicated for the simulation, and also the 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.

Keywords: software modeling, place/transition net, VDM, simulation, regression testing

1. Introduction

Formal specifications that represent abstracted 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. Engineers need to understand such complex aspects of the EPNAT model when constructing, validating and refining it.

In order to address this problem, we propose a technique of simulation and regression testing for EPNAT models, and then show an early 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 for the simulation, and also the 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.

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 early prototype tool. Finally, we show discussion and future work in section 4.

2. Technique Overview

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

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 Fig. 1. Step 2, 3, 5, and 7 can be automatically executed, and should be supported by a tool.

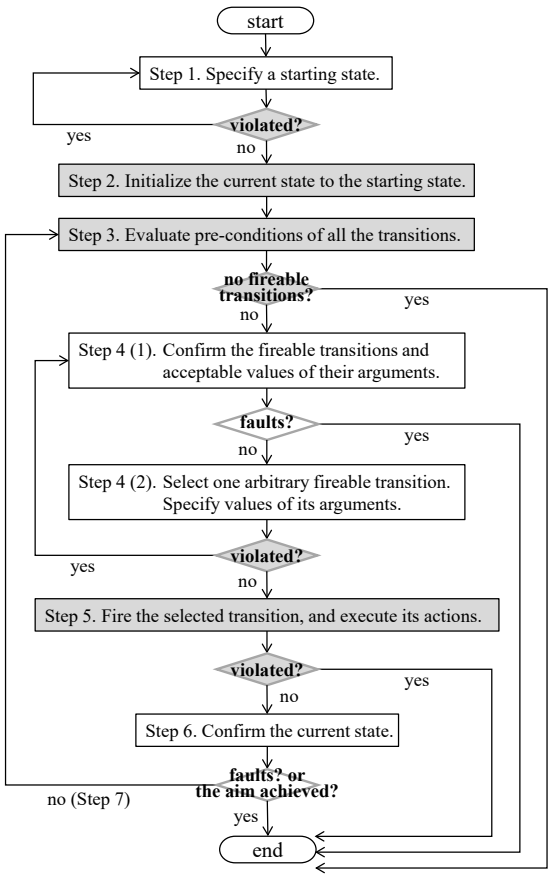
Step 1. An engineer specifies a starting state (that is, a state to start the simulation) according to the aim of the simulation. In this study, a state is characterized by a marking and values of all the global variables in an EPNAT model. Note that a marking in an EPNAT model includes values of attributes. The starting state does not necessarily need 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, the procedure returns to Step 1. Otherwise, the current state of the given EPNAT model is initialized to the starting state.

Step 3. The pre-conditions of all the transitions are evaluated in order 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 and invariants, the 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.



** The shaded parts can be automatically executed.*

Fig. 1. Procedure of simulation (overview).

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.

Step 7. 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. In this context, the word "good" means that (a) the execution trace does not include the occurrence of faults, (b) the execution trace is useful for 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[2]. The item (a) is particularly essential for a test case.

```
<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>
```

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

When an EPNAT model is modified, the test cases are applied to it in order to reveal regression failures. Some model-based coverage criteria[3,4] can be introduced to evaluate the effectiveness of the regression testing. Additionally, EPNAT models can be converted into VDM++ specifications[1,5], and thus common code-based coverage criteria[6] also can be introduced. In general, multiple test cases need to be executed to satisfy coverage criteria. Also, regression testing is repeatedly performed, and thus it should be automated by a tool in order to save time and effort.

Fig. 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 in order to test invariants and a post-condition, the following state should be a violation state, that is, a state in which some invariant or the post-condition 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 regression testing of EPNAT models.

Transitions and a starting state correspond to test data (also called test input). On the other hand, fireable transitions and states excepting the starting state correspond to expected output. 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. Early Prototype Tool

This section shows our early prototype tool to partially support the simulation and regression testing of EPNAT models. The tool includes two functions. One is an EPNAT editor to support the construction of EPNAT models, which has been developed in our previous study[1]. Another is an EPNAT simulator we are developing in this study, and it can be invoked from the EPNAT editor. The EPNAT simulator interacts with an existing tool called VDMJ[5] in order to execute a given EPNAT model.

Fig. 3 shows a screen shot of the EPNAT simulator that executes the simulation of the EPNAT model that represents the behavior of a simple load balancer[1]. The EPNAT model under simulation is visualized on the right pane of the EPNAT simulator. Fireable transitions in the current state are highlighted in green. Therefore, an engineer as a user of the EPNAT simulator will be able to easily confirm them, and select a next transition in order 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, the EPNAT simulator automatically executes the firing of the selected transition, 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. Discussion and Future Work

In this paper, we have proposed a technique of simulation and regression testing for EPNAT models, and then have shown an early 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 for the simulation, and also the 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.

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[7] 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.

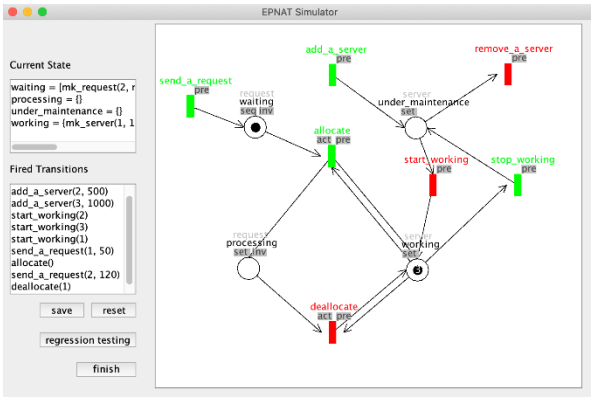


Fig. 3. Screen shot of an early prototype tool (EPNAT simulator).

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

Acknowledgements

This work was supported by JSPS KAKENHI Grant Number JP17K00103.

References

1. T. Takagi and R. Kurozumi, Software Modeling Technique and its Prototype Tool for Behavior of Multiple Objects Using Extended Place/Transition Nets with Attributed Tokens, *Journal of Robotics, Networking and Artificial Life*, Vol.7, No.3, pp.194-198, Dec. 2020.
2. J.A. Whittaker, J. Arbon and J. Carollo, *How Google Tests Software*, Addison-Wesley Professional, 2012.
3. T. Takagi, R. Kurozumi and T. Katayama, State Transition Tuple Coverage Criterion for Extended Place/Transition Net-Based Testing, *Proceedings of Pacific Rim International Symposium on Dependable Computing*, pp.29-30, Dec. 2019.
4. T. Takagi, N. Oyaizu and Z. Furukawa, Concurrent N-switch Coverage Criterion for Generating Test Cases from Place/Transition Nets, *Proceedings of 9th International Conference on Computer and Information Science*, pp.782-787, Aug. 2010.
5. J. Fitzgerald, P.G. Larsen, P. Mukherjee, N. Plat and M. Verhoef, *Validated Designs for Object-Oriented Systems*, Springer-Verlag London, 2005.
6. B. Beizer, *Software Testing Techniques*, Van Nostrand Reinhold, 2nd edition, 1990.
7. E.M. Clarke Jr., O. Grumberg and D. Peled, *Model Checking*, MIT Press, 1999.