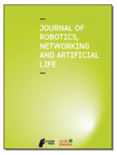
Journal of Robotics, Networking and Artificial Life
Vol. 5(4); March (2019), *pp.* 278–282
DOI: https://doi.org/10.2991/jrnal.k.190402.001; ISSN 2405-9021 print; 2352-6386 online
https://www.atlantis-press.com/journals/jrnal



Animated Graphics-based Training Support Method and Prototype Tool for Bug Fixing of Extended Place/Transition Nets

Tomohiko Takagi[1,*], Shogo Morimoto[2], Yuki Ue[2], Yoshiro Imai[1]

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

ARTICLE INFO

Article History
Received 15 October 2018
Accepted 22 November 2018

Keywords
Training support
software modeling
place/transition net
bug fixing

ABSTRACT

This paper describes an animated graphics-based training support method for bug fixing of Extended Place/transition Nets (EPNs), and illustrates a prototype tool that the core of the method has been implemented into. The prototype tool gives a trainee a faulty EPN and its animated graphics. The motion of the animated graphics is synchronized with the motion of the faulty/fixed EPN. Therefore, it is expected that a trainee can intuitively understand the EPN, and smoothly try to fix its bug. The result of bug fixing is checked by the prototype tool. The discussion based on trial application of the prototype tool reveals its effectiveness and challenges for the future.

1. INTRODUCTION

Formal models in graphical and mathematical languages can give unambiguous abstracted representation of the behavior of software. In software development, they are useful to define specifications, generate program codes, and generate test cases systematically. For example, in Ho and Lin [1], time Petri nets are used to model real-time software and generate test cases that focus on its state transition and timing properties. Katayama et al. [2] show a prototype tool to generate boundary value test cases from formal specifications written in VDM++ [3].

In our previous study, an Extended Place/transition Net (EPN) that is a combination of a PN and VDM++ was proposed and introduced into test case generation [4] and evaluation [5]. The PN is a kind of Petri nets that can be used to model software from the viewpoint of its basic state transition. The advantage of the EPN for the PN is that some complex and essential aspects such as guards and actions on transitions can be written in VDM++, and a state of software is closely expressed as a marking and values of instance variables. The EPN is executable on a tool, and software engineers can confirm the abstracted behavior of software before beginning its implementation. However, it would not be so easy for most of software engineers to learn and use the EPN efficiently. In general, a formal model with higher representation power requires software engineers to gain higher degree of expertise and experience, which will prevent the spread of the formal model in actual software development. There are only a limited number of software engineers skilled in formal modeling, and they are too busy to train unskilled software engineers. Therefore, a training support tool should be established to solve this problem.

In this study, we are planning to develop a tool to support the training of formal modeling using EPNs. Different support methods should be constructed based on the difference of trainees' achievement. First, we classified the trainees' achievement into the following two levels:

- Level 1: A trainee has gained the ability to understand the behavior of software from a given EPN, and fix its bug (i.e., a difference between the EPN and true software requirements).
- Level 2: A trainee has gained the ability to construct an EPN from given software requirements.

As the first step of this study, we have constructed an animated graphics-based training support method and prototype tool to achieve the level 1. This paper shows this method, functions of the prototype tool, and discussion about its effectiveness. The prototype tool gives a trainee a faulty EPN and its animated graphics. The faulty EPN is an EPN that includes an intended bug, and the animated graphics illustrate the behavior of software based on the faulty/fixed EPN. The motion of the animated graphics is linked to (i.e., synchronized with) the motion of the faulty/fixed EPN. Therefore, it is expected that a trainee can intuitively understand the EPN, and smoothly try to fix its bug. The prototype tool checks the result of bug fixing, and evaluates each trainee's achievement.

The rest of this paper is organized as follows. In Section 2, we propose an animated graphics-based training support method. Section 3 illustrates the functions of the prototype tool that the core of the proposed method has been implemented into. Section 4 gives discussion

*Corresponding author. Email: takagi@eng.kagawa-u.ac.jp about effectiveness based on trial application of the prototype tool, and finally Section 5 shows conclusion and future work.

2. TRAINING SUPPORT METHOD

In this section, we propose an animated graphics-based training support method for bug fixing of EPNs. This method consists of (1) creation of exercises, (2) work on exercises, and (3) evaluation.

2.1. Creation of Exercises

A trainer, i.e., a skilled software engineer creates exercises for trainees. Each exercise consists of software requirements that were written in a natural language, an original EPN that was correctly constructed based on the software requirements, a faulty EPN that was constructed by inserting an intended bug into the original EPN, operations to fix the intended bug of the faulty EPN, and animated graphics that illustrate the behavior of software based on the faulty/fixed EPN.

An intended bug can be created by applying existing mutation operators to an original EPN. A good exercise includes an intended bug that cannot be easily found and is likely to be made in actual software development.

The operations for bug fixing are defined as a pair (or pairs) of a kind of existing mutation operators, and the number of times to apply the mutation operator to a faulty EPN. For example, when (arc addition, 2) is defined as operations, correct bug fixing can be achieved by adding two arcs somewhere to a faulty EPN. In other words, the intended bug of the faulty EPN would be created by (arc deletion, 2).

The motion of animated graphics needs to be synchronized with the motion of the faulty/fixed EPN that is shown as the change of its current state (i.e., its current marking and values of instance variables) and the highlighting of a recently fired transition. Trigger of motion in the animated graphics is the fire of a transition in the faulty/fixed EPN; on the contrary, trigger of the fire of a transition can be the result of motion in the animated graphics. The definition of the trigger in an EPN and its animated graphics is one of trainer's tasks.

2.2. Work on Exercises

A trainee selects an exercise, and works on it. Software requirements, a faulty EPN, and operations for bug fixing are given to the trainee on the exercise. In our method, there are the following two options for the synchronous motion of the faulty/fixed EPN and its animated graphics.

- Option 1: A trainee can view the synchronous motion at any time.
- Option 2: A trainee can view the synchronous motion after finishing his/her bug fixing.

If the option 1 is chosen, animated graphics also need to be given to the trainee from this phase. It is expected that the trainee can intuitively understand the faulty EPN from the animated graphics. If the trainee has found a difference between the faulty EPN and the software requirements, he/she tries to remove an intended bug from the faulty EPN. The trainee can modify the faulty EPN by using only the given operations for bug fixing. Modification to the faulty EPN is reflected on the animated graphics, and thus the trainee can confirm the result of his/her bug fixing by viewing the synchronous motion of the fixed EPN and its animated graphics.

The motion of the animated graphics should be strictly linked to the motion of the faulty/fixed EPN. In order to achieve it, the following procedure is automatically performed based on the trigger definitions.

(i) In the faulty/fixed EPN, each transition is checked whether it can fire or not. A transition is fireable, if all from-places of the transition contain the required number of tokens, and the guard of the transition is true. If there are no fireable transitions, it will be a failure that was caused by an intended bug or an incorrect bug fix, and this procedure is terminated.

(ii) If a trigger event occurs in the animated graphics, a transition relating to the trigger event is identified in the faulty/fixed EPN. If the transition is fireable, corresponding motion in both of the faulty/fixed EPN and the animated graphics is invoked, and then this procedure returns to (i). If the transition is not fireable, it will be a failure that was caused by an intended bug or an incorrect bug fix, and this procedure is terminated.

(iii) A fireable transition is randomly selected, and then corresponding motion in both of the faulty/fixed EPN and the animated graphics is invoked. This procedure returns to (i).

If the above-mentioned procedure is terminated, all the motion is stopped, and then, the faulty/fixed EPN and the animated graphics return to their initial states.

2.3. Evaluation

When a trainee has finished his/her bug fixing, the fixed EPN is automatically compared with the original EPN. If they are the same, it is judged as correct bug fixing. If they are not the same, it is judged as incorrect bug fixing, and the trainee is suggested to retry the exercise. The trainee is shown not only the judgement (correct or incorrect) but also the synchronous motion of the fixed EPN and its animated graphics. The synchronous motion will help the trainee to understand the judgement.

The time taken for the completion of correct bug fixing, and the number of incorrect bug fixing are recorded in order to evaluate not only trainees' achievement but also the quality of exercises.

3. PROTOTYPE OF A TRAINING SUPPORT TOOL

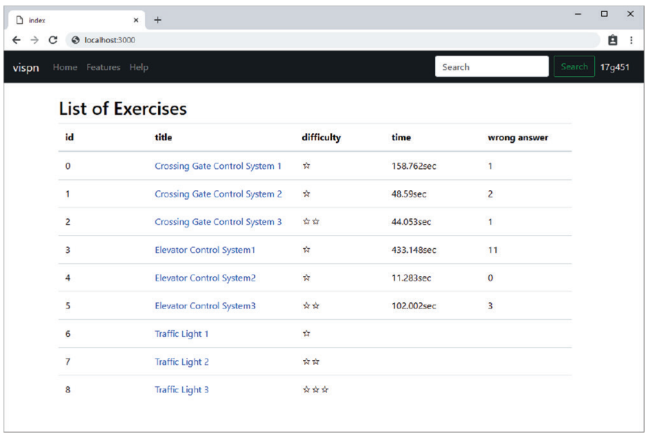
The core of the training support method proposed in the previous section has been implemented as a prototype tool. This section illustrates its functions.

The prototype tool has been constructed as a Web application in order that trainees can use it easily. The aim of this prototyping is to evaluate the effectiveness of animated graphics against understandability of EPNs, and thus it does not include the implementation of functions to help the creation of exercises. Also, only model-based mutation operators [6] are introduced to prepare faulty EPNs and operations for bug fixing.

First, a trainee needs to log on to the prototype tool by using a Web browser, and select an arbitrary exercise on a top page shown in Figure 1. The top page provides the information about each exercise, including its title, degree of difficulty that was supposed by a trainer, time taken for the completion of correct bug fixing, and number of incorrect bug fixing.

Soon after clicking a title of a desired exercise on the top page, a trainee receives a corresponding training page. A training page consists of software requirements, animated graphics, a faulty EPN, operations for bug fixing, and some buttons (an "Answer" button, a "Retry" button, buttons for operations for bug fixing, and so on). When selecting "Crossing Gate Control System" [7] from the list shown in Figure 1, the trainee receives the training page shown in Figure 2. The animated graphics are formed from map chips (blocks to form a map, such as rails and a road) and motion objects on the map (such as a train and a crossing gate). The operation for bug fixing in this exercise is (arc deletion, 1), and therefore a "X" button to apply a mutation operator of arc deletion only once is provided on the pane that contains the faulty EPN of the crossing gate control system. The trainee pushes the "X" button, and clicks an arc of the faulty EPN to delete it. In Figure 3, the arc from T5 to P10 has been deleted from the faulty EPN. The number on a button for operations for bug fixing tells how many times its mutation operator needs to be applied to a faulty EPN, and it decreases with application, as shown in Figures 2 and 3. A button for operations for bug fixing becomes unavailable if its number reaches 0. Also, a trainee can push the "Retry" button to undo all his/her operations at any time.

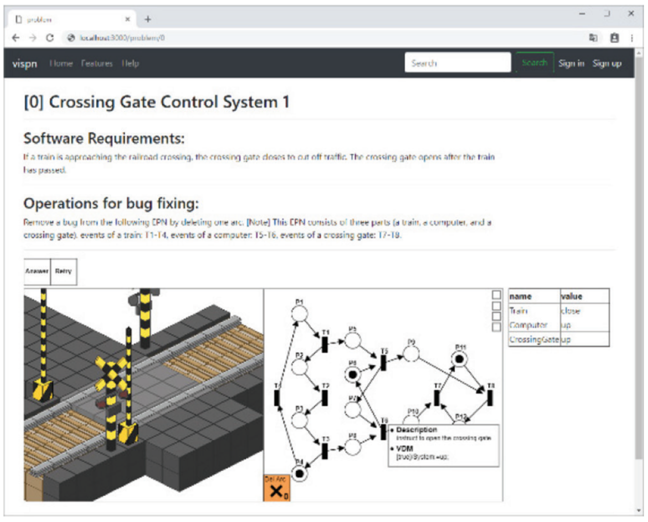
Soon after pushing the "Answer" button, the trainee is notified whether his/her bug fix is correct or not. Additionally, the training page shows the synchronous motion of the fixed EPN and its animated graphics. The motion can be controlled by using fast/normal/slow play buttons and a stop button. Figures 4 and 5 give the screen shots of the training page on the subject of the



List of Exercises

id	title	difficulty	time	wrong answer
0	Crossing Gate Control System 1	☆	158.762sec	1
1	Crossing Gate Control System 2	☆	48.59sec	2
2	Crossing Gate Control System 3	☆☆	44.053sec	1
3	Elevator Control System1	☆	433.148sec	11
4	Elevator Control System2	☆	11.283sec	0
5	Elevator Control System3	☆☆	102.002sec	3
6	Traffic Light 1	☆		
7	Traffic Light 2	☆☆		
8	Traffic Light 3	☆☆☆		

Figure 1 | Top page providing the list of exercises.



[0] Crossing Gate Control System 1

Software Requirements:

If a train is approaching the railroad crossing, the crossing gate closes to cut off traffic. The crossing gate opens after the train has passed.

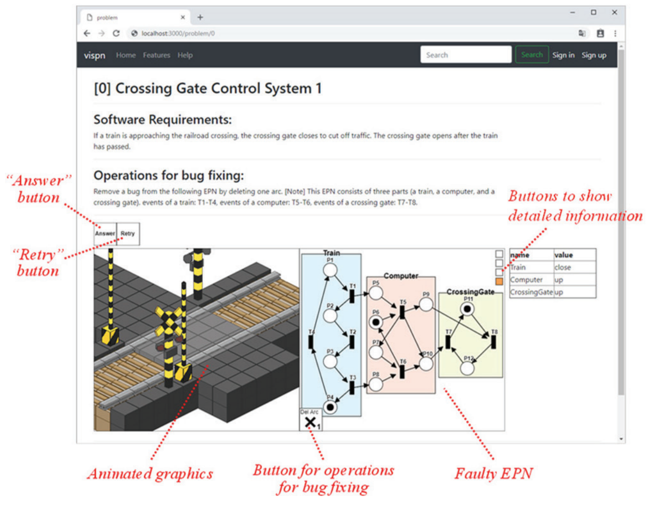
Operations for bug fixing:

Remove a bug from the following EPN by deleting one arc. [Note] This EPN consists of three parts (a train, a computer, and a crossing gate). events of a train: T1-T4, events of a computer: T5-T6, events of a crossing gate: T7-T8.

name	value
Train	close
Computer	up
CrossingGate	up

- Description
  Instruct to open the crossing gate
- VDM
  [true]/System.→up;

Figure 3 | Training page (modifying a faulty EPN).



[0] Crossing Gate Control System 1

Software Requirements:

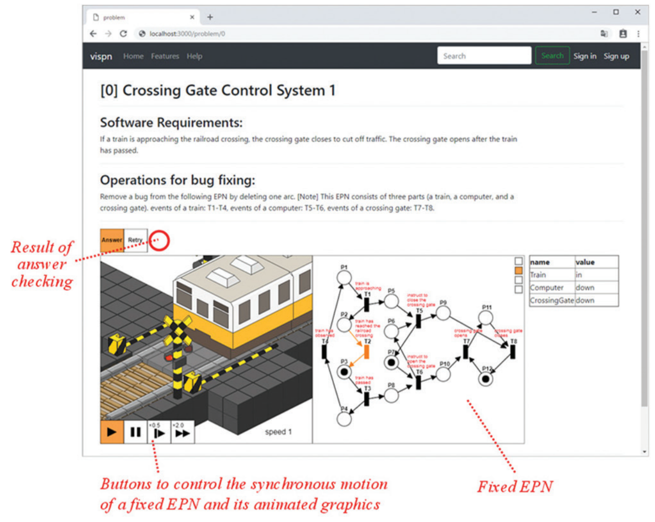
If a train is approaching the railroad crossing, the crossing gate closes to cut off traffic. The crossing gate opens after the train has passed.

Operations for bug fixing:

Remove a bug from the following EPN by deleting one arc. [Note] This EPN consists of three parts (a train, a computer, and a crossing gate). events of a train: T1-T4, events of a computer: T5-T6, events of a crossing gate: T7-T8.

name	value
Train	close
Computer	up
CrossingGate	up

Figure 2 | Training page (initial state).



[0] Crossing Gate Control System 1

Software Requirements:

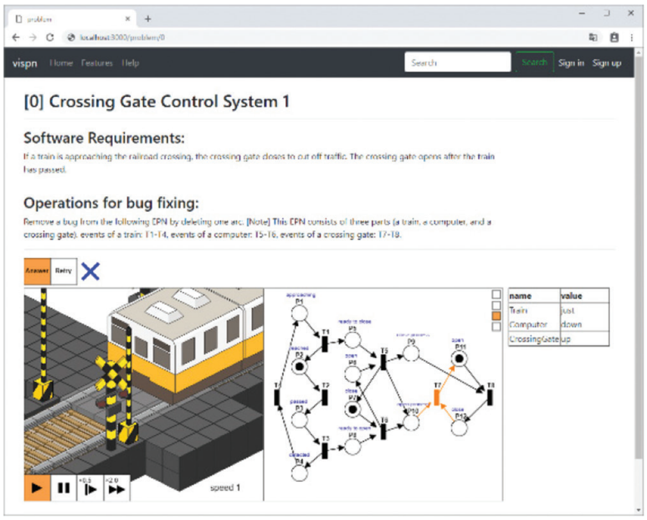
If a train is approaching the railroad crossing, the crossing gate closes to cut off traffic. The crossing gate opens after the train has passed.

Operations for bug fixing:

Remove a bug from the following EPN by deleting one arc. [Note] This EPN consists of three parts (a train, a computer, and a crossing gate). events of a train: T1-T4, events of a computer: T5-T6, events of a crossing gate: T7-T8.

name	value
Train	in
Computer	down
CrossingGate	down

Figure 4 | Training page (answering a correct bug fix).



[0] Crossing Gate Control System 1

Software Requirements:

If a train is approaching the railroad crossing, the crossing gate closes to cut off traffic. The crossing gate opens after the train has passed.

Operations for bug fixing:

Remove a bug from the following EPN by deleting one arc. [Note] This EPN consists of three parts (a train, a computer, and a crossing gate). events of a train: T1-T4, events of a computer: T5-T6, events of a crossing gate: T7-T8.

name	value
Train	just
Computer	down
CrossingGate	up

Figure 5 | Training page (answering an incorrect bug fix).

crossing gate control system, just after pushing the "Answer" button. In Figure 4, the trainee has successfully fixed the intended bug, and the synchronous motion of the fixed EPN and its animated graphics shows that the crossing gate closes with the approach of a train. On the other hand, in Figure 5, the crossing gate does not close in spite of the approach of a train, since the trainee has failed in bug fixing. The trainee can undo his/her incorrect bug fixing, and retry this exercise by pushing the "Retry" button.

Note that the option 2 (see Section 2.2) was basically chosen, and therefore a trainee cannot view the synchronous motion of a faulty/fixed EPN and its animated graphics before pushing the "Answer" button in the prototype tool, which will encourage a trainee to carefully view and think about the EPN at first. When a trainee cannot understand a given faulty EPN, he/she pushes the "Answer" button without any modification and can view the synchronous motion.

4. DISCUSSION

This section gives discussion about effectiveness based on trial application of the prototype tool.

We created six exercises on the subject of a crossing gate control system and an elevator control system, and set them to the prototype tool. After that, a graduate student who is studying a test case generation technique based on EPNs in our laboratory worked on the exercises on the prototype tool. As a result of this trial application, he succeeded in all bug fixing. His average time to finish correct bug fixing is about 133 seconds, and his average number of incorrect bug fixing is three. Finally, we and the graduate student entered a free discussion, and the following opinions were obtained:

(a) The prototype tool is of high quality, and animated graphics seem to accelerate trainee's understanding for EPNs.

(b) However, some exercises have lower degree of difficulty, since same original EPNs and their animated graphics are reused to reduce the cost of creating exercises.

(c) Trainer's effort will be largely devoted to creating animated graphics, and a technique and tool need to be developed to support it.

(d) The quality of intended bugs will depend on the degree of expertise and experience of a trainer. Heuristic approaches such as a genetic algorithm will be able to be applied to support/automate the creation of good intended bugs.

(e) A trainee will not know which exercise to select. It is better that the prototype tool can suggest appropriate exercises for each trainee, according to his/her achievement.

(f) It will be better to allow a trainee to choose between the options 1 and 2, since some trainees (especially beginners) will want to view the synchronous motion from the first. The option 2 seems to be suitable for trainees that have learned EPNs to some extent.

(g) If the prototype tool provides a function that helps trainees and trainers to interact with one another, it may be useful to support their training and creation of exercises.

(h) Some user interfaces should be improved.

The items from (b) to (e) are important, and are our challenges for the future. If (c) is addressed, (b) will be improved. The items from (f) to (h) will not be difficult problems, and (f) was addressed after we had finished the trial application.

5. CONCLUSION AND FUTURE WORK

In this paper, we have proposed an animated graphics-based training support method for bug fixing of EPNs, and showed a prototype tool that the core of the proposed method has been implemented into. As a result of trial application of the prototype tool, we found that animated graphics seem to accelerate trainee's understanding for EPNs, but the cost to create good exercises is not small. To improve this problem, we will construct a technique and tool to assist the creation of animated graphics and intended bugs. Also, we plan to develop a function to suggest appropriate exercises automatically for each trainee, according to his/her achievement.

ACKNOWLEDGMENTS

This work was supported by JSPS KAKENHI Grant Number JP17K00103.

REFERENCES

[1] I. Ho, J-C. Lin, Generating test cases for real-time software by time Petri nets model, Proceedings of Eighth Asian Test Symposium (ATS'99), IEEE, Shanghai, China, 1999, pp. 295–300.

[2] T. Katayama, H. Tachiyama, Y. Kita, H. Yamaba, K. Aburada, N. Okazaki, BWDM: test cases automatic generation tool based on boundary value analysis with VDM++, J. Robot. Netw. Artif. Life 4 (2017), 110–113.

[3] J. Fitzgerald, P.G. Larsen, P. Mukherjee, N. Plat, M. Verhoef, Validated designs for object-oriented systems, Springer-Verlag London, 2005.

[4] T. Takagi, T. Katayama, Negative test case generation from an extended place/transition net-based mutants, Proceedings of the 2018 International Conference on Artificial Life and Robotics (ICAROB2018), ALife Robotics, Oita, Japan, 2018, pp. 513–516.

[5] T. Takagi, S. Morimoto, T. Katayama, Development of a tool for extended place/transition net-based mutation testing and its application example, J. Robot. Netw. Artif. Life 4 (2017), 168–174.

[6] T. Takagi, R. Takata, Z. Furukawa, F. Belli, M. Beyazıt, Metrics for model-based mutation testing based on place/transition nets, Proceedings of the Joint Conference of 21st International Workshop on Software Measurement and Sixth International Conference on Software Process and Product Measurement (IWSM-MENSURA), Nara, Japan, 2011, pp. 7–10.

[7] N.G. Leveson, J.L. Stolzy, Safety analysis using Petri nets, IEEE Transactions on Software Engineering, IEEE, USA, 1987, pp. 386–397.

Authors Introduction

Dr. Tomohiko Takagi

Dr. Tomohiko Takagi 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. Yuki Ue

Mr. Yuki Ue received the B.S. degree from Kagawa University in 2018. 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.

Mr. Shogo Morimoto

Mr. Shogo Morimoto received the B.S. degree from Kagawa University in 2017. He is a master's student in the Graduate School of Engineering at Kagawa University. His research interests are in software engineering, particularly software testing.

Dr. Yoshiro Imai

Dr. Yoshiro Imai received the B.S. degree from Kyoto University in 1980 and the PhD degree from Tokyo University of Agriculture and Technology in 2008. He became an associate professor in 1997 at Kagawa University. Since 2018 he has been a professor in the Faculty of Engineering and Design at Kagawa University. His research interests are in computer architecture, system software and their e-Learning tools development.