SUGISAKA MASANORI

Journal of Robotics, Networking and Artificial Life
Vol. 9(1); June (2022), pp. 81–86
ISSN (Online): 2352-6386; ISSN (Print): 2405-9021
2352-6386 https://alife-robotics.org/jrnal.html

Research Article

A Technique for Learning Software Modeling Using Extended Place/Transition Net and Its Prototype Tool

Tomohiko Takagi[1], Akio Usuda[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 22 May 2022

Keywords

software modeling
place/transition net
VDM
personal on-demand learning

ABSTRACT

Extended Place/transition Net (EPN) is expected to be one of formal software modeling techniques to realize model-driven development. This paper shows a personal on-demand learning technique that helps engineers to acquire skills for using EPN, and then illustrates a prototype tool for it. In the technique, each engineer as a learner tries to construct his/her EPN model so as to satisfy given software requirements. The key ideas of the technique are (i) to construct a learner's EPN model by using given components only, (ii) to convert a learner's EPN model into a VDM++ specification, and (iii) to visualize the behavior of software by using animated graphics. Preliminary discussion and experiments with trial users of the prototype tool have been conducted to evaluate the effectiveness of the technique.

1. Introduction

Extended Place/transition Net (EPN)[1] is Place/transition Net (PN)[2] that includes some additional elements written in VDM++[3]. It can be used to formally model the state transition-based behavior of software in detail in development processes. A software model drawn up by using EPN is called an EPN model. It can be executed on interpreters, and will help engineers to understand and validate software specifications. Also, it can be converted into another formal software model, source codes, and test cases. Thus EPN will be useful to realize Model-Driven Development (MDD)[4] that accelerates the pace of software development. However, engineers need technical knowledge and skills to use EPN.

In order to address this problem, we show a Technique for Learning software modeling using EPN (TL-EPN), and then illustrate its prototype tool. TL-EPN is designed for learners who have already understood PN and VDM++ to some extent. In TL-EPN, a learner tries to construct his/her EPN model so as to satisfy given software requirements by using given components only. The EPN model is converted into a VDM++ specification[5] for a learner who is familiar with Vienna Development Method (VDM). Also, the behavior of software based on the EPN model is visualized by using animated graphics[6,7] for a beginning learner. The learner finally receives the result of checking the correctness of his/her EPN model. Some essential functions to support TL-EPN have been implemented in the prototype tool,

Corresponding author's E-mail: takagi@eng.kagawa-u.ac.jp, s21g457@kagawa-u.ac.jp

and its effectiveness has been evaluated through preliminary discussion and experiments.

There are a few closely related studies. A Technique for Learning software modeling using PN (TL-PN) was discussed in a previous study[7], and it provides a base of TL-EPN. Unlike TL-PN, TL-EPN is intended to support personal on-demand learning, and therefore does not include the steps of advising, review, and demonstration by instructors and other learners. Additionally, the study of TL-PN does not use EPN and VDM++. Ref. 6 shows a training support method and tool for bug fixing of EPN models. Its characteristics are to introduce animated graphics and to focus on bug fixing. The idea of the animated graphics is used also in TL-EPN. Further, there are studies on learning of other software modeling. One of the most widely used techniques in software modeling is Unified Modeling Language (UML)[8]. For example, Akayama et al.[9] discussed the effectiveness of applying MDD tools to the education of UML. Soler et al.[10] developed a web-based tool to teach class diagrams effectively in a university. Ogata et al.[11] proposed an approach to test many learners' answers efficiently in the teaching of state machine models.

This paper is organized as follows. Section 2 shows three steps of which TL-EPN consists, and then section 3 illustrates its prototype tool. Section 4 gives the results of preliminary discussion and experiments.

2. Learning Software Modeling Using EPN

This section shows the three steps of which TL-EPN consists, that is, (1) creating exercises, (2) working on exercises, and (3) checking learner's answers. (1) is for skilled engineers as instructors, and (2) and (3) are for learners who have already understood PN and VDM++ to some extent.

2.1. *Creating exercises*

In the first step, instructors create exercises for learners. Each exercise consists of (i) software requirements, (ii) a set of completed EPN models, (iii) a set of component candidates, (iv) sets of test cases, (v) animated graphics, and (vi) hints about modeling. They are basically created in this order. (iv) and (vi) are optional.

2.1.1. Software requirements

The software requirements are written in natural languages. They should include enough information for learners to construct a correct EPN model, such as detailed workflows to be supported by software, concrete data to be processed, and their constraints.

2.1.2. Set of completed EPN models

The completed EPN model is a correct answer in the exercise, and is used to check learners' answers in the last step. It should be strictly based on the software requirements. The instructors initially create one original completed EPN model. After creating a set of component candidates, they need to add equivalent completed EPN models (EPN models that are not exactly the same as the original completed EPN model but satisfy the software requirements).

2.1.3. Set of component candidates

The component candidates are used by learners for constructing their EPN models in the next step, and are classified into the following two subsets.

One is a subset of correct components. They are obtained by disassembling the original completed EPN model, as shown in Fig. 1. When the exercise is intended for beginning learners, the size of each component may be made bigger to reduce the level of its difficulty. Another is a subset of incorrect components, and they are created by mutating the correct components. Model-based mutation operators[12] can be applied to the elements of PN. Also, traditional mutation operators can be applied to the additional elements written in VDM++. When the exercise is intended for beginning learners, the subset of incorrect components may be made smaller or empty to reduce the level of its difficulty.

The instructors should confirm whether the component candidates lead to equivalent completed EPN models. In order to check learners' answers correctly in the last step, all the equivalent completed EPN models need to be found and added to the set of completed EPN models. The instructors can modify the software requirements and the component candidates so as to avoid the equivalent completed EPN models.

2.1.4. Sets of test cases

When there are many equivalent completed EPN models caused by slight differences especially in actions and guards, sets of test cases can be used instead of them. A test case in this study is a sequence of successive state transitions on an existing completed EPN model. A set of test cases is created so as to satisfy a coverage criterion[1] on the model. Each set is used to check learners' answers in the last step.

2.1.5. Animated graphics

The animated graphics consist of graphical parts, and visualize the behavior of software based on a given EPN model. Some of the graphical parts are programmed to move by trigger, such as the fire of specific transitions and the satisfaction of specific conditions in a given EPN model.

2.1.6. Hints about modeling

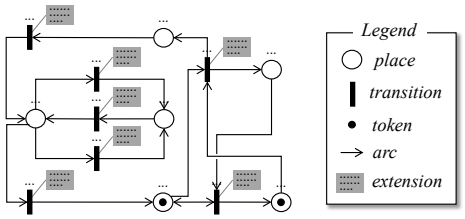
The hints about modeling are written in natural languages, and are used by learners as clues about how their EPN model can be correctly constructed. The hints are not always needed when the exercise is for advanced learners.

2.2. *Working on exercises*

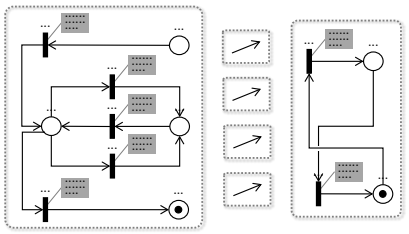
In the second step, each learner works on the exercises. The learner is given all the materials excepting the set of completed EPN models and the sets of test cases. The learner will firstly try to understand the given software requirements and hints about modeling. After that, the learner will select appropriate component candidates from the given set, and assemble them into an EPN model. The learner's EPN model is automatically converted into a VDM++ specification[5]. It will provide suitable viewpoint for learners who are familiar with VDM. Also, beginning learners can watch animated graphics to understand their EPN models intuitively[6,7]. When the learner finishes constructing his/her EPN model, he/she moves to the last step.

2.3. *Checking learner's answers*

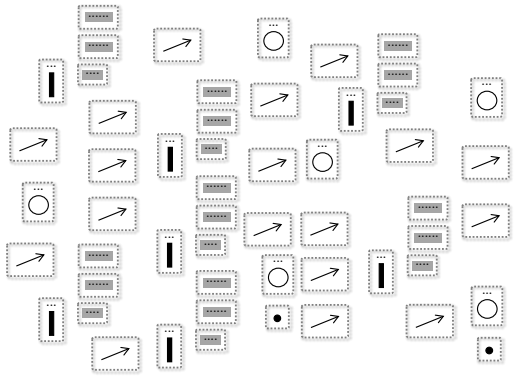
In the last step, a learner's EPN model is checked by using the set of completed EPN models and the sets of test cases. If the learner's EPN model is exactly the same as one of the completed EPN models, or if it has passed one of the



(a) Original completed EPN model



(b) Correct components for beginning learners



(c) Correct components for advanced learners

Fig. 1. Creation of a subset of correct components (overview).

sets of test cases, it is judged as a correct answer. Otherwise, the learner shall try to correct all the mistakes on his/her EPN model. If needed, some additional hints are given to the learner. For example, the information about the mistaken parts in the learner's EPN model can be used as the additional hints. When the learner gives up on constructing his/her EPN model, the original completed EPN model is disclosed to him/her as a correct answer.

3. Prototype Tool

We are developing a prototype tool for TL-EPN. The prototype tool does not fully support the steps that have

been discussed in the previous section, but includes some essential functions.

Fig. 2 is a screen shot of the prototype tool that shows a sample exercise on the subject of a simple elevator control system. The prototype tool is used on Web browsers. Its GUI chiefly consists of (A) the section for software requirements, (B) the section for hints about modeling, (C) the pane to construct a learner's EPN model, (D) the pane to select component candidates, (E) the pane to show a VDM++ specification, and (F) the pane to show animated graphics.

In (D), component candidates are classified by the kinds of elements of EPN, that is, places, transitions, tokens, guards and actions. A learner can select an arbitrary one from (D), and then can move it to (C). A VDM++ specification shown in (E) reflects a learner's EPN model given in (C). A learner can check the correctness of his/her EPN model at any time. The checking is executed as the comparison between a learner's EPN model and a completed EPN model. Its result is indicated as "O" or "X" that are the symbols for a correct/incorrect answer, respectively. If a learner's EPN model is correct, (F) shows the animated graphics of the expected behavior of the elevator. Otherwise, an additional hint (information about mistaken parts) is given to the learner so that he/she can retry the exercise. When the learner gives up on constructing his/her EPN model, a completed EPN model is shown in (C).

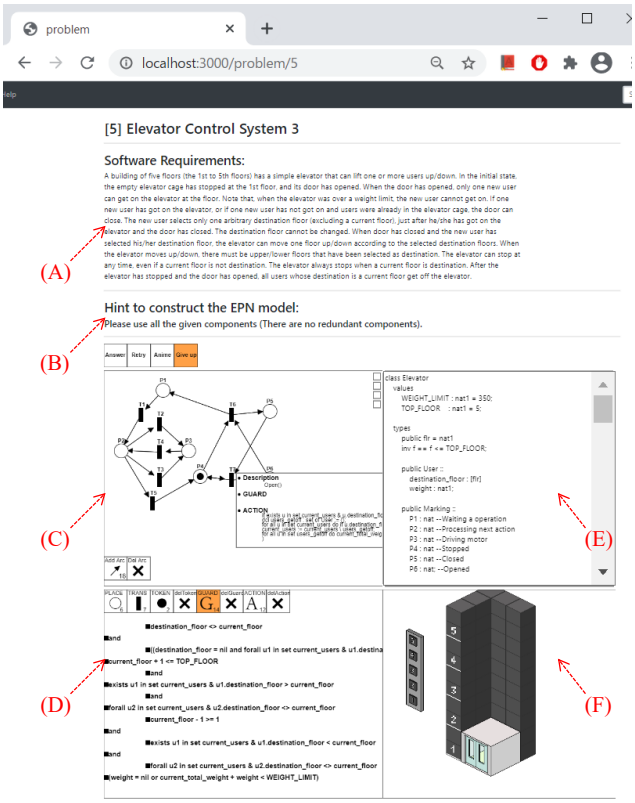


Fig. 2. Screen shot of our prototype tool.

4. Evaluation

This section shows the results of preliminary discussion for qualitative evaluation and experiments chiefly for quantitative evaluation.

We created exercises #1, #2 and #3 on the subject of a simple elevator control system, and set them into the prototype tool. The most complex exercise #3 was developed from the exercise #2, and the exercise #2 was developed from the simplest exercise #1. After that, three trial users (two master's students and one undergraduate student in our laboratory) worked on them. As a result of preliminary discussion with the trial users, we found the following:

(α) Components, VDM++ specifications, and animated graphics will be useful to support learners. However, the quality of their user interface is important for learners, and there is room to improve it.

(β) It is better to add functions that support the reading of VDM++, such as reference documents and popup-helps.

(γ) It will be often difficult for learners to directly derive a correct EPN model from given software requirements. The software requirements include little explicit representation about elements of the EPN model.

(β) and (γ) were assumed to relatively affect the effectiveness of learning. Therefore, we prepared the reference document and the explicit representation before experiments.

In the experiments, additional trial users (four undergraduate students in our laboratory) worked on the exercises in the order of #1, #2 and #3. We observed the behavior of the users, and described the bare essentials orally to one who has given up on understanding a disclosed correct answer. Also, the users answered our

questionnaire. The following were found from the results of the experiments:

- The levels of understanding of PN and VDM++ before the experiments were within a range of 3-4. They were subjectively estimated by the users themselves with a measure from 1 (not understood) to 5 (understood). This result means that the users will be appropriate for TL-EPN.
- The column (a) in Table 1 shows the rate of the users who finally answered correctly in each of the exercises. Two users could not answer correctly in the exercise #2. One of the two gave up on understanding its disclosed correct answer, and asked us for its oral description. The one would not be able to answer correctly in the following exercise #3 without the oral description. The prototype tool will need to provide some assistance for users like the one.
- The column (b) in Table 1 shows the time required to finish each of the exercises. Note that the time includes the process of retrying. The users tended to spend much time on the exercise #2, since it has a high degree of difficulty due to the great expansion from the exercise #1. The time will generally vary according to some factors, such as the degree of difficulty of each exercise, and the individual ability for software modeling.
- The column (c) in Table 1 shows the frequencies of retrying in each of the exercises. Additional hints by retrying seem to lead users to correct answers.
- The levels of having advanced in understanding of EPN by the prototype tool were within a range of 3-5 and an average of 4.0. They were subjectively estimated by the users themselves with a measure from 1 (not advanced) to 5 (advanced).
- The prototype tool provides three key functions, that is, (i) the construction using components, (ii) the conversion into VDM++ specifications, and (iii) the visualization using animated graphics. The most useful key functions for the users were (i) (3/4) and (ii) (1/4). This was subjectively estimated by the users themselves. Creation of animated graphics requires much effort, and therefore (iii) will not be cost-effective.

Table 1. Overview of experimental results.

	correct ans. rate[(a)]	**time (min.)[(b)]**		**freq. of retrying[(c)]**	
		average	**range**	**average**	**range**
#1	4/4	34	20-50	2.3	0-7
#2	2/4	45 (73)	42-49 (70-77)	4.0 (5.0)	3-5 (1-9)
#3	4/4	29	13-45	3.3	0-9

* In the columns (b) and (c) in the row #2, data of users who finally answered correctly and of other users are shown separately. The latter data are put in parentheses.

5. Conclusion

In this paper, we showed TL-EPN, and then illustrated its prototype tool. TL-EPN consists of three steps, that is, (1) creating exercises, (2) working on exercises, and (3) checking learner's answers. Its key ideas are (i) the construction using components, (ii) the conversion into VDM++ specifications, and (iii) the visualization using animated graphics. Some essential functions to support TL-EPN have been implemented in the prototype tool. As a result of preliminary discussion with its trial users, we found that components, VDM++ specifications, and animated graphics would be useful to support learners. Also, as a result of experiments with additional trial users, we found that the function for (i) would be the most useful for users. However, there are some challenges to be addressed in future. For example, the prototype tool will need to provide some assistance for users who gave up on understanding a disclosed correct answer.

In a future study, we plan to extend TL-EPN and its prototype tool, and then conduct experiments to evaluate their effectiveness.

Acknowledgements

The authors thank Mr. Y. Ue and Mr. S. Morimoto for their efforts to develop prototype tools in previous studies. This work was supported by JSPS KAKENHI Grant Number JP17K00103.

References

1. 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*, Kyoto, Japan, pp.29-30, Dec. 2019.
2. N.G. Leveson and J.L. Stolzy, Safety Analysis Using Petri Nets, *IEEE Transactions on Software Engineering*, Vol.13, No.3, IEEE, United States, pp.386-397, Mar. 1987.
3. J. Fitzgerald, P.G. Larsen, P. Mukherjee, N. Plat and M. Verhoef, *Validated Designs for Object-Oriented Systems*, Springer-Verlag London, 2005.

4. D.S. Frankel, *Model Driven Architecture - Applying MDA to Enterprise Computing*, John Wiley & Sons, United States, 2003.
5. T. Takagi and R. Kurozumi, Prototype of a Modeling Tool to Convert between Extended Place/Transition Nets and VDM++ Specifications, *Proceedings of International Conference on Artificial Life and Robotics*, ALife Robotics, Oita, Japan, pp.157-160, Jan. 2019.
6. T. Takagi, S. Morimoto, Y. Ue and Y. Imai, Animated Graphics-based Training Support Method and Prototype Tool for Bug Fixing of Extended Place/Transition Nets, *Journal of Robotics, Networking and Artificial Life*, Vol.5, No.4, pp.278-282, Mar. 2019.
7. Y. Ue and T. Takagi, Learning Support Technique of Software Visual Modeling Using Place/Transition Nets, *Proceedings of International Conference on Artificial Life and Robotics*, ALife Robotics, Oita, Japan, pp.751-754, Jan. 2020.
8. Object Management Group, *Unified Modeling Language*, https://www.uml.org/.
9. S. Akayama, K. Hisazumi, S. Hiya and A. Fukuda, Using Model-Driven Development Tools for Object-Oriented Modeling Education, *Proceedings of Educators' Symposium*, Florida, United States, 8 pages, Sep. 2013.
10. J. Soler, I. Boada, F. Prados, J. Poch and R. Fabregat, A Web-based E-learning Tool for UML Class Diagrams, *Proceedings of EDUCON Education Engineering*, Madrid, Spain, pp.973-979, Apr. 2010.
11. S. Ogata, M. Kayama and K. Okano, Approach to Testing Many State Machine Models in Education, *Proceedings of International Conference on Computer Supported Education*, Crete, Greece, pp.481-488, May 2019.
12. T. Takagi, R. Takata, Z. Furukawa, F. Belli and M. Beyazıt, Metrics for Model-Based Mutation Testing Based on Place/Transition Nets, *Proceedings of Joint Conference of International Workshop on Software Measurement and International Conference on Software Process and Product Measurement*, Nara, Japan, pp.7-10, Nov. 2011.

==========

Authors Introduction

Dr. Tomohiko Takagi

He received the B.S., M.S. and Ph.D. 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. Akio Usuda

He received the B.S. degree from Kagawa University in 2021. 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.

==========