Development of an Early Prototype Tool for Learning Software Modeling Using Extended Place/Transition Net

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

Akio Usuda
Faculty of Engineering, Kagawa University
2217-20 Hayashi-cho, Takamatsu-shi, Kagawa 761-0396, Japan
E-mail: takagi@eng.kagawa-u.ac.jp, s17t213@stu.kagawa-u.ac.jp

Abstract

This paper shows an early prototype tool for learning software modeling using Extended Place/transition Net (EPN), and then gives the discussion about its effectiveness and challenges. A user of the tool, that is, an engineer tries to construct his/her EPN model based on given software requirements by selecting and putting the given components of EPN. The EPN model is converted into a VDM++ specification for a user who is familiar with Vienna Development Method (VDM). Also, the behavior of software based on the EPN model is partially visualized by using animated graphics for a user who is a learner at the first stage. In the end, the correctness of the user's EPN model is automatically checked.

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

1. Introduction

Extended Place/transition Net (EPN)[1] is Place/transition Net (PN) that includes some additional elements written in VDM++[2] to enhance its representation power, and can be used to formally model the state transition-based behavior of software in development processes. An EPN model, that is, a software model drawn by using EPN, can be executed on interpreters to understand and validate software specifications, and also can be converted to another formal software specification, source codes, and test cases. However, the use of EPN is based on technical knowledge and skills, and therefore engineers will need to learn them.

This paper shows an early prototype tool for learning software modeling using EPN. A user of the tool, that is, an engineer tries to construct his/her EPN model based on given software requirements by selecting and putting the given components of EPN. The EPN model is converted into a VDM++ specification[3] for a user who is familiar with Vienna Development Method (VDM). Also, the behavior of software based on the EPN model is partially visualized by using animated graphics[4,5] for a user who is a learner at the first stage. In the end, the correctness of the user's EPN model is automatically checked. This study is intended to support personal on-demand learning, and the key ideas of this study are (1) the construction using components, (2) the conversion into VDM++ specifications, and (3) the visualization using animated graphics.

This paper is organized as follows. Section 2 shows the steps to learn software modeling using EPN in this study. Section 3 illustrates the early prototype tool we are developing in this study, and then section 4 gives the discussion about its effectiveness and challenges.

2. Learning Software Modeling Using EPN

This section shows the three steps to learn software modeling using EPN in this study, that is, (1) creating exercises, (2) working on exercises, and (3) checking learner's answers. They have been developed based on our previous study[5].

2.1. *Creating exercises*

In the first step, instructors, that is, skilled engineers create exercises for learners. Each exercise consists of (i) software requirements, (ii) a completed EPN model, (iii) component candidates, (iv) animated graphics, and (v) hints to construct EPN models. They are generally created in this order.

2.1.1. *Software requirements*

The software requirements are written in natural languages. They should include enough information to construct a correct EPN model in the next step, such as the use case scenarios, state transitions, data processing, and constraints of the software.

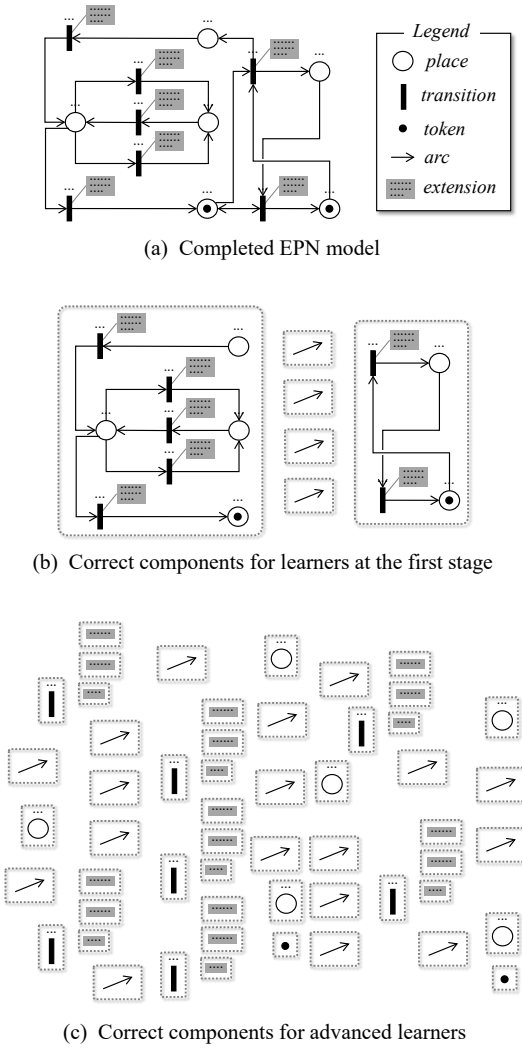
2.1.2. *Completed EPN model*

The completed EPN model is a correct answer in the exercise, and will be used to check learner's answers in the last step. It should be strictly based on the software requirements.

2.1.3. *Component candidates*

The component candidates that will be used by learners for constructing their EPN models in the next step are classified into the following two sets.

One is the set of correct components, and they are obtained by disassembling the completed EPN model, as shown in Fig. 1. When the exercise is intended for learners at the first stage, the size of each component may be made bigger in order to reduce the level of its difficulty. Another is the set of incorrect components, and they are created by mutating the correct components. Model-based mutation operators[6] can be applied to the elements of PN (that is, places, transitions, arcs, and tokens), and traditional mutation operators can be applied to the extension from PN to EPN (that is, actions and guards written in VDM++). When the exercise is intended for learners at the first stage, the set of incorrect components may be made smaller or empty in order to reduce the level of its difficulty.



(a) Completed EPN model

(b) Correct components for learners at the first stage

(c) Correct components for advanced learners

Fig. 1. Creation of the set of correct components (overview).

The instructors should confirm whether the component candidates lead to any other correct answers, that is, EPN models that do not have the same structure as the completed EPN model but satisfy the software requirements.

2.1.4. *Animated Graphics*

The animated graphics consist of several 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.

2.1.5. *Hints to construct EPN models*

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

2.2. *Working on exercises*

In the second step, learners work on the exercises the instructors have created. The learners are given all the materials excepting the completed EPN model. They will firstly try to understand the given software requirements and hints to construct EPN models. After that, they will select arbitrary components from the given component candidates in order to add onto their EPN models. The learners' EPN models under construction are automatically converted into VDM++ specifications[3], and learners who are familiar with VDM will review the VDM++ specifications in order to confirm their EPN models from another viewpoint. Also, learners at the first stage will watch animated graphics in order to confirm their EPN models intuitively[4,5]. When learners finish constructing their EPN models, they move to the last step.

2.3. *Checking learner's answers*

In the third (that is, the last) step, an EPN model that has been constructed by a learner in the second step is compared to the completed EPN model that has been constructed by the instructors in the first step. If they are exactly the same, it is concluded that the learner has successfully constructed the correct EPN model. If they are not the same, the learner shall try to correct all the mistakes on his/her EPN model. If needed, the learner is given additional hints to construct the correct EPN model, such as the information about the mistaken parts in his/her EPN model and the unsatisfied items in the given software requirements.

3. Early Prototype Tool

In this study, we are developing an early prototype tool for learning software modeling using EPN. The tool does not fully support the steps that have been discussed in the previous section, but includes some essential functions.

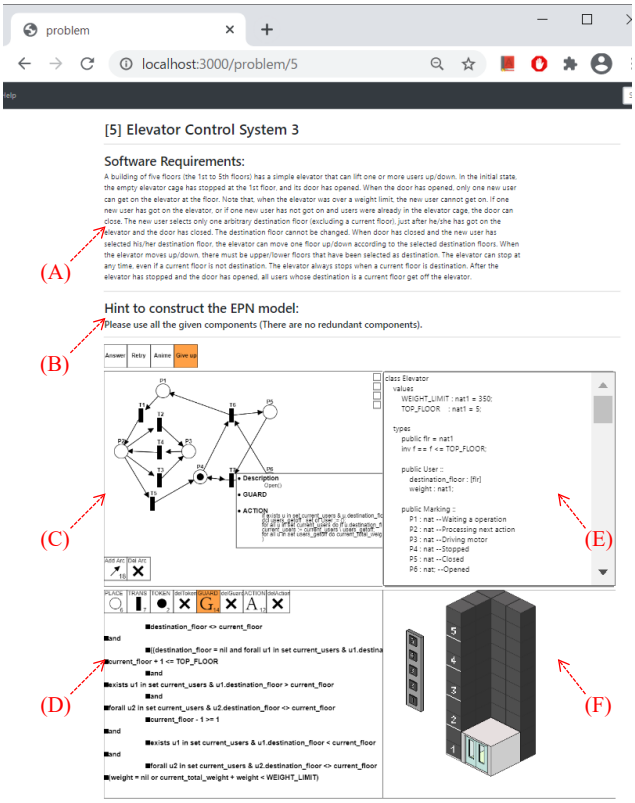


Fig. 2. Screen shot of our early prototype tool.

Fig. 2 is the screen shot of the tool that shows a sample exercise on the subject of a simple elevator control system. The tool is used on a Web browser. Its GUI chiefly consists of (A) the description of software requirements, (B) the description of hints to construct EPN models, (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 can move it to (C). Components that have been moved from (D) can be returned to (D) by the learner. When a learner's EPN model is changed, its VDM++ specification shown in (E) is automatically updated in order to help a learner who is familiar with VDM to confirm his/her EPN model from another viewpoint. Also, in order to help a learner at the first stage to confirm his/her EPN model intuitively, (F)

is intended to show the animated graphics of the elevator of which behavior is based on his/her EPN model. A learner can check the correctness of his/her EPN model, that is, can perform the automatic comparison between his/her EPN model and the correct EPN model at any time. The result of checking the correctness is indicated as "O" or "X" that are the symbols for a correct/incorrect answer, respectively. When a learner's EPN model is incorrect, the messages about mistakes may be given to the learner in order that he/she retries to the exercise. When the learner gives up constructing his/her EPN model, the completed EPN model is shown in (C).

4. Discussion

As shown in the previous sections, the key ideas of this study are (1) the construction using components, (2) the conversion into VDM++ specifications, and (3) the visualization using animated graphics. (1) will be useful to adjust the level of the difficulty of exercises. (2) will help learners who are familiar with VDM to confirm their EPN models from another viewpoint. (3) will help learners at the first stage to confirm their EPN models intuitively. However, through the development of the early prototype tool, we found the following challenges to be addressed in future:

- As with the method of Ref. 4, the creation of exercises, particularly the creation of animated graphics will require hard effort, and authoring tools should be introduced to support it.
- The animated graphics will not be so easy to quickly reveal learner's mistakes included in a large and complex EPN model. Optimized test case generation techniques may need to be introduced to solve it.
- Component candidates often lead to other correct answers. Instructors will require some techniques and tools to avoid the other correct answers or to confirm whether the set of their completed EPN models covers all the possible correct answers.

There are some closely related works. For example, a learning support technique for software modeling using PN was discussed in our previous study[5]. The steps discussed in section 2 have been developed based on it. However, unlike the previous study, this study 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. Also, the previous study does not include the discussion about tools, and does not take EPN and VDM as objects of study. Ref. 4 shows a training support method and tool for bug fixing of EPN models. The characteristics of this previous study are to introduce animated graphics and to focus on bug fixing. The idea of the animated graphics is used also in this study.

5. Conclusion

In this paper, we showed an early prototype tool for learning software modeling using EPN, and then gave the discussion about its effectiveness and challenges. The key ideas of this study are (1) the construction using components, (2) the conversion into VDM++ specifications, and (3) the visualization using animated graphics. In a future study, we plan to develop the prototype tool and conduct preliminary experiments to improve our method and tool.

Acknowledgements

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*, pp.29-30, Dec. 2019.
2. J. Fitzgerald, P.G. Larsen, P. Mukherjee, N. Plat and M. Verhoef, *Validated Designs for Object-Oriented Systems*, Springer-Verlag London, 2005.
3. 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*, pp.157-160, Jan. 2019.
4. 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.
5. 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*, pp.751-754, Jan. 2020.
6. 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*, pp.7-10, Nov. 2011.