Learning Support Technique of Software Visual Modeling Using Place/Transition Nets

Yuki Ue
Graduate School of Engineering, Kagawa University
2217-20 Hayashi-cho, Takamatsu-shi, Kagawa 761-0396, Japan

Tomohiko Takagi
Faculty of Engineering and Design, Kagawa University
2217-20 Hayashi-cho, Takamatsu-shi, Kagawa 761-0396, Japan
E-mail: s18g459@stu.kagawa-u.ac.jp, takagi@eng.kagawa-u.ac.jp

Abstract

This paper shows a learning support technique of software visual modeling. The ideas of block and animated graphics are introduced into this technique in order that learners can intuitively understand the notation and behavior of models, and can be given guidance about the way to construct proper models. Place/transition net is selected as a modeling language in this paper, but other formal behavioral modeling languages based on directed graphs also can be introduced into this technique.

Keywords: software modeling, visual modeling, learning support, place/transition net

1. Introduction

Software modeling is important especially for the development of large and complex software. For example, UML (Unified Modeling Language), state machines, and Petri nets are well known as modeling languages to construct software models, and they have been widely used in the actual development processes of requirements definition, design, and testing. However, the quality of software models depends on the skill of engineers, and learning it costs time and effort.[1]

We propose a learning support technique of software visual modeling to address this problem. The ideas of block and animated graphics, which are well-known in the field of visual programming,[2] are introduced into this technique in order that learners can understand the notation and behavior of models intuitively, and can be given guidance about the way to construct proper models. PN (Place/transition Net) that is a kind of Petri net[3] is selected as a modeling language in this paper. However, other formal behavioral modeling languages based on directed graphs also can be introduced into this technique, since the formal behavioral modeling languages based on directed graphs including PN have the following common characteristics.

- The action of software is associated with nodes and/or arcs.
- The execution order of the action is determined by the arcs.

In this paper, we show the basic concept of visual modeling in section 2, and propose the learning support technique of the visual modeling using PN in section 3. Last section gives conclusion and future work.

2. Basic Concept of Visual Modeling

The visual modeling in this paper is developed from the existing visual programming languages of block type, such as Scratch and Blockly. In the visual programming, programs can be easily created by connecting visual objects that are called blocks. The block describes an instruction to perform specific action, such as to move a cartoon character, to make a sound, and so on. Thus the visual programming is one of effective learning methods for beginners in programing.[2]

Our visual modeling is aimed at supporting the intuitive understanding of the notation (syntax) and behavior (semantics) of PN models. PN is one of formal behavioral modeling languages that are based on directed graphs. In general, formal modeling languages are useful to construct unambiguous and executable definitions, but it will not be so easy for beginners to learn them. Therefore, we introduce the following two key ideas of the visual programming into our visual modeling.

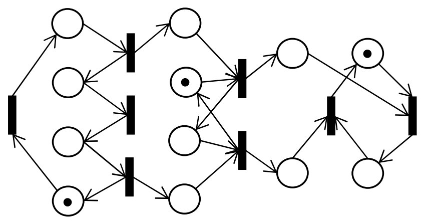
- The structural elements in a program are defined as blocks, and the blocks can be connected with each other only if they are not syntactically wrong. Therefore, it is expected that beginners who are not familiar with syntax can intuitively make their programs.
- The behavior of a program is visualized as animated graphics, and thus it is expected that beginners who are not familiar with semantics can intuitively understand the behavior of their programs.

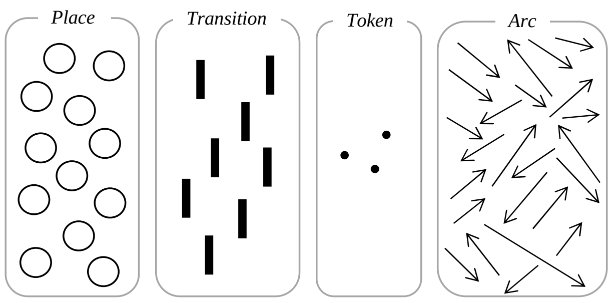
In our visual modeling, the structural elements of a PN model (that is, places, transitions, tokens, and arcs) are defined as blocks in which the notation rules of PN models are implemented. For example, each place can hold one or more tokens, and it must be connected with one or more transitions by arcs, and so on. As in the case of the visual programming, it is expected that beginners who are not familiar with the notation rules of PN models can intuitively make their PN models.

Animated graphics in our visual modeling are used to visualize the behavior of a PN model under construction. If the PN model is wrong (that is, it does not satisfy software requirements), its animated graphics do not move correctly. For example, the animated graphics of a wrong PN model of a crossing gate control system will show that a crossing gate does not work when a train has reached the grade crossing. As in the case of the visual programming, it is expected that beginners who are not familiar with the semantics of PN models can intuitively understand the behavior of their PN models.

3. Learning Support Technique

In this section, we propose the learning support technique of our visual modeling. This technique consists of (1) creation of exercises, (2) work on exercises, (3) review, (4) answer checking, and (5) demonstration. It is assumed that PN is used as a modeling language, and expected learners are beginners in software modeling using PN. This technique should be implemented as a Web application in order to enable instructors and learners to interact with each other.


(a) PN model



(b) A set of blocks that are classified into places, transitions, tokens, and arcs

Fig. 1. Example of a PN model and a set of blocks (abstracted illustration).

3.1. *Creation of Exercises*

Instructors who are expert in software modeling using PN create exercises for learners. Each exercise consists of software requirements written in natural languages, a correct PN model as an answer, a set of blocks of which the correct PN model consists, and animated graphics.

The instructors construct the correct PN model based on the software requirements, and then the set of blocks is constructed by disassembling the correct PN model, as shown in Fig. 1. The set of blocks in Fig. 1 (b) is constructed by disassembling the PN model in Fig. 1 (a). Each block corresponds to a place, transition, token, or arc, and it is given the notation rules. Also, blocks of places and transitions are labeled with state names and event names, respectively (Note that the labels are omitted in Fig. 1, since it is an abstracted illustration). Some wrong blocks, that is, blocks that should not be included in the correct PN model can be added to the set of blocks, in order to increase the difficulty of an exercise. On the contrary, if the difficulty needs to be decreased, a semi-finished PN model and a small set of blocks that are used to finish it are prepared instead of the full set of blocks.

Fig. 2 shows an example of animated graphics of a crossing gate control system. The animated graphics consist of multiple graphical components, and the blocks include the internal programs to control the detailed behavior of the graphical components. For example, the animated graphics of a crossing gate control system will contain the graphical components of a train, crossing gate bars, crossing alarms, and so on. The instructors need to prepare the graphical components, integrate them into the animated graphics, and write the internal programs. The internal programs usually need not to be written by learners, and are hidden from the learners. However, if an internal program includes some statements that are essential to specify the behavior of software, the instructors need to translate the statements into a compact sentence in a natural language. A block including an internal program including essential statements is labeled with their compact sentence, and thus learners can understand them. The instructors can introduce a blank problem style by removing an important word or value from the compact sentence.

A newly created exercise is added to the collection of exercises that is open to all learners.

3.2. *Work on Exercises*

Each learner selects an exercise from the collection of exercises based on the domain and difficulty. The domain includes information systems, embedded software, and so on. The difficulty can be determined based on the size of the given set of blocks, the complexity of the correct PN model (for example, the number of branches on the reachability graph of the correct PN model), and so on.

First a learner reads the given software requirements, and then tries to construct his/her PN model that satisfies them. The learner selects a block from the given set of blocks, and puts it on a workspace. The learner is allowed only the connection among blocks that satisfies the notation rules of PN models. The learner can remove an arbitrary block that has already been connected with other blocks from the workspace, and then return it to the set of blocks. However, the learner is not allowed to disassemble a semi-finished PN model that has been given by instructors of the exercise.

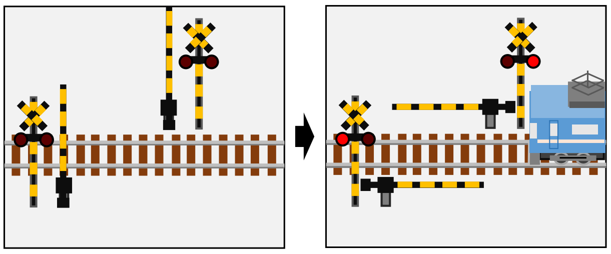


Fig. 2. Example of animated graphics.

The animated graphics are played according to the internal programs of blocks of which learner's PN model under construction consists, and give the learner a better understanding of its behavior. Additionally, if the learner cannot construct his/her PN model due to the high degree of the difficulty of the exercise, he/she can call in advisers, that is, ask instructors and other learners for their advice. The advisers can send their comments at any time, if the learner permits it.

3.3. *Review*

Before performing answer checking, the learner can call in reviewers, that is, ask instructors and other learners to review his/her PN model. The review comments from the reviewers will be useful not only to understand the modeling technique but also to maintain the motivation for learning. If the learner finds any errors in his/her PN model by the review comments, he/she tries to remove the errors. The reviewers can send their review comments at any time, if the learner permits it.

3.4. *Answer Checking*

When the learner has finished constructing his/her PN model, he/she performs answer checking, that is, his/her PN model is automatically compared to the correct PN model given by instructors. If they are not the same, it is wrong answer and the learner tries to remove his/her errors.

In general, a PN model consists of multiple sub-PN models that correspond to components, modules, or subsystems of software. For example, a PN model of a crossing gate control system will consist of sub-PN models that represent the behavior of a train, a crossing gate, and a controller. A sub-PN model that includes errors can be highlighted, in order to give the learner a clue to assist in removing the errors.

The learner can call in advisers or reviewers, and discuss with them.

3.5. *Demonstration*

If many learners cannot construct their PN models correctly in a specific exercise, an instructor who has created it gives a demonstration of how to construct a correct PN model for the learners.

4. Conclusion and Future Work

In this paper, we proposed a learning support technique of software visual modeling using PN in order that learners can intuitively understand the notation and behavior of a PN model. The ideas of block and animated graphics, which are well-known in the field of visual programming, are introduced into this technique. This technique consists of (1) creation of exercises, (2) work on exercises, (3) review, (4) answer checking, and (5) demonstration. It is assumed that learners are beginners in software modeling using PN. This technique should be implemented as a Web application in order to enable instructors and learners to interact with each other. The instructors' workload will not be small especially in the creation of exercises, and should be decreased by additional techniques and tools.

This technique is also related to our previous study. Ref. 4 shows a training support method for removing an error from a faulty EPN (Extended PN) model. Same as this technique, the training support method in Ref. 4 introduces animated graphics to illustrate the behavior of an EPN model, and thus this technique can be extended so as to support EPN. In EPN, a text-based formal modeling language is used to represent the essential and detailed behavior of software instead of a natural language, and visual programming of block type will be applicable to it.

In future work, we plan to develop a prototype tool to support our technique, and will evaluate its learning effect by using it.

References

1. M. Petre, UML in practice, *Proceedings of International Conference on Software Engineering*, May 2013, pp.722-731.
2. D. Bau, J. Gray, C. Kelleher, J. Sheldon and F. Turbak, *Learnable Programming: Blocks and Beyond*, Communications of the ACM, Vol.60, No.6, June 2017, pp.72-80.
3. N.G. Leveson and J.L. Stolzy, Safety Analysis Using Petri Nets, *IEEE Transactions on Software Engineering*, Vol.13, No.3, Mar. 1987, pp.386-397.
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, Mar. 2019, pp.278-282.