An Improved Conversion Technique from EPNAT Models to VDM++ Specifications for Simulation of Abstract Software Behavior

Sho Matsumoto
Graduate School of Science for Creative Emergence, Kagawa University
2217-20 Hayashi-cho, Takamatsu-shi, Kagawa 761-0396, Japan

Ryoichi Ishigami
Graduate School of Science for Creative Emergence, Kagawa University
2217-20 Hayashi-cho, Takamatsu-shi, Kagawa 761-0396, Japan

Tetsuro Katayama
Department of Computer Science and Systems Engineering, Faculty of Engineering, University of Miyazaki
1-1 Gakuen-kibanadai nishi, Miyazaki-shi, Miyazaki 889-2192, Japan

Tomohiko Takagi
Department of Engineering and Design, Faculty of Engineering and Design, Kagawa University
2217-20 Hayashi-cho, Takamatsu-shi, Kagawa 761-0396, Japan
Email: s23g364@kagawa-u.ac.jp, takagi@eng.kagawa-u.ac.jp

Abstract

Formal software models based on EPNAT (Extended Place/transition Net with Attributed Tokens) can be converted to VDM++ specifications that enable simulation of abstract software behavior before implementation processes. However, the conversion technique has two problems, that is, (1) extracting all properties to be checked from the VDM++ specifications requires time and effort, and (2) the structure of the VDM++ specifications has less readability and maintainability. In this study, we improve the conversion technique by (1) adding features to extract an abstract current state of software, and (2) dividing into classes that correspond to subnets of EPNAT models. This paper shows a new conversion rule, a new structure of VDM++ specifications, a simple example, and the discussion about their effectiveness.

Keywords: Software model, Place/transition net, Vienna development method, Conversion technique

1. Introduction

Formal software models are useful to define specifications and designs clearly, and to find failures early in software development. EPNAT (Extended Place/transition Net with Attributed Tokens), which is based on place/transition nets and VDM++, has been proposed as a formal software modeling language especially for concurrent and distributed software that is composed of multiple subsystems (and/or systems) [1]. In an EPNAT model (that is, a formal software model that is defined by using EPNAT), the behavior of each subsystem is defined as a subnet, and the interactions between the subsystems are defined as glue transitions. EPNAT models can be converted to VDM++ specifications, and enable simulation of abstract software behavior before implementation processes [2].

However, the conversion technique has two problems. One is that extracting all properties (that is, data to be checked in the simulation) from the VDM++ specifications requires time and effort. In the simulation, a common VDM++ interpreter [3] is used to execute the VDM++ specifications, and moreover, a particular feature of our support tool to manage its original EPNAT model needs to be used to process and obtain some properties. Another is that the structure of the VDM++ specifications has less readability and maintainability. The structure is not appropriately divided into modules that correspond to the elements of its original EPNAT model.

To tackle the above problems, we propose an improved conversion technique from EPNAT models to VDM++ specifications. In this technique, features to extract an abstract current state of software are added to the VDM++ specifications in order to tackle the first problem. Additionally, the structure of the VDM++ specifications is divided into classes that correspond to subnets of EPNAT models in order to tackle the second problems. In general, the important things in the conversion of

software models are correctness and automation [4], and thus we try to keep or improve them in this study. This paper shows a new conversion rule, a new structure of VDM++ specifications, and a simple example in section 2. The discussion about their effectiveness is given in section 3, and then conclusion and future work are shown in section 4.

2. Conversion Technique

The improvement of the previous technique to convert from EPNAT models to VDM++ specifications consists of introducing the features to obtain marking data (discussed in section 2.1) and the basic class structure of the VDM++ specifications (discussed in section 2.2). The former is proposed for tackling the first problem, and the latter is proposed for tackling the second problem.

For the discussion in this study, we have constructed an experimental EPNAT model of BSS (imaginary Business Support Software) that is composed of a purchase subsystem, a production subsystem, and a stock management subsystem. Its overview is shown in Fig. 1.

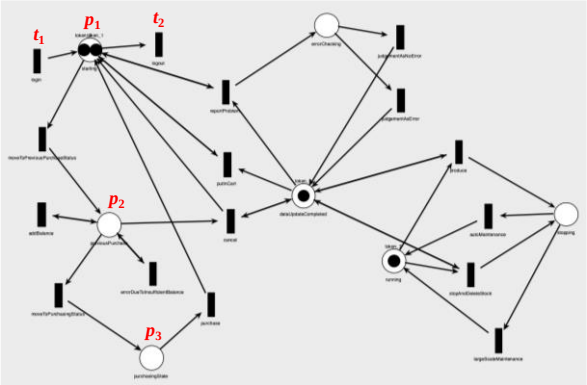

Fig. 1. Overview of an experimental EPNAT model.

2.1. *Features to obtain marking data*

The features to extract an abstract current state of software should be added to the VDM++ specifications. In EPNAT models, the abstract states are expressed as the distributions of attributed tokens, that is, markings. Therefore, in this technique, the features to obtain original markings and partitioned markings are added to the VDM++ specifications. The idea of the partitioned markings has been derived from equivalence partitioning testing and coverability trees [5].

An original marking is the list of the numbers of attributed tokens in every place. For example, the original marking in the purchase subsystem of BSS (Fig. 1) is expressed as $[p_1, p_2, p_3] = [2, 0, 0]$. In general, there is a huge number of original markings in an EPNAT model, and it is not so easy to cover all of them in the simulation. In partitioned markings, the ranges of the numbers of attributed tokens in which software seems to have the same behavior are expressed as symbols (the ranges are defined by engineers). A symbol represents all the numbers of attributed tokens in a corresponding range, and therefore the number of markings to be covered can be reasonably reduced. For example, in the purchase subsystem of BSS (Fig. 1), an execution path by using original markings [1, 0, 0]->t_1->[2, 0, 0]->t_1->[3, 0, 0]->t_2->[2, 0, 0]->t_2->[1, 0, 0] can be converted into an execution path by using partitioned markings [1, 0, 0]->t_1->[w, 0, 0]->t_1->[w, 0, 0]->t_2->[w, 0, 0]->t_2->[1, 0, 0], when the number of attributed tokens in p_1 in the range of two or more is represented as a symbol w. In this example, the original markings [2, 0, 0] and [3, 0, 0] are replaced with the partitioned marking [w, 0, 0].

In the simulation, if possible, a search algorithm should avoid to revisit the same partitioned markings to improve its efficiency. The features to obtain marking data will be frequently used in the simulation, since the marking data is important properties to capture the abstract software behavior. The way of adding the features to VDM++ specifications is discussed in the next section.

2.2. *Basic class structure of VDM++ specifications*

The structure of VDM++ specifications should be divided into modules that correspond to the elements of its original EPNAT model. Therefore, the basic class structure of VDM++ specifications is constructed in this study.

The basic class structure is shown in Fig. 2. It is composed of five kinds of classes, that is, “Software”, “Subsystem”, “AttributedToken”, “Places”, and “Marking”. “Subsystem” and “AttributedToken” are the sets of classes for specifying each subsystem. In each of the sets, there are N classes (N expresses the number of subsystems), and one class peculiar to each subsystem needs to be defined in the VDM++ specifications. The others are classes for managing the data of all of the subsystems. In the VDM++ specifications, they need to be defined as one class, respectively.

“Subsystem” is a part of “Software”, and corresponds to the subnets of the original EPNAT model. There are relations between “Subsystem$_i$” and “AttributedToken$_i$”, and also there are relations between “AttributedToken$_i$” and “Places” ($1 \leq i \leq N$; $ClassName_i$ expresses the i-th class in the set $ClassName$). They mean that each subsystem keeps corresponding attributed tokens in its own places. There are relations between “Marking” and “Subsystem$_i$”, which means that original markings and partitioned markings are derived from the places of subsystems. The operations that correspond to glue transitions are defined in “Software”, since they have a role in the interaction between different subsystems. The other transitions (hereinafter, referred to as normal transitions) are peculiar to each subsystem, and thus the operations that correspond to normal transitions are

defined in corresponding "Subsystem$_i$". Executing the operations of the transitions changes the current marking, which corresponds to arcs. Therefore, the VDM++ specifications have no explicit elements that correspond to arcs.

Fig. 3 shows the overview of the VDM++ specifications of the experimental EPNAT model that are based on the basic class structure. The classes "PurchaseSubsystem", "StockManagementSubsystem", and "ProductionSubsystem" correspond to "Subsystem" of the basic class structure. The classes "Customer", "Goods", and "ProductionMachine" correspond to "AttributedToken" of the basic class structure.

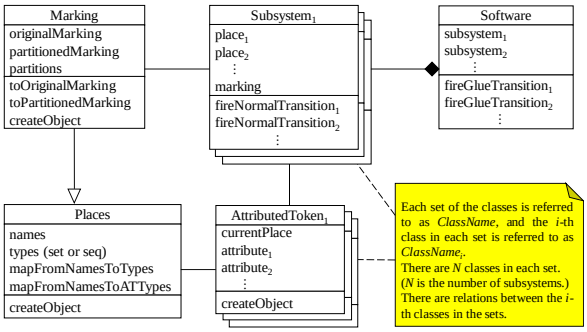

Fig. 2. Overview of the basic class structure.

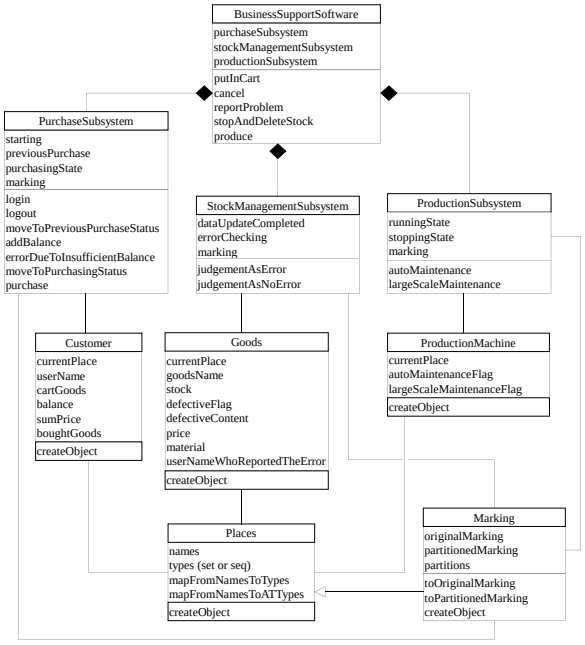

Fig. 3. Overview of VDM++ specifications of the experimental EPNAT model based on the basic class structure.

3. Discussion

This section gives the discussion about the effectiveness of this technique. We have manually converted the experimental EPNAT model to VDM++ specifications by using this technique. Also, we have converted the experimental EPNAT model to VDM++ specifications by using the previous technique. Hereinafter, the former is referred to as improved specifications, and the latter is referred to as previous specifications. We have compared the improved specifications and the previous specifications, and have found the followings.

- The improved specifications can additionally provide marking data (original markings and partitioned markings). Also, the improved specifications can provide the information about individual attributed tokens by subsystems, and variables by subsystems. These will be useful to search the VDM++ specifications for the failures of software in the simulation.
- The elements of EPNAT models correspond to the elements of the improved specifications. There is traceability between EPNAT models and the improved specifications to some extent. Additionally, the improved technique can avoid the errors that the actions of normal transitions of a subsystem try to access the elements of other subsystems, since subsystems are separately defined as classes in the improved specifications.
- The improved specifications and the previous specifications consist of about 440 and 280 lines of code in VDM++, respectively. The previous specifications do not have the features to obtain marking data discussed in section 2.1, and the improved specifications excluding the features consist of about 340 lines of code. In any case, the improved specifications are larger than the previous specifications.

4. Conclusion and Future Work

In this paper, we have proposed an improved conversion technique from EPNAT models to VDM++ specifications for the simulation of abstract software behavior. The improvement consists of (1) introducing the features to obtain marking data and (2) introducing the basic class structure of the VDM++ specifications. Regarding (1), the features will be frequently used in the simulation, since the marking data is important properties to capture the abstract software behavior. We have concentrated on obtaining marking data, but other properties also need to be collected to be efficiently checked in the simulation. The features to collect other properties will be considered in future work. Regarding (2), the basic class structure establishes traceability between EPNAT models and VDM++ specifications to some extent, and avoids a certain kind of errors. However, VDM++ specifications by the improved conversion technique are larger than VDM++ specifications by the previous conversion technique. The basic class structure

may need to be refined further to improve the readability and maintainability in future work.

Acknowledgements

This work was supported by JSPS KAKENHI Grant Number JP22K11976, and Young Scientists Fund of Kagawa University Research Promotion Program 2021 (KURPP).

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. S. Matsumoto, T. Katayama and T. Takagi, Automated Random Simulation for Checking a Behavioral Model of Systems Based on Extended Place/Transition Net with Attributed Tokens, *Proceedings of International Conference on Artificial Life and Robotics*, online, Japan, pp.337-340, Feb. 2023.
3. J. Fitzgerald, P.G. Larsen, P. Mukherjee, N. Plat and M. Verhoef, *Validated Designs for Object-Oriented Systems*, Springer-Verlag London, 2005.
4. K. Lano and S. Kolahdouz-Rahimi, Model Transformation Specification and Design, *Advances in Computers*, Vol.85, pp.123-163, Elsevier, 2012.
5. T. Takagi and Z. Furukawa, Test Case Generation Technique Based on Extended Coverability Trees, *Proceedings of 13th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, IEEE, Kyoto, Japan, pp. 301-306, Aug. 2012.

Authors Introduction

Mr. Sho Matsumoto

He received the B.S. degree from Kagawa University in 2023. He is a master's student in the Graduate School of Science for Creative Emergence at Kagawa University. His research interests are in software engineering, particularly software quality control.

Mr. Ryoichi Ishigami

He received the B.S. degree from Kagawa University in 2023. He is a master's student in the Graduate School of Science for Creative Emergence at Kagawa University. His research interests are in software engineering, particularly software quality control.

Dr. Tetsuro Katayama

He received a Ph.D. degree in engineering from Kyushu University, Fukuoka, Japan, in 1996. From 1996 to 2000, he has been a Research Associate at the Graduate School of Information Science, Nara Institute of Science and Technology, Japan. Since 2000 he has been an Associate Professor at the Faculty of Engineering, Miyazaki University, Japan. He is currently a Professor with the Faculty of Engineering, University of Miyazaki, Japan. His research interests include software testing and quality. He is a member of the IPSJ, IEICE, and JSSST.

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.