Journal of Robotics, Networking and Artificial Life
Vol. 7(3); December (2020), pp. 165–169
DOI: https://doi.org/10.2991/jrnal.k.200909.005; ISSN 2405-9021; eISSN 2352-6386
https://www.atlantis-press.com/journals/jrnal

Research Article

Proposal of an Algorithm to Generate VDM++ Specification Based on its Grammar by Using Word Lists Extracted from the Natural Language Specification

Tetsuro Katayama[1,*], Yasuhiro Shigyo[1], Yoshihiro Kita[2], Hisaaki Yamaba[1], Kentaro Aburada[1], Naonobu Okazaki[1]

[1]*Department of Computer Science and Systems Engineering, Faculty of Engineering, University of Miyazaki, 1-1 Gakuen-kibanadai nishi, Miyazaki, 889-2192 Japan*
[2]*Department of Information Security, Faculty of Information Systems, Siebold Campus, University of Nagasaki, 1-1-1 Manabino, Nagayo-cho, Nishi-Sonogi-gun, Nagasaki, 851-2195 Japan*

ARTICLE INFO

Article History

Received 10 November 2019
Accepted 21 May 2020

Keywords

Natural language specification
VDM++
automatic generation
formal method
formal specification

ABSTRACT

The natural language includes ambiguous expressions. Vienna Development Method (VDM) is one of methodology on the formal methods to write the specification without ambiguity. Because VDM++ is written by strict grammar, it is difficult to write a VDM++ specification. This research attempts to generate a VDM++ specification automatically from a natural language specification by machine learning. To generate a VDM++ specification, it is necessary to extract words that consist of predicate corresponding to the function and nouns corresponding to variable from the natural language specification. This paper proposes an approach to generate a VDM++ specification based on its grammar from the classified word list. Identifiers are generated from the classified word list, and then the VDM++ specification can be generated by converting them into VDM++ grammar.

1. INTRODUCTION

Software bugs make a huge impact on our society [1,2]. Most of them can be caused by the general use of a natural language in the upstream process of software development because the natural language contains an ambiguity. The ambiguity can drive programmers to misinterpret the specification [2]. Hence, the programmer might embed some bugs in the program.

One way of solving this problem is to design software in the upstream process by using a formal method. In development by using it, specifications are written in a formal specification language based on mathematical logic. They can be proved the mathematically by using theorem proving and mechanical checking [3]. It allows for precise design without the ambiguity.

Vienna Development Method (VDM)++ is a formal specification language that can handle object-oriented modeling [4]. VDM++ has strict grammar. Because of the strict grammar, it is difficult to describe them because they are needed to write data types and system invariants that are not available in a natural language specification.

This study attempts to generate a VDM++ specification automatically from a natural language specification by machine learning. In the automatic generation of a VDM++ specification, it is necessary to extract words that consist of predicates and nouns from the natural language specifications. The predicates are corresponding to functions and nouns are corresponding to variables, respectively. However, it is difficult to use the extracted nouns and predicates along with the grammar of VDM++ specification. A VDM++ specification cannot be described from the only extracted nouns and predicates.

This paper proposes an approach to generate VDM++ specification from the extracted nouns and predicates.

*Corresponding author. Email: kat@cs.miyazaki-u.ac.jp

2. VDM++

Vienna development method is one of the formal methods. VDM++ is an object-oriented extension of the VDM-SL language and is currently most used in VDM [4].

Table 1 shows the definitions in VDM++ as a target in this paper. The keywords contain the number of elements and syntax information. The syntax is the set of description rules in VDM++, and each element corresponds to ordinal numbers in the syntax in Table 1 [4].

3. THE PROPOSED APPROACH

The proposed approach automatically generates a VDM++ specification. Figure 1 shows the flow of the proposed approach.

1. Extract a word list from a natural language specification. It is referred to as a word list as the extracted word list.

Table 1 | The definitions in VDM++

VDM++ keyword	Number of elements	Syntax
Values	2	1st = 2nd
Types	2	public 1st = 2nd
Instance variables	3	1st: 2nd: = 3rd
Operations	5 or more	public 1st: 2nd ⇒ 3rd pre 4th post 5th

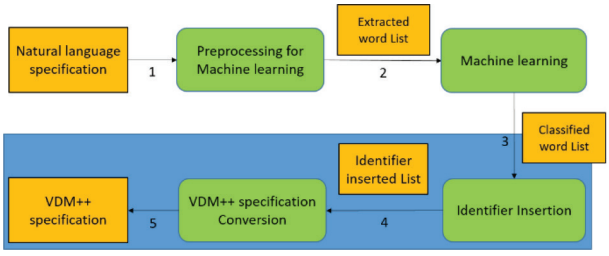

Figure 1 | The flow of the proposed approach.

2. Classify word list by machine learning from the extracted word list. It is referred to as a word list as the classified word list.
3. Insert an identifier into the classified word list.
4. Get the identifier from the word list into which the identifier is inserted.
5. Put the elements of each word list into the VDM++ specification syntax corresponding to the gotten identifier.

This paper focuses on 3–5 steps (lower frame of Figure 1). Here, the generation of a classified word list by machine learning from the natural language specification in 1 and 2 steps is a future issue.

3.1. The Data Structure

The proposed approach defines two data structures: classified word list and identifier.

3.1.1. Classified word list

A classified word list is a per line of the list of elements required to generate an identifier based on a syntax for a variable or function. Figure 2 shows the flow of the identifier insertion process described in Section 3.2.1. It also shows an example of a classified word list at the upper.

Each column of the classified word list is explained as follows. The element in the first column (Column A) is a temporary ID, and it is replaced with an identifier later. The element in the second column (Column B) is the variable name or function name described in the VDM++ specification.

The elements in the third column (Column C) and the subsequent columns differ depending on each target: values, types, instance variables, and operations.

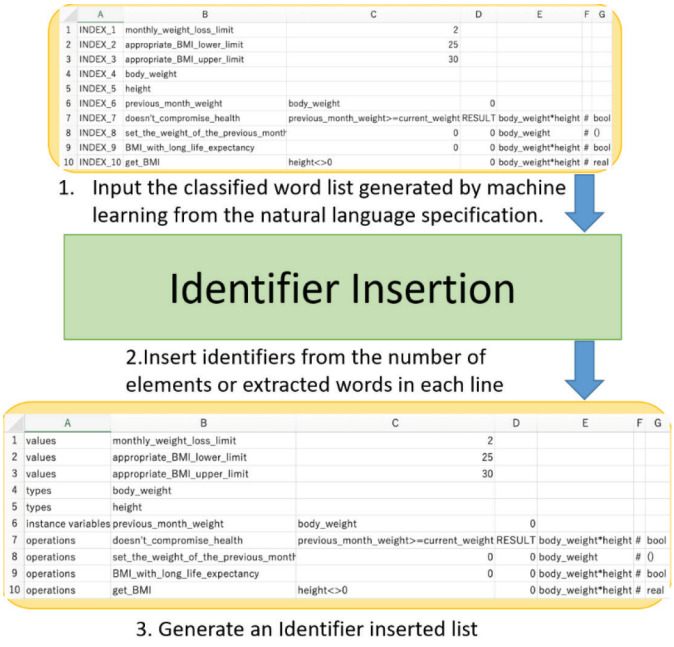

Figure 2 | The flow of the identifier insertion process.

- In the case of the values, the element in Column C is real value.
- In the case of the types, the element in Column C is type's name.
- In the case of the instance variable, the columns after Column C consist of the type name and initial value. In another case, the values may not be specified.
- In the case of the operations, the columns after Column C consist of a pre-condition, post-condition, function argument, delimiter "#", and return type.

3.1.2. Identifiers

Identifiers are generated based on the number of elements in the classified word list corresponding to the keywords of the VDM++ in Table 1. The insertion condition of each identifier in the classified word list is described as follows.

- The "values" keyword indicates that the number of elements in the second and subsequent columns (Column B or later) of the classified word list is 2, and the third column (Column C) is a value.
- The "types" keyword indicates that the number of elements in the second and subsequent columns (Column B or later) of the classified word list is 2, and the third column (Column C) is a type definition. If the number of elements is 1, the element in the third column (Column C) is real.
- The "instance variables" keyword indicates that the number of elements in the second and subsequent columns (Column B or later) of the classified word list is 3.
- The "operations" keyword indicates that the number of elements in the second and subsequent columns (Column B or later) of the classified word list is 5 or greater.

3.2. The Approach Process

To generate VDM++ specification, it is proposed two algorithms: the identifier insertion process and the VDM++ specification conversion process.

3.2.1. Identifier insertion process

Figure 2 shows the flow of the identifier insertion process. The identifier insertion process replaces a temporary ID in the 1st column (Column A) of the classified word list with an identifier following the conditions described in Section 3.1.2.

3.2.2. VDM++ specification conversion process

Figure 3 shows the flow of the VDM++ specification conversion process. The VDM++ specification conversion process describes the VDM++ specification based on the identifier inserted list.

How to describe the VDM++ specification is explained. First, the "class <class-name>" is described on the first line. The word "<class-name>" is described as the class name of the specification. The "<class-name>" is given as an argument when we generate the specification. Next, each VDM++ keyword is described. Statements are written after the keyword according to the syntax. Finally, the "end <class-name>" is described to the end of the line.

As an example, the third row of the list in which the identifier is inserted in Figure 2 is described. The first step reads the identifier of the first column and finds the "values". The second step, according to its syntax "1st = 2nd;", associates "appropriate_BMI_upper_limit" in the second column (Column B) and "30" in the third column (Column C) of the identifier inserted list, respectively. The final step writes the statement "appropriate_BMI_upper_limit = 30;" after keyword "types" that in the VDM++ specifications.

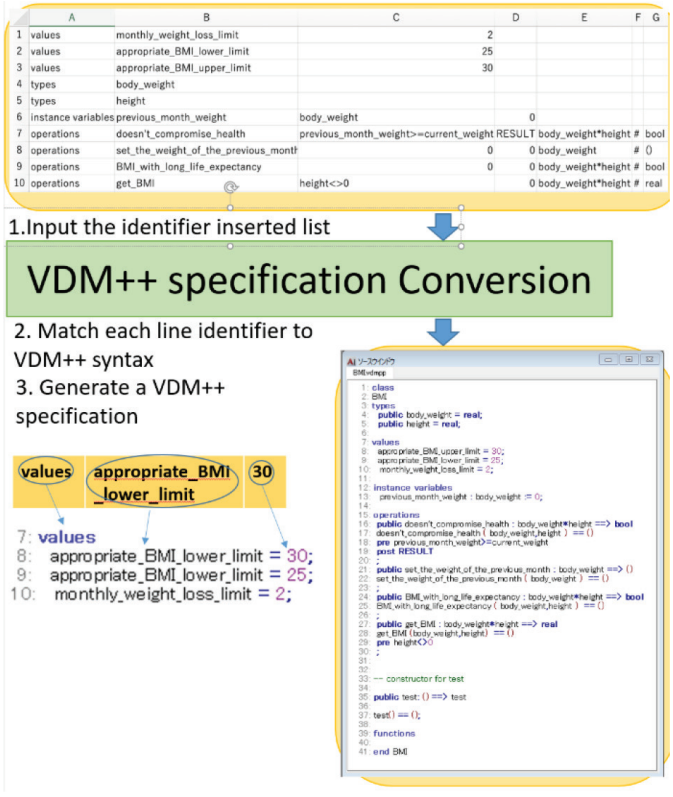

Figure 3 | The flow of VDM++ specification conversion process.

4. APPLICATION EXAMPLES AND DISCUSSION

We have developed two tools: an inserter and a converter, which are implemented in the two proposed process. They have applied a classified word list to the inserter and an identifier inserted list to the converter.

The upper of Figure 2 shows the input of inserter, and the lower of Figure 2 shows the output of the inserter. For each row in the output list, we verify that the identifier can be inserted correctly as follows. Rows 1–3 in the classified word list represents a variable with an initial value. Because the number of elements in the second and subsequent columns (Column B or later) of the input list is 2, and the third column (Column C) is a number, the inserter inserts an identifier "values". Rows 4 and 5 represents a variable that defines only the types. Because the number of elements in the second and subsequent columns (Column B or later) of the input list is 1, the inserter inserts an identifier "types". Row 6 represents the instance variable. Because the number of the elements in the second and subsequent columns (Column B or later) of the input list is 3, the inserter inserts an identifier "instance variables". Rows 7–10 represent operations. Because the number of elements in the second and subsequent columns (Column B or later) of the input list is 5 or greater, the inserter inserts an identifier "operations". Thus, the inserter can generate an identifier inserted list correctly.

The upper of Figure 3 shows the input of a converter and the lower of Figure 3 shows the output of the converter. The output of the inserter in Figure 2 is the same as the input of the converter in Figure 3. We verify that the converter generates a VDM++ specification correctly based on the syntax in Table 1 as well as the inserter. We confirm that the converter can generate the VDM++ specification that consists of statements corresponding to the syntax in Table 1.

Furthermore, we verify that the VDM++ specification generated by the tools based follows the VDM++ syntax. Figure 4 shows an output of the converter to describe the VDM++ specification. VDM++ Toolbox with syntax and type checker displays no warning for the generated VDM++ specification. Therefore, we have confirmed that the proposed process can automatically generate the VDM++ specification corresponding to the VDM++ syntax from the classified word list.

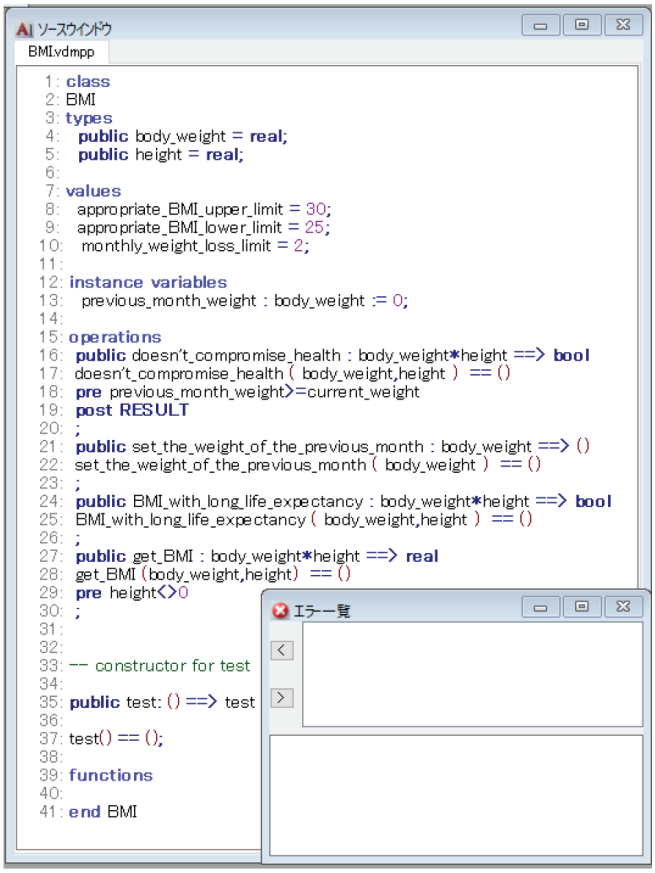

Figure 4 | The VDM++ specification displayed by VDM++ Toolbox.

5. CONCLUSION

This paper has proposed the approach to aim at automatically generating a VDM++ specification from the classified word list. It means this paper proposes 3–5 steps (lower frame of Figure 1). As a constraint, it assumes that a classified word list is generated by the machine learning from a natural language specification by the steps 1 and 2. As a result, the VDM++ specification can be automatically generated from the list, which will be able to classify from the natural language specification by machine learning.

We confirmed that the VDM++ specification is generated by applying the classified word list to the proposed approach. From this result, it can be useful by the automatic generation of VDM++ specifications.

Future issues are as follows:

- Generating a classified word list by machine learning from a natural language specification.
- Corresponding to another syntax in VDM++.

About the first issue, it corresponds to steps 1 and 2 in Section 3. We think that it is possible to generate the classified word list from the natural language specification by implementing the following.

1. Generate a modification relation by extracting the variable names and function names from the natural language specification.
2. Let a model learn the elements to insert the identifiers by using the modification relation [5].
3. Generate the classified word list from the extracted word list by using the learning model.

CONFLICTS OF INTEREST

The authors declare they have no conflicts of interest.

REFERENCES

[1] E. Marcus, H. Stern, Blueprints for high availability, Wiley Publishing, Hoboken, NJ, USA, 2003. Available from: https://dl.acm.org/doi/book/10.5555/2826066.

[2] National Institute of Standards and Technology (NIST), Department of Commerce, Software errors cost U.S. economy $59.5 billion annually, NIST news release 2002-10, 2002. Available from: http://www.abeacha.com/NIST_press_release_bugs_cost.html.

[3] M. Fähndrich, M. Barnett, F. Logozzo, Embedded contract languages, Proceedings of the 2010 ACM Symposium on Applied Computing (SAC), Association for Computing Machinery, 2010, pp. 2103–2110. Available from: https://www.researchgate.net/publication/221002507_Embedded_contract_languages.

[4] International Organization for Standardization, "ISO/IEC 13817-1:1996, Information technology - Programming languages, their environments and system software interfaces - Vienna Development Method - Specification Language - Part 1: Base language", International Organization for Standardization, 1996. Available from: https://www.iso.org/standard/22988.html.

[5] S. Ioffe, C. Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift, Proceedings of the 32nd International Conference on Machine Learning (ICML), 2015, pp. 448–456. Available from: http://proceedings.mlr.press/v37/ioffe15.html.

AUTHORS INTRODUCTION

Tetsuro Katayama

He received the PhD 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 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.

Hisaaki Yamaba

He received the B.S. and M.S. degrees in Chemical Engineering from Tokyo Institute of Technology, Japan, in 1988 and 1990, respectively, and the PhD degree in Systems Engineering from University of Miyazaki, Japan, in 2011. He is currently an Assistant Professor with the Faculty of Engineering, University of Miyazaki, Japan. His research interests include network security and user authentication. He is a member of SICE and SCEJ.

Yasuhiro Shigyo

He received the Bachelor's degree in engineering (Computer Science and Systems Engineering) from University of Miyazaki, Japan in 2019. He is currently a Master's student in Graduate School of Engineering at the University of Miyazaki, Japan. His research interests natural language processing and formal specifications.

Kentaro Aburada

He received the B.S., M.S., and PhD Degrees in Computer Science and System Engineering from the University of Miyazaki, Japan, in 2003, 2005 and 2009, respectively. He is currently an Associate Professor with the Faculty of Engineering, University of Miyazaki, Japan. His research interests include computer network and security. He is a member of IPSJ and IEICE.

Yoshihiro Kita

He received a PhD degree in Systems Engineering from University of Miyazaki, Japan, in 2011. He is currently an Associate Professor with the Faculty of Information Systems, University of Nagasaki, Japan. His research interests include software testing and biometrics authentication.

Naonobu Okazaki

He received his B.S., M.S., and PhD degrees in Electrical and Communication Engineering from Tohoku University, Japan, in 1986, 1988 and 1992, respectively. He joined the Information Technology Research and Development Center, Mitsubishi Electric Corporation in 1991. He is currently a Professor with the Faculty of Engineering, University of Miyazaki since 2002. His research interests include mobile network and network security. He is a member of IPSJ, IEICE and IEEE.