Internationalization of Domain Dictionary Management Tool

Shinta Inoue[1] Yoichi Omori[2] Keijiro Araki[2]

Abstract: Formal methods have an ability to provide a software model with strictly defined semantics, and it enables verification by tools on a computer. Requirements of software, however, involving stakeholders who are not familiar with information engineering must be maintained through the development process not only in formal specifications but in natural languages. We proposed a process to build a formal model out of the natural description and developed a dictionary tool to support it. The tool reduces pains to maintain the map between them and to compose a whole formal model from each formal definitions of word in the vocabulary. It is also effective to an inter-linguistic software development project, since formal methods can define mathematical meanings of a model of the target system, which are common over the countries. We applied the process and the tool to a public specification of an electric pot for training as a case study through the implementation of a simulator, and succeeded to detect some flaws in the specification. We tested the tool and modified developing process on an international environment where requirements are written in various natural languages in this paper.

Keywords: requirements, software tool, VDM, implementation, internationalization

1. Introduction

1.1 Background

Documents in natural language play important role in the early stage of software development. Because system development usually starts with requirements from domain experts, managers, or end users, who are not necessarily familiar with information engineering. Communications among stakeholders including non-engineers are accordingly performed on natural language.

There is, however, a semantic gap between a natural language and a programming language, which sometime causes discrepancies between them and misunderstanding among stakeholders. Advantages of natural languages would adversely lead into a confusion from ambiguous comprehension in the later phases such as design or implementation. The role of software engineers is a kind of interpreter from natural language to programming language through the design process, but the gap is so immense though programming environments have become rich.

In other words,we believe that abstract and integral specifications should be described in natural language, and computational and componential specifications should be described in formal language. Formal model can compensate the semantic gap between software requirement specifications and a program code by mathematical proofs. Thus, we propose a way to realize early verification with our invented tool that focus on the above points to crunch the hobble keeping consistency between two languages.

[1] Graduate School of Information Science and Electrical Engineering, Kyushu University
[2] Graduate Faculty of Information Science and Electrical Engineering, Kyushu University

1.2 Related Work

Limits of software specifications in natural language were pointed out, and the way of formal descriptions was investigated[1]. Thus, whether automatic or by hand, extraction of a formal specification from requirements in natural language has been studied for long time.

Most of those assume to adopt one-way process of refinement from natural language to machine program through formal specification[2]. But, the role of natural language is revised to capture ambiguous requirement especially in non-functional features[3].

There are two major ideas in this field. First one is algorithmic conversion from a specification in natural language into a formal model. Most thrived method in this type is the software cost reduction method(SCR) that is famous for the flight program in the U.S. Navy[4]. SCR is a systematic toolset to store requirements described by a restricted natural language into a table and automatically refine them into a code in a specific domain [5].

Another idea is knowledge based interpretation of natural language. In [6], extracting the data structure of the formal specification descriptive language VDM from natural language description is studied.

Formalization of ether conversion process or intermediate products become more important to encompass widespread adoption[7][8][9].

The rest of this paper is organized as follows. In Section 2, we give introduce a domain dictionary management tool. In Section 3, we describe an evaluation method of the tool, the evaluation result and the problem of the tool. In Section 4, we account for internationalization tool by multilingual support. Finally, Section 5 concludes this paper.

2. Domain Dictionary Management Tool

The domain dictionary management tool that can support the conversion from a natural language description into a formal specification using VDM++ was developed[10].

The tool is a plugin of eclipse which is an Integrated Development Environment (IDE) developed[11]. It is to be noted that our contribution is not an automatic conversion, but to help modelers' work and communication among stakeholders.

2.1 Formal Method

Formal method is a method of verifying modeled specification mathematically. Since a mathematical verification is possible to abstract model, not only a product but intermediate product can be verified. Formal methods are mathematically-based techniques or tools useful in developing system.They can give some assurances based on its background logic and chance to revise specifications of requirements at the early phase, because computer software should have a nature as a finite state automaton, and its all properties can be verified mathematically.

Formal languages have a defined semantics based on mathematics to describe specifications and afford the background logic. We adopt VDM++ as the formal modeling language in our method[12]. VDM++ is a formal language based on set theory and first order predicate logic. It has quite simple syntax to be understood and affinity with object-oriented programming languages. Our tool treats an entry in dictionaries as a class, an instance, an operation, or a function in object-oriented way.

2.2 Dictionary Management

The functions of the domain dictionary management tool are following.

- Dictionary management
- Coloring keywords
- Output formal model

The GUI appearance of the tool is shown in **Fig. 1**. Dictionaries of the tool store keywords in specifications as entries. The dictionary files are managed through the dictionary view of the tool, and they can be edited independently by other editors because the files are in a XML format.

2.2.1 Registration and edit of Entry

A selected keyword is added to the specified dictionary as an entry. Registration is performed on an arbitrary region as a phrase that is specified by dragging mouse in the documents. The entry is not limited to a lexical word but a phrase and is treated as a keyword.

The following fields are set to the entry registered in dictionary.

- Part of speech
- Informal definition
- Metaclass
- Formal definition

"Part of speech" is a column to set attribute of the entry. "Informal definition" is a column to describe explanation of the entry by natural language.

"Metaclass" is a column to select the item corresponding to the block of VDM++.

"Formal definition" is the column to enter VDM++ description. Modeling of specifications is performed by writing in VDM description directly in the dictionary.

2.3 Coloring entry

The tool searches and highlights the whole entry words of the dictionary in the specification to assist coverage check by engineers. The tool improves visibility of the keyword in specification by coloring.

As a result, the visibility of keywords in the documents of natural language would be much improved for the modelers. The complete coverage rate of the entry registered in the dictionary can also be checked.

2.4 Model output

The tool support composition from the formal definition of each entry in the dictionary into a model composed of descriptions in VDM++. When the model output from the dictionary form, each keyword whose type is "class" in the model, will create an output file whose name is the same for the class. The domain dictionary management tool outputs VDM++ description file to specified file by pushing "Model output button".

In addition, if type is not "class" ("variable", "constant", "function", and "operation""), they are included into the previously declared class definition and output as members of the class. Therefore, the tool can alternate two aspects for a dictionary at anytime. Those are the registration form and the generation form, and the former is in order to the words appear in the requirement documents and the latter is in order to represent structures of domain.

Type of "variable", "constant", "function", and "operation" are written into "instance variables", "values", "functions", and "operations" section of the VDM++, respectively.

The check of syntax is left to other tools such as VDM-Tools[13] or Overture[14]. Program code generation from VDM++ is also delegated to these external tools.

3. Evaluation of Tool

In order to evaluate the domain dictionary management tool, we carried out a case study through the software development.

- Modeling of a specification
- Verification of the model
- Implementation of the model

We created the VDM++ model from a specification of an embedded system by using the tool. Then, we verified the VDM++ model using the state transition table, and finally implemented a simulator of electronic pot based on the VDM++ model.

3.1 Modeling

We developed a VDM++ model from a specification of an electric pot from SESSAME[15] for training material. We applied the following procedure[10].

(1) Registration of the keywords in specification to the tool's dictionary
(2) Regulation of the dictionary's entries which is "Noun phrase"
(3) Unification of the entries with same meaning

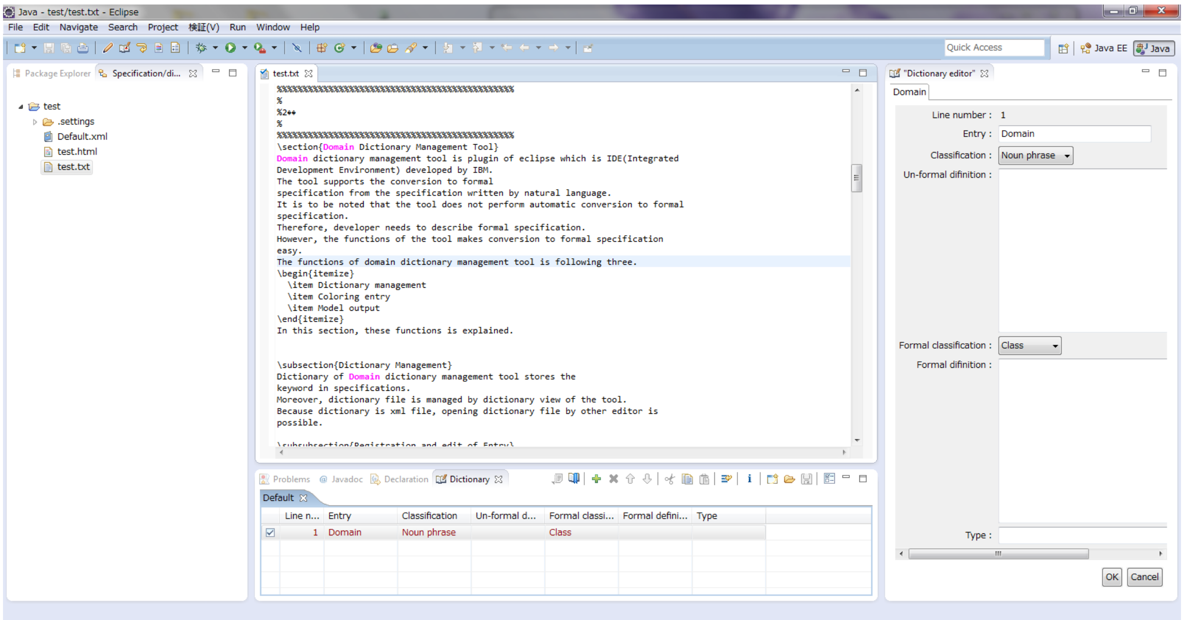

Fig. 1 Domain dictionary management tool

(4) Determination of the meaning of polysemy
(5) Division of class
(6) Regulation of the dictionary's entries which is "Verb phrase" and "State"
(7) Writing of the VDM++ descriptions to the dictionary
(8) Output of the VDM++ model
(9) Refinement of the VDM++ model

We registered 69 keywords to the dictionary. The VDM++ model was reviewed four times with VDM++ specialists, because the abstraction level of the first model was too low.

We modify the above procedure in Sec.4.

3.2 Verification

We verified the VDM++ model created by using the tool. Verification was performed using a state transition table. Using the state transition table is due to the VDM++ expresses behavior of a system by state transition of class. This table was described based on following three variables showing a state.

- State of the pot
- State of the hot-water supply
- State of error

A system state is expressed in the combination of these variables. We verified two points, as described below. One of them is whether the states change as the intention of specification. Two of them is that transition conditions are described correctly. As a result of these verification, it was possible that we had mistaken description of the conditions of a temperature error.

3.3 Implementation

We implemented a simulator based on the verified VDM++ model. Implementation consists of two steps, design and coding. First, we designed a software using VDM++ continuously. In the design, the addition of a class required for a program and a decision of the algorithm were performed. Second, we coded the simulator program using Java. Users can operate a simulator by pushing the button on a screen and check the behavior of the pot. The simulator of electronic pot is shown in **Fig. 2**.

Fig. 2 Simulator of electronic pot

4. Internationalization of Tool

The tool had supported only Japanese. However, software development is recently globalized. For example, offshore developments with the overseas firms are commonly performed[16]. Definition of a word in a natural language easily become ambiguous because it drags every day meaning. Formal methods are based on universal mathematics so that can relieve the misunderstanding from different cultures or languages. Therefore, we think it is important that the tool can support many languages.

4.1 Tool and process extension

We extend the existing tool in following points;

- Utilization of informal definition

  A dictionary holds informal definition field for each entry. We can use the field for translation from original requirement documents to other language. If readers who are sup-

posed to be developers or programmers are not familiar with the original language of requirements, they can intuitively understand by reading this field.

Exact and precise meaning of each entry is still held in the formal definition field, therefore misunderstandings which are arose from translation of natural languages can be avoid.

- dictionary header holds language information of both input and output

  New version of this tool allows comma-separated multi-languages in meta information of the dictionary, which is used in case of the original requirement documents are written in multi-languages.

There are two major stories of a multi-linguistic development. One is vertical international specialization and another is requirements itself are written in multi-languages, which are correspond to the above items respectively.

In the former case, the customers take the initiative from step (1) through (4), thus the selection of keywords for entries and arrangement of them are responsible to them. And on the contrary, developers lead from step (5) through (9), therefore consistency of the target model and refinements to implementation depend on their decision. The informal definition field had been expected as an explanation, but we can use it for side by side translations in this case. Customers and developers can respectively use their own natural language to understand the target, but they have to share the same formal definition of important concepts in the domain.

In the latter case, the tool can utilize external tools to check syntax or style of descriptions based on the meta information in the extended dictionary. The roles of fields in each entry is the same as the preceding case study, because it it not necessary to distinct which language is used to represent keywords in the requirements.

The validity of definition should be checked by few bilingual specialist but not all stakeholders. Stakeholders can also confirm the validity of the model by animation on VDMTools which we used in the test case. Model animation can show probability of the model without examination definitions.

4.2 Translation of the tool into English

First, we translate tool menu into English. Since the menu of the tool was Japanese, we translated the menu into English. The tool was coded by Java, we rewrite literal Japanese to English.

Next, we ran the tool with English documents on English environment. We drew up the dictionary using the tool from the English document as evaluation of the tool. The carried-out evaluation is following two.

- Registration to the dictionary of English words
- Check of a coloring function

The tool could register the entries in the dictionary satisfactorily, and it worked well on the English specification by using the function of coloring keywords. Model output also worked without any trouble. By taking care about a setup of the encoding of documents, it was found that this function operates satisfactorily. Moreover, we can conjecture that the tool can support other and multiple languages by setting up the encoding code appropriately,

.

5. Conclusions

In this study, we developed and evaluated the domain dictionary management tool.

We carried out the following two. First, we showed that the tool was useful to support of the conversion to formal specification from natural language. It was helpful in the check of the correspondence relation between specification and the model to use the dictionary. It was found that domain dictionary management tool is effective when we detailed VDM++ model stepwise. In addition, it was also found that the management of specification described by natural language becomes easier than not using the dictionary. By our approach, the specifications of natural language are also manageable.

Second, we translated and checked the tool on English and other languages for internationalization. It was found that the tool can be used in languages other than Japanese.

Acknowledgments Thanks to SESSAME for the publication of example specifications about an electric pot GOMA-1015. Construction of the tool is supported by Mr. Yasuharu Yoshimura and others in Kyushu Business Corporation. Professor Han-Myung Chang reviewed and give useful advice to our VDM++ model. This work was partially supported by MEXT Grant-in-Aid for Scientific Research(S) HBG4220001.

References

[1] Meyer, B.: On Formalism in Specifications, *IEEE Software*, Vol. 2, No. 1, pp. 6–26 (1985).

[2] Saeki, M., Horai, H. and Enomoto, H.: Software development process from natural language specification, *Proceedings of the 11th international conference on Software engineering*, ACM, pp. 64–73 (online), DOI: http://doi.acm.org/10.1145/74587.74594 (1989).

[3] Ryan, K.: The role of natural language in requirements engineering, *Proceedings of IEEE International Symposium on Requirements Engineering*, pp. 240–242 (1993).

[4] Heitmeyer, C. L.: Formal Methods for Specifying Validating, and Verifying Requirements, *Journal of Universal Computer Science*, Vol. 13, No. 5, pp. 607–618 (2007).

[5] Heninger, K. L.: Specifying Software Requirements for Complex Systems: New Techniques and Their Application, *IEEE Transaction on Software Engineering*, Vol. SE-6, No. 1, pp. 2–13 (1980).

[6] Vadera, S. and Meziane, F.: From English to Formal Specifications, *Journal of Systems and Software*, Vol. 37, No. 9, pp. 753–763 (1994).

[7] Kemmerer, R. A.: Integrating formal methods into the development process, *IEEE Software*, Vol. 7, No. 5, pp. 37–50 (1990).

[8] Moreno, A. M.: Object-Oriented Analysis from Textual Specifications, *Proceedings of 9 th International Conference on Software Engineering and Knowledge Engineering* (1997).

[9] Lee, B.-S. and Bryant, B. R.: Automated Conversion from Requirements Documentation to an Object-Oriented Formal Specification Language, *Proceedings of the 2002 ACM Symposium on Applied Computing*, pp. 932–936 (2002).

[10] Omori, Y. and Araki, K.: Tool Support for Domain Analysis of the Software Specification in Natural Language, *Proceedings of the IEEE TENCON 2010*, pp. T7–3.3(CD–ROM) (2010).

[11] The Eclipse foundation: Eclipse IDE, http://www.eclipse.org/.

[12] Larsen, P. G., Mukherjee, P., Plat, N., Verhoef, M. and Fitzgerald, J.: *Validated Designs For Object-oriented Systems*, Springer (2005).

[13] SCSK systems: VDMTools Information, http://vdmtools.jp/.

[14] Overture comitee: The Overture tool, http://wiki.overturetool.org/.

[15] SESSAME: Training material electric pot GOMA-1015 (In Japanese), http://www.sessame.jp/workinggroup/WorkingGroup2/POT_Specification.htm.

[16] Aspray, W. and adn Moshe Y. Vardi, F. M.(eds.): *Globalization and Offshoring of Software A Report of the ACM Job Migration Task Force*, ACM Press (2006).