How to cite this paper:

Swee Yin Wong, Edwin Mit, & Jonathan Sidi. (2017). Validation of UC2VDM++ architecture in Zulikha, J. & N. H. Zakaria (Eds.), Proceedings of the 6th International Conference on Computing & Informatics (pp 663-668). Sintok: School of Computing.

VALIDATION OF UC2VDM++ ARCHITECTURE

Swee Yin Wong[1], Edwin Mit[2], and Jonathan Sidi[3]

[1]*Universiti Malaysia Sarawak, Malaysia,* cyinwsy@gmail.com
[2]*Universiti Malaysia Sarawak, Malaysia,* edwin@fit.unimas.my
[3] *Universiti Malaysia Sarawak, Malaysia,* jonathan@fit.unimas.my

ABSTRACT. Bridging the gaps between use case and formal specification can be very helpful in obtaining a reliable and rigor software model through economic and easy object modeling. However, bridging the syntax and semantic formalism gaps between natural language use case scenario and VDM++ formal specification is a huge challenge. This is because natural language requirement has been well recognized that it is inherently ambiguous, incomplete and inconsistent. Its ambiguity will result in an incorrect and inaccurate analysis and design model. In order to obtain economic and rigor architecture of software model, this paper would introduce and discuss the motivations and challenges in the implementation of mapping rules and restriction rules embedded in the Uc2VDM++, and discuss the validation process of Uc2VDM++ in producing a correct software model.

Keywords: use case, formal model, validation, architecture

INTRODUCTION

Use case diagram of Unified Modeling Language (UML) is an essential for understanding how to bridge the gap between user and software developer and a high level model for the software systems (Sengupta & Bhattacharya, 2006). Yet, most use case specifications are described in natural language (i.e. English) which is inherently ambiguous, incomplete, and imprecise (Ghosh, Elenius, Li & Lincoln, 2016). Thus, it will cause some misunderstanding among software developers, domain experts and end users of a system due to its ambiguity and flexibility. As a consequence, it will result in incorrect and inaccurate analysis and design model.

Use case is used in the early stage of software development; formalism of use cases can reduce a lot of confusion and misinterpretation among end users and software developers and hence can be very favorable to the software quality (Shen & Liu, 2003). Formal method is the most favorable technique in reducing errors particularly at the earlier stages of software development (Bakri, Harun, Alzoubi, & Ibrahim, 2013). In addition, formal method has been quite successfully used in requirements representation for discovery inconsistencies, incompleteness and ambiguities (Sharma & Biswas, 2015; Mondal, Das, & Banerjee, 2014).

This paper presents our continuous research in Formal-Object Tool (FOTool), (Mit, Ng, & Cheah, 2014). This paper will discuss the validation process of Uc2VDM++ in producing a correct software model, the motivations and challenges in the implementing the Uc2VDM++. This paper also discusses the development of the prototype which is embedded with the proposed formal use case template and its restriction and mapping rules defined in previous work (Wong et al., 2016). Through this use case formalization, it is believed that a precise, con-

sistent, complete, and unambiguous software specification can be produced. Hence, it can indirectly ameliorate the software reliability, quality, understandability and design time, at the same time prevent software failure and lower redesign costs.

The following sections of this paper presents on the review of related works, Uc2VDM++ environment , the implementation and validation of the Uc2VDM++ prototype on a small case study, and the motivations and challenges faced in the implementation of mapping and restriction rules. The last section is to conclude this paper.

RELATED WORKS

A number of research works have been done in the direction of formalization of UML diagrams into different formal models such as Petri Net, CASL, Z notation, and so on. Nevertheless, only few researches formalize use cases into VDM++ formal specification.

Sengupta and Bhattacharya (2008) suggested a extendable and structured format for use case and formalized them into Z notation schema. The Z notation schema was validated by using a Z notation type checker, ZTC. Yet, Z cannot be executed as it has no compiler (Jia, 2002). Due to this limitation, automated verification is not probable only if Z is mapped into other executable models (Chanda, Kanjilal, Sengupta, & Bhattacharya, 2009). Chanda et al. (2009) proposed a context free grammar for defining UML class, use case, and activity diagrams formally in order to develop a complier and formally verify the design of object-oriented systems (Chanda et al., 2009).

A systematic approach to translate functional requirements describing in use case diagram into Maude formal specification was suggested by Mokhati and Badri (2009) in order to bridge the gaps between informal and formal specification. Maude is a programming language and executable formal specification using rewriting logic (Clavel et al., 2002). The formal framework for analyzing and verifying functional requirements of system through Maude descriptions by using Maude model checker had been proposed (Mokhati & Badri, 2009).

Zhao and Duan (2009) proposed to derive Timed and Controlled Petri Nets (TCPN) from use case description. The authors adopted Petri Nets because of its mathematical simplicity, existence analysis tool, well developed quantitative and qualitative analysis techniques, etc, which will help on the validation of the model.

Furthermore, in a continuous research (Lee, Bordbar, & Bajwa, 2009; Bajwa, Bordbar, & Lee, 2010; Bajwa, Lee, & Bordbar, 2011, 2012a), the authors proposed the automated transformation of UML model describing in English representation into Object Constraint Language (OCL). But, OCL is the least used member in UML family of languages, mainly because of its uncommon semantic and syntax (Bajwa et al., 2010). The authors transformed natural language into OCL statement by adopting Semantic Business Vocabulary and Rules (SBVR) as intermediary measure for dealing with the ambiguities in semantic and inconsistencies in syntax of English expression. (Bajwa et al., 2012b).

In another continuous research, Yue, Briand, and Labiche (2009, 2010, 2013, 2015) derived UML analysis models from use cases by recommending Restricted Use Case Modeling (RUCM) for reducing incompleteness and imprecision of use case specification. The proposed RUCM is made up of one set of restriction rules and one use case template for analysis facilitaion in automation and ambiguities reduction (Yue et al., 2010). The authors applied Stanford parser in their developed tool, called aToucan (Yue et al., 2015). However, the parser has some limitations and is unable to generate accurate results frequently (Yue et al., 2010, 2013, 2015).

Besides that, Mondal et al. (2014) suggested to formalize UML use case diagram into an expressive specification language that is Common Algebraic Specification Language (CASL). The language was designed to substitute existing algebraic specification languages (Astesiano, 2002). The authors highlighted that CASL approach was used by them as both Z and B notation are unable to give algebraic prototyping and architectural specification. A case study was also carried out in their work to show how use case model was defined formally into CASL according to their proposed mapping. Their on going research is to validate the case study using CASE tool (Mondal et al., 2014).

In addition, Mit, Ng, and Cheah (2014) carried out a study focusing on the translation of UML activity and class diagrams into VDM++ formal specification. This was encouraged by the incompleteness of mapping between VDM++ formal specification and activity diagram due to their semantic differences. The authors defined a set of mapping rules for integrating the object models and the formal specification according to their semantics similarity. Formal-Object tool (FOTool) was developed for capturing and transforming UML software model specifications into VDM++ formal specifications. The VDM++ formal specifications can be later verified formally by VDM++ ToolBox (Mit et al., 2014).

Numberous researches have been carried out to transform UML into formal model in order to validate the UML models. However, there are a number of limitations in order to produce accurate result. This work adopts some validation process from these literature studies by developing a prototype to prove the Uc2VDM++ archiecture and validates the output generated by the prototype using VDM++ ToolBox.

UC2VDM++ IMPLEMENTATION ENVIRONMENT

The prototype is developed by using JAVA language. Eclipse is selected and used as the platform for the development of the prototype due to its good error checking feature. JDK version 1.8.0_66, the latest version of JDK so far, is used for compiling, debugging and running the program in Eclipse. The latest Stanford parser version 3.6.0 is deployed to assign POS tags and to extract the required information from the proposed textual use case template. XML file is used to temporarily store the UML use case details before transforming them into VDM++ formal specification (Larsen & Fitzgerald, 2007; Overture, n.d.). JDOM version 2.0.5 is also adopted to generate XML structure between these two models. The use case template is designed as the interface of the prototype in form-fill style, used to capture all the required UML use case details. Meanwhile, a tool tip is created by using Balloon tip version 1.2.4.1 for each field of the use case template interface in order to provide users instructions to fill in the use case template.

Process Flow of the Prototype

Figure 1 shows the simple process flow of the prototype. The input of the prototype is a use case in form of template. Since natural language is too flexible and existing natural language parsers have the limitation of assigning 100% correct POS tags for each input, so the prototype involved a very strict validation process for checking each input against the restriction rules. For instance, each simple sentence must follow either Subject-Verb-Direct Object (SVO) or Subject-Verb-Direct Object-Indirect Object (SVOO) grammar form (e.g.: SVO - “Member places book.”). We are unable to describe all scenarios of the restriction rules here due to the limitation of the space. The valid grammar input is then parsed to Stanford parser. The parser is deployed to perform the POS tagging and assign the suitable POS tags for the inputs. The tagged inputs are then extracted and temporary stored as Extensible Markup Language (XML) structure in an XML file. The XML file is generated automatically if the file does not exist. Otherwise, the XML file will be overridden each time for a new use case. Once the XML file is ready, the prototype will directly continue to generate a VDM++

formal specification and store with file extension ".vpp" (e.g.: VDM Output.vpp) based on the proposed set of mapping rules. The output of the prototype is a VDM++ formal specification.

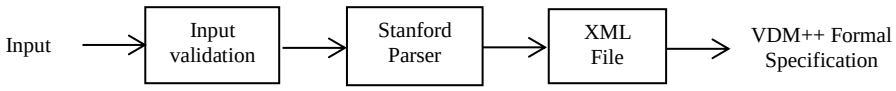

Figure 1. The Simple Process Flow of the Prototype

VALIDATION OF TEMPLATE AND RULES

The proposed use case template had been refined, which consists of Use Case Name, Primary Actor(s), Secondary Actor(s), Generalization(s), Dependency(s), Main Success Scenario, Extension (Alternative), Exception (Error Handling) and Pre/Postcondition(s). Each field of the use case template of the prototype has been tested with both valid and invalid data, complete and also incomplete data. All the features of the prototype are working properly. A VDM++ formal specification is also successfully generated by the prototype and validated by the latest version of VDM++ support tool (i.e.: VDM++ ToolBox version 9.0.6) for syntax and type checking. There are no syntax and type checking errors found in the generated output. Hence, the prototype can be said that it is able to generate a correct VDM++ formal specification.

Case Study

A small case study on library management system was carried out to depict the process as shown in Figure 2.

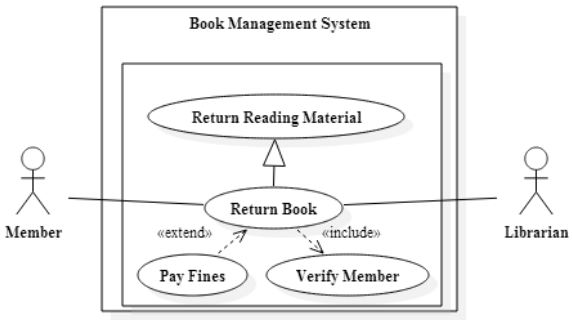

Figure 2. The Partial Use Case Diagram of the Book Management Sub System

A use case "return book" is chosen to be further described as an example here since it involves all the relationships such as association, generalization, extend and include relationship. The use case is mainly described on how a member returns book. For instance, the noun of the use case name "return book" and the generalization use case "return reading material" are defined in VDM++ as "*class* Book *is subclass of* ReadingMaterial". While the whole use case name "return book" is defined as the main operation in VDM++ as "*public* ReturnBook() ==". Furthermore, steps of main success scenario such as "Member places book." is defined as a sub operation "placesBook();" within the block of main operation in VDM++. The generated VDM++ formal specification for that use case "return book" has been imported into VDM++ ToolBox for validation. The output generated by the prototype is correct since both syntax and type checking are correct.

MOTIVATION AND CHALLENGES OF UC2VDM++ RULES

Uc2VDM++ architecture (Wong et al., 2016) is developed to reduce the formalism gaps by formalizing use case into VDM++ formal specification. Restriction rules are created to guide users to fill in correct inputs on the use case template in order to produce a correct output. In addition, mapping rules are produced to reduce the gaps by mapping use case into VDM++ formal specification based on semantics similarity.

However, there are some challenges faced in deriving the rules, such as the flexibility of natural language which can be written in various grammar forms. In order to deal with this problem and ensure correct mapping, only simple SVO or SVOO grammar form (e.g.: SVO - "Member places book.") is allowed. At the same time, the limitation of existing natural language parser has made the prototype to involve a very strict validation process based on the restriction rules. Wrong input or incorrect grammar form will alert an error pop up message. Although the restriction rules can contribute to correct output, the rules may not allow users to have their own way to write natural language use case scenario. Another challenge is the absence of some use case properties that can be mapped to the VDM++ formal specification. Therefore, augmentation and manipulation of the model properties have been carried out between use case and VDM++ formal specification.

CONCLUSION

In this work, the validation rules (i.e., the restriction and mapping rules) have been embedded in a prototype to prove the correctness of Uc2VDM++ architecture. The prototype consists of the proposed use case template, restriction and mapping rules. The prototype is also applied to a small case study and its output has been validated by using VDM++ ToolBox. The result shows that use case can be formalized into VDM++ formal specification successfully. By having valid VDM++ formal specification, it can contribute to produce a precise, complete, unambiguous and consistent software specification and lead to a correct software model.

ACKNOWLEDGEMENTS

The authors would like to thank Universiti Malaysia Sarawak for providing the funding to publish and present this paper. This work is supported by Fundamental Research Grant Scheme: FRGS/02/2014/ICT07/UNIMAS/02/1.

REFERENCES

Astesiano, E., Bidoit, M., Kirchner, H., Krieg-Bruckner, B., Mosses, P. D., Sannella, D., & Tarlecki, A. (2002). CASL: The Common Algebraic Specification Language. *Theoretical Computer Science*, *286*(2), 153-196. doi: 10.1016/S0304-3975(01)00368-1.

Bajwa, I. S., Bordbar, B., & Lee, M. G. (2010). OCL Constraints Generation from Natural Language Specification. *14th IEEE International Enterprise Distributed Object Computing Conference (EDOC)* (pp. 204-213). doi: 10.1109/EDOC.2010.33.

Bajwa, I. S., Lee, M. G., & Bordbar, B. (2011). SBVR Business Rules Generation from Natural Language Specification. *AAAI Spring Symposium: AI for Business Agility, (*pp. 2-8).

Bajwa, I. S., Lee, M. G., & Bordbar, B. (2012a). Resolving Syntactic Ambiguities in Natural Language Specification of Constraint. In A. Gelbukh (Ed.), *Computational Linguistics and Intelligent Text Processing* (pp. 178-187). New York: Springer-Verlag Berlin Heidelber.

Bajwa, I. S., Lee, G., Bordbar, B. (2012b). Translating Natural Language Constraints to OCL. *Journal of King Saud University – Computer Information and Sciences, 24*(2), 117-128. doi:10.1016/j.jksuci.2011.12.003.

Bakri, S. H., Harun, H., Alzoubi, A., & Ibrahim, R. (2013). The Formal Specification For The Inventory System Using Z Language. In Z. Jamaludin, N. ChePa, & M. S. A. Bakar (Eds.), *Proceedings of the 4th International Conference on Computing and Informatics*, *ICOCI* (pp. 419-425). Sintok: Universiti Utara Malaysia.

Chanda, J., Kanjilal, A., Sengupta, S., & Bhattacharya, S. (2009). Tracibility of Requirements and Consistency Verification of UML UseCase, Activity and Class Diagram: A Formal Approcah. *Proceeding of International Conference on Methods and Models in Computer Science* (pp. 1-4). doi: 10.1109/ICM2CS.2009.5397941.

Clavel, M., Duran, F., Eker, S., Lincoln, P., Marti-Oliet, N., Meseguer, J., & Quesada, J. F. (2002). Maude: specification and programming in rewritting logic. *Theoretical Computer Science, 285*(2), 187-243.

Ghosh, S., Elenius, D., Li, W., & Lincoln, P. (2016). ARSENAL:Automatic Requirements Specifiction Extraction from Natural Language. In S. Rayadurgam & O. Tkachuk (Eds.), *NASA Formal Methods Symposium* (pp. 41-46). Switzerland: Springer International Publishing Switzerland.

Jia, X. (2002). *ZTC: A Type Checker for Z Notation User's Guide*. Retrieve from http://Isi.ugr.es

Larsen, P. G., & Fitzgerald, J. (2007). Recent Industrial Applications of VDM in Japan. In P. Boca, J. Bowen, & P. G. Larsen (Eds.), *FACS 2007 Christmas Workshop: Formal Method* (pp. 1-6). London, UK: The British Computer Society.

Lee, M. G., Bordbar, B., & Bajwa, I. S. (2009). *NL2OCL Project*. Retrieve from http://www.cs.bham.ac.uk/~bxb/NL2OCLviaSBVR/NL2OCLviaSBVR.html

Mit, E., Ng, B. D., & Cheah, W. S. (2014). FOTool: Modelling Indigenous Community Cultures In Sarawak. *Journal of Software Engineering and Application, 7*(8), 720-729. doi: 10.4236/jsea.2014.78067.

Mokhati, F., & Badri, M. (2009). Generating Maude Specifications From UML Use Case Diagrams. *Journal of Object Technology, 8*(2), 119-136.

Mondal, B., Das, B., & Banerjee, P. (2014). Formal Specification of UML Use Case Diagram – A CASL Based Approach. *International Journal of Computer Science and Information Technologies, 5*(9), 2113-2717.

Overture. (n.d.). *The Vienna Development Method*. Retrieved Jan 25, 2016 from *http://overturetool.org/*

Sengupta, S., & Bhattacharya, S. (2006). Formalization of UML Use Case Diagram – A Z Notation Based Approach. *'06International Conference on Computing & Informatics* (pp. 1-6). doi: 10.1109/ICOCI.2006.5276507.

Sengupta, S., & Bhattacharya, S. (2008). Formalization of Functional Requirement in Software Development Process. *Foundations of Computing and Decision Science, 33*(1), 83-115.

Sharma, R., & Biswas, K. K. (2015). Generating Logical Representations for Natural Language Requirements Using Syntactic Dependencies and Norm Analysis Patterns. In C. Biemann, S. Handschuh, A. Freitas, F. Meziane, & E. Metais (Eds.), *Natural Language processing and Infromation Systems: 20th International Conference on Applications of Natural Language to Information Systems* (pp. 432-436). Switzerland: Springer International Publishing.

Shen, W., & Liu, S. (2003). Formalization, Testing and Execution of a Use Case Diagram. ln J. S. Dong & J. Woodcock (Eds.), *Formal Methods and Software Engineering* (pp. 68-85). New York: Springer-Verlag Berlin Heidelberg.

Wong, S. Y., Mit, E., & Sidi, J. (2016, August). *Integration of Use case Formal Template Using Mapping Rules*. Paper presented at the CAMP'16 3rd International Conference on Information Retrieval and Management, Melaka, Malaysia.

Zhao, J., & Duan, Z. (2009). Verification of Use Case with Petri Nets in Requirement Analysis. In O. Gervasi, D. Taniar, B. Murgante, A. Lagana, Y. Mun, & M. L. Gavrilova (Eds.), *Computational Science and its Application* (pp. 29-42). New York: Springer-Verlag Berlin Heidelber.

Yue, T., Briand, L. C., & Labiche, Y. (2009). A Use Case Modeling Approach to Facilitate the Transition Towards Analysis Models: Concepts and Empirical Evaluation. In A. Schurr & B. Selic (Eds.), *Model Driven Engineering Languages and Systems: 12th International Conference, MODELS* (pp. 484-498). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-642-04425-0_37.

Yue, T., Briand, L. C., & Labiche, Y. (2010). An Automated Approach toTransform Use Cases into Activity Diagrams. In T. Kuhne, B. Selic, M. Gervais, & F. Terrier (Eds.), *Modelling Foundations and Applications: 6th European Conference, ECMFA* (pp. 337-353). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-642-13595-8_26.

Yue, T., Briand, L. C., & Labiche, Y. (2013). Facilitating the Transition from Use Case Models to Analysis Models: Approach and Experiments. *ACM Transactions on Software Engineering and Methodology (TOSEM), 22*(1), 5:1-38. doi: 10.1145/2430536.2430539.

Yue, T., Briand, L. C., & Labiche, Y. (2015). aToucan: An Automated Framework to Derive UML Analysis Models from Use Case Models. *ACM Transactions on Software Engineering and Methodology*, *24*(3), 13:1-52. doi: 10.1145/2699697.