UML Versus LePUS3 for Modelling Structure of Security Patterns

Abdullah A H Alzahrani, Amnon Eden

Unified Modelling Language (UML) is a widely used and accepted software modelling language. However, UML still suffers from many issues such as formality, lack of precision and low support for abstraction. This paper highlights some of the aforementioned issues with comparison to Language for Pattern Uniform Specification (LePUS3).

Introduction

During the process of software development many people are involved such as software architects, security engineers, designers, implementers, maintainers and others. Each of those might not have enough knowledge about the others' views on points which they all share and work with. For example, the implementer and/or designer might understand that the security engineer needs to secure the communication with the client by using password authentication pattern. However, the implementer and/or designer might have insufficient knowledge or have a different understanding about password authentication pattern. This might introduce unnecessary confusion which costs time, effort and money. Therefore, it is essential to deliver the knowledge of the security engineer clearly to the implementer and/or designer.

This would promote many aspects in all the software development stages. In the case of security it will enhance the overall security of the system as it assures that all people in the development process share the same understanding about the specific security matter. In addition, it will allow validation and/or verification team to check the implementation against clear points. Also, it will improve the process of maintaining the software as it will be easier to locate the aspects that the security engineer delivered to the designer, implementer, verification team and finally the maintainer [1].

In order to convey expertise, the patterns were introduced. For software security, security patterns are used to deliver expertise. Security design patterns are solutions to solve reoccurring security problems in specific domain [2, 3]. In principle, security patterns are design patterns but they are for specific domain which is security domain. Security patterns have gained a growing number of interests. As a result, many researchers have introduced much research to describe, classify, detect, and verify security patterns.

In the case of security patterns descriptions, many have proposed their approaches to improve the current descriptions of security patterns [2, 3, 1, 4, 5, 6]. These approaches usually use UML class diagrams and sequence diagrams to demonstrate the structure and the behaviour, respectively, of a pattern. The approaches, also, combine this with the use of informal description (expressed in English language). However, in these approaches, a number of issues are noticed. These issues are as follows: Lack of precision and Ambiguity.

This paper aims to highlight the differences between UML class diagram and LePUS3 in capturing the structure of security patterns. The main comparison criteria are abstraction and formality.

UML

Unified Modelling Language (UML) is a widely used and accepted software modelling language [7]. It has a number of diagrams used for structural and behavioural representations of the object-oriented software system aspects such as class diagram, sequence diagram, activity diagram, state chart diagram and others. Since UML introduction, it has been an interesting area for many researchers as well as industry people. As a result, an enormous number of a host of studies has been carried out to develop it. This led to introducing versions of UML such as UML 1.0 and UML 2.0. Mainly, a UML version is introduced to overcome problems in the previous version. Since the adoption of UML by Object Management Group (OMG), OMG become main organization in UML development [8]. Although UML is a modelling standard widely used in industry, it raises many concerns. First, it is not formally defined. Second, it lacks precision. These issues made it difficult for UML to support rigorous analysis [9]. Therefore, the problem of ambiguity and incompleteness in the UML diagrams is still persistent. As a result, it is considerably hard to assure that the end design is consistent with the final product [10].

LePUS3

Codecharts are the statement of static design encoded in OOP languages. The language of Codecharts, LePUS3, is an object-oriented design description language. LePUS3 is built on top of the strength of other specification and modelling languages. Mainly, LePUS3 came to address the following concerns [11]:

1. Rigour;
2. Scalability;
3. Minimality;
4. Program Visualization;
5. Automated Verifiability.

LePUS3 offers solutions for the aforementioned concerns. For example, LePUS3 specification (CodeCharts) is graphical. This addresses the Program Visualization concern. On the other hand, LePUS3 uses first order predicates calculus to formalise each element in the specification. This makes it a formal (rigour) graphical language. With having such formal specifications, design conformance verification process can be automated in a more accurate and confident way. This addresses the concern of Automated Verifiability. However, this does not mean that this comes at the expense of minimality and elegance which are LePUS3 concerns. Figure 1 shows a summary of LePUS3 visual vocabulary. In addition, LePUS3 provides means of variable in order to represent design motifs. This offers an opportunity to specify static structure without constraints of naming. Furthermore, LePUS3 introduces an abstraction mechanism which is highly beneficial when modelling large scale software (scalability). Abstraction mechanism relies on the notion of dimension, hierarchy and transitive relations [12]. Moreover, LePUS3 has a tool support which is called Two Tier Programming toolkit (TTP toolkit) [13].

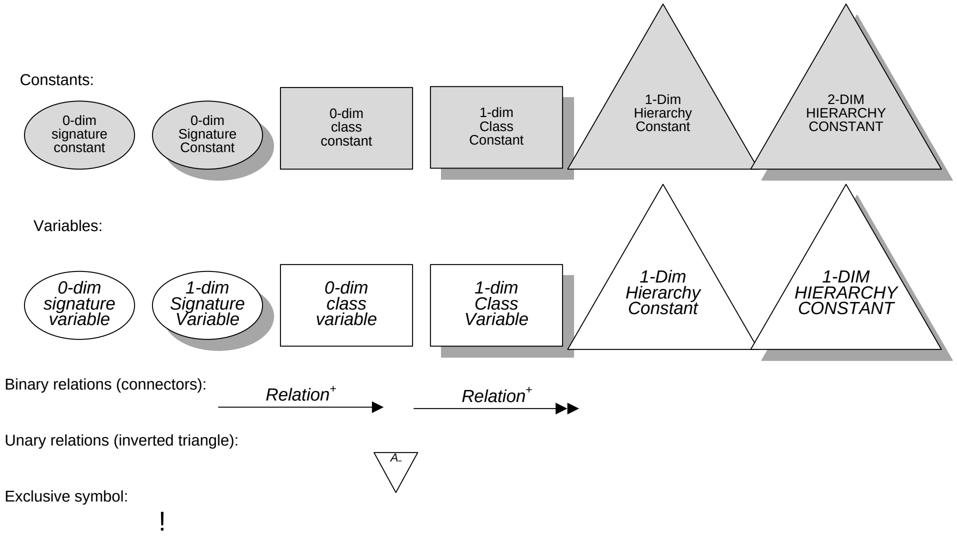


Figure 1. LePUS3 visual vocabulary [12]

Security patterns

Security patterns are solutions to solve reoccurring security problems in specific domain [2, 3]. They are aimed to combine the experience and knowledge of both security experts and software engineers to build a reusable way to solve specific security problem. There are a growing number of catalogues [2, 3, 6, 1, 5, 4] which describe security patterns. However, it has been noticed that they contain some ambiguity in their descriptions of the patterns [14]. Security patterns are analysed in many sources to have both structural and behavioural specifications. They have been described using UML notations and diagrams. This has a number of drawbacks such as the difficulty or impossibility of automating the verification process. From this emerges the need of using a formal design description language in order to describe the security patterns for further investigations. Currently, security patterns are holding many researchers' interests in terms of evolution, classification, modelling, formalisation, verification and detection. However, this field of research is not covered sufficiently. It is important to mention here that when saying formalisation, verification and/or detection of security patterns, it is referred to formalisation, verification and/or detection of security patterns solutions.

Comparison

In order to allow understanding, implementing, verifying and/or detecting instances of security patterns, it is important to model them in a graphical, formal and elegant design description language. UML is graphical modelling language. This makes it more useful and comprehensive when describing any design pattern. In another word, it is better than using a mathematical or textual representation for pattern solution. However, here we compare UML with LePUS3 in modelling security patterns' structure. This means modelling the static structure of the pattern's solution. The comparison will highlight the difference between the two languages (UML and LePUS3) in promoting precision and solve ambiguity.

It might be argued, why to use LePUS3 while UML class diagrams already exist in the descriptions of security patterns. There are a number of reasons behind using LePUS3 instead of UML class diagram. These reasons are as follows:

1. Scalability and abstraction;
2. Formality and tool support;
3. Some of LePUS3 properties which are not in UML class diagram.

Scalability and abstraction are important properties in the modelling language. They are tangibly beneficial especially when modelling large-scale software. LePUS3 has a number of elements (vocabulary) which demonstrate abstraction when modelling. For the context of modelling security patterns, we are interested in the following elements (vocabulary): a) 1-Dim Hierarchy; b) 1-Dim Signature; c) 1-Dim Class. While studying security patterns catalogues and descriptions, it was noticed that some patterns are described in an abstract way. For example, one statement describing Single Access Point Pattern is "Participants: Internal Entities (are all components located inside the system's boundaries)" [1]. This statement is clearly abstracting all the internal components in one entity. However, the UML class diagram, Figure 2, models this statement in a number of classes and relations symbols. This does not reflect the abstraction given in the description of the pattern. However, using LePUS3, it is possible to abstract unnecessary information given in UML class diagram. This can be done by using the 1-Dim Signature and 1-Dim Class vocabulary. This is illustrated in Figure 3.

Another example of abstraction is that in the description of Intercepting Validator Pattern in [2], the statement of "InterceptingValidator retrieves the appropriate validators according to the configuration for the target". Clearly indicates the use of dynamic binding, especially as it is also stated, in the description, that "InterceptingValidator invokes a series of validators" and "Each Validator validates and scrubs the request data, if necessary". The UML class diagram modelling this statement has gone far too abstract as shown in Figure 4. It can be seen that in order to model this statement in Figure 4, InterceptingValidator has a dependency relation with Validator labelled by "creates". This might lead to the misunderstanding that there is only one class called "Validator" and it encapsulates all validation process.

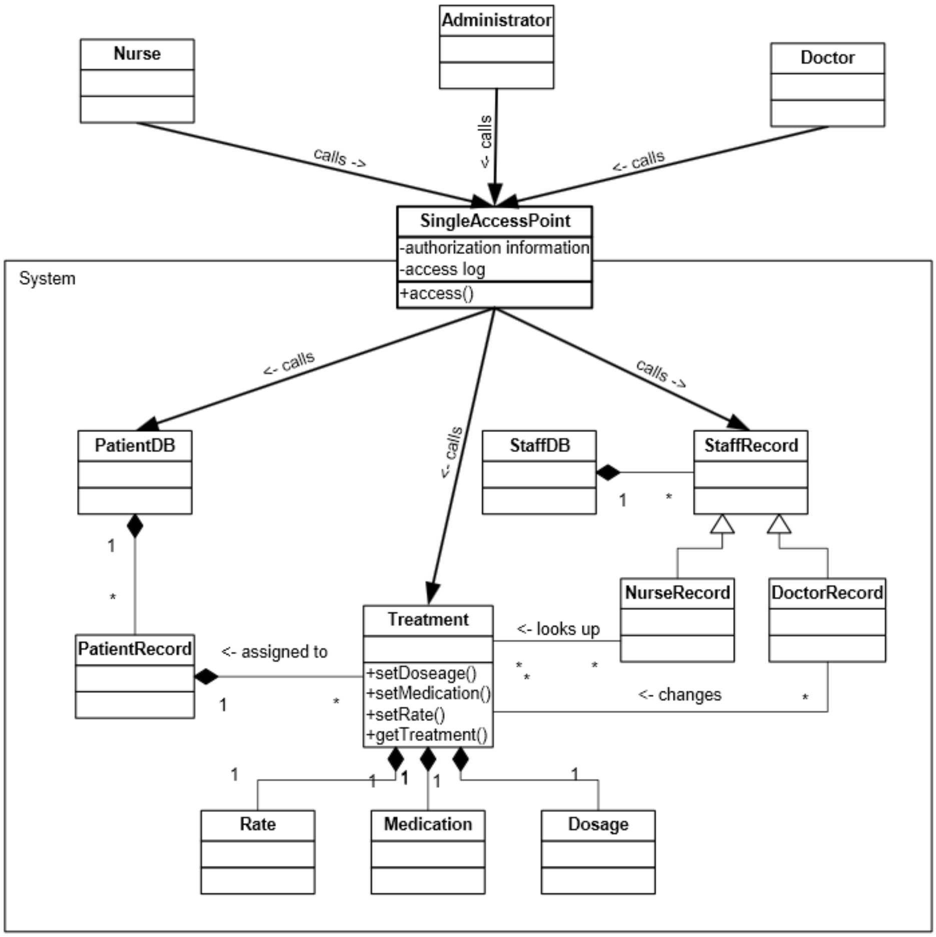


Figure 2. UML class diagram: Single Access Point [1]

If considering modelling them in a better way using UML, the above discussed statements in Intercepting Validator Pattern description could be modelled as shown in Figure 5. However, this would also not help with capturing the abstraction of the statement in the pattern's description. It is clear that 2 more class symbols (besides the existing one) are introduced to illustrate the idea of different validators. On the other hand, in LePUS3, it is possible to model the same statement using only one symbol which is a 1-Dim Hierarchy symbol as shown in Figure 6.

In conclusion, if the UML class diagram does not reflect the patterns structure description, this will introduce an issue of precision. UML class diagram is a good way to represent the structure of a pattern. However, it does not help with modelling abstract structure effectively. Whereas, LePUS3 can model abstract structure in a

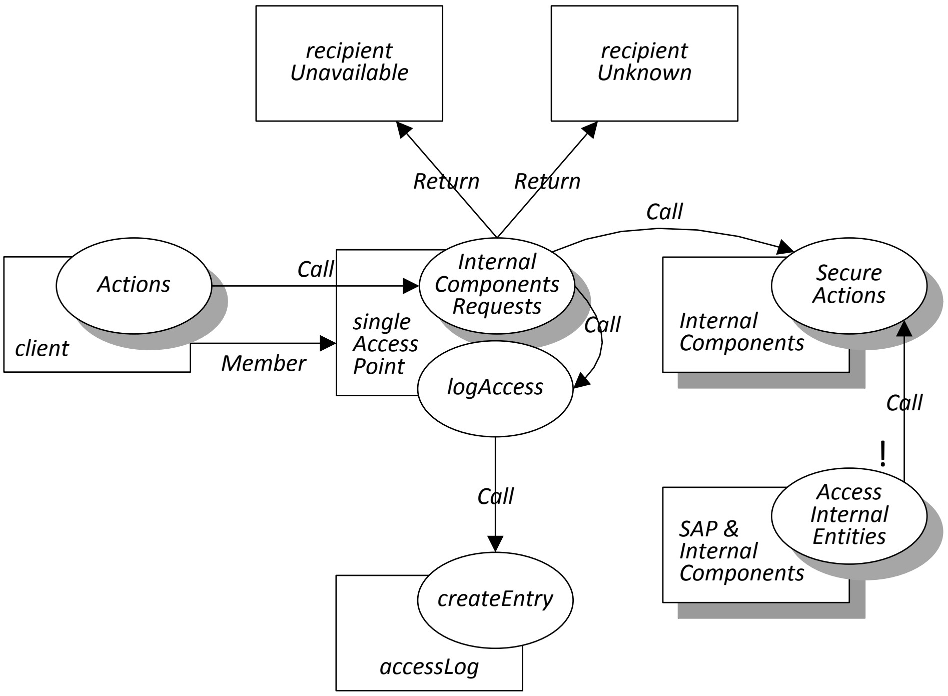


Figure 3. Single Access Point Codechart [1]

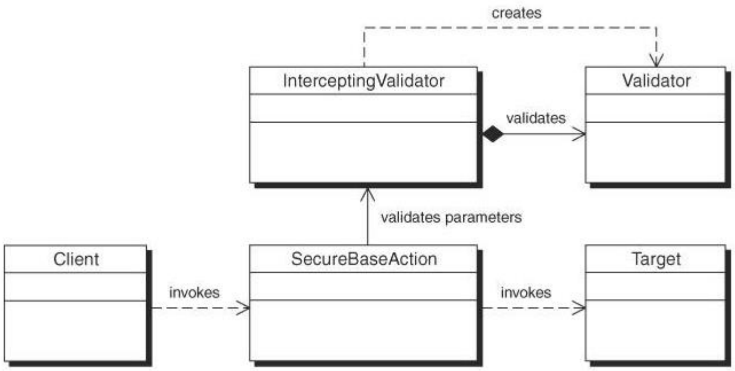


Figure 4. UML class diagram: Intercepting Validator Pattern [2]

more elegant and effective manner as LePUS3 provides few symbols which can be used to represent abstract aspects of the pattern's structure

UML is an informal modelling language [9]. This means that is not based on a formal semantics. Therefore, it cannot be used to represent the design or design

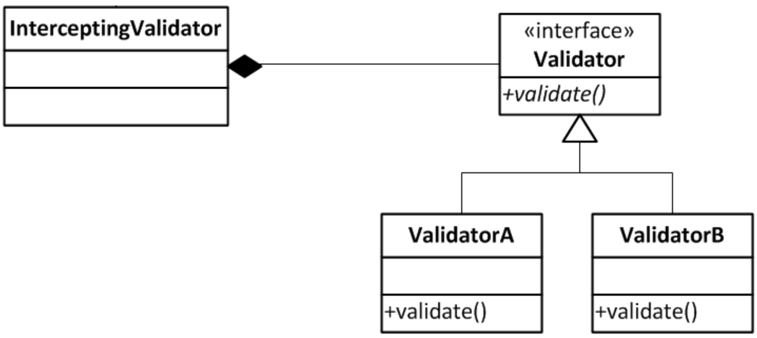


Figure 5. UML class digram for modelling different Validator in Intercepting Validator Pattern

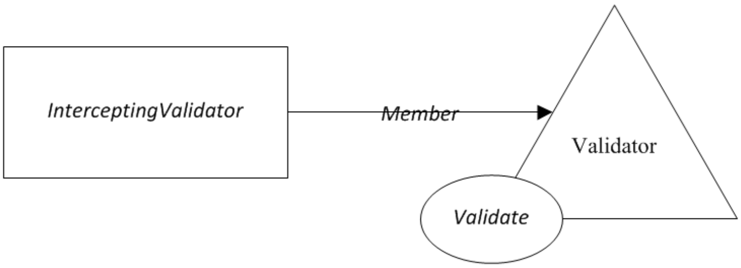


Figure 6. LePUS3 for modelling different Validator in Intercepting Validator Pattern Codechart

pattern formally. Formal representation of the pattern allows many things to be carried out on it. For example, it allows searching for the instance of the pattern in the source code. It also allows verifying source code conformance to the pattern.

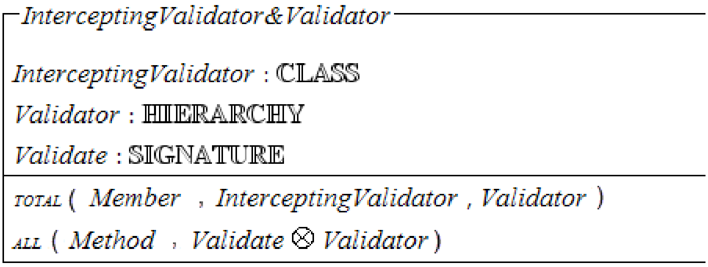
However, many researchers [10, 15, 16, 17, 18, 19, 20, 21] have introduced their approaches to add formality to the UML diagrams. Several formal languages have been proposed to formalise the UML semantics. The most dominant languages are OCL, Description Logic (DL) languages and Z. However, these languages differ in the capability of representing UML properties.

For the context of representing formally the structure of the security pattern form UML class diagram, using these languages introduces a risk of losing information as the languages might not be able to capture some UML properties, for instance, UML dependency in DLs languages [10] and interface notation in Z [22].

Another issue with using these languages is the problem of tools supporting automatic translation of UML class diagram to the desired formal language. In the case of DL languages, the most well-known tools are FACT [23] and RACER [24].

However, these tools are suffering from a considerable efficiency issue as well as they do not support all UML properties [15]. In the case of using Z language, many have introduced their approaches to translate UML class diagram to Z. However, these approaches have not been combined with tools to automate the translation process or the tools are not available to be tried and tested.

On the other hand LePUS3 diagrams stands on a formal language which is First Order Logic Predicate (FOLP). Each element shown on the LePUS3 diagram (Codechart) has a formal meaning and semantic. So, this shortcuts the process of detecting instances of the pattern structure as there will be no need for finding a suitable translation language and a tool for automating the translation process. Moreover, LePUS3 has a tool support, TTP Toolkit [13], which offers functionality such as automatic translation of LePUS3 diagram, reverse engineering of java-based source code and verification of LePUS3 diagram against source code. In order to show this tool in action we auto generated the formal representation of Figure 6 and the result obtained from the tool is in Figure 7.


InterceptingValidator&Validator

InterceptingValidator : CLASS

Validator : HIERARCHY

Validate : SIGNATURE

TOTAL (*Member* , *InterceptingValidator* , *Validator*)

ALL (*Method* , *Validate* $\otimes$ *Validator*)

Figure 7. Auto-generated LePUS3 schema for Codechart in Figure 6

Last aspect making LePUS3 better way for modelling security pattern structure is that LePUS3 can model some structural aspect which UML class diagram cannot. LePUS3 has an interesting symbol called exclusivity symbol which indicates either LEFT exclusive or RIGHT exclusive predicate [12]. This symbol serves information hiding or neglecting when modelling a pattern. For example, in the description of Single Access Point Pattern (SAP) in [1], it is expressed that the internal components can communicate with each other and/or with the single access point, however, external entities cannot communicate with the internal components except via single access point participant in Figure 2, this has not been represented effectively. In addition, it might not be possible to translate into any formal language as the UML class diagram has no notation expressing such constraint. This issue can be solved when using LePUS3 to model the pattern as shown in Figure 3.

Conclusion

In conclusion, UML class diagram is a useful way for capturing security design pattern structure. However, it is clear that LePUS3 Codecharts can solve some issues which UML class diagram cannot. So, this means that UML class diagram is not the best way to capture the structure of security patterns as system security might be at risk. In addition, LePUS3 stands on a formal semantics which allows automating detection of instances of the pattern structure and verification of these instances against any given implementation.

References

[1] R. Wassermann and B. H. C. Cheng, "Security patterns," in *Michigan State University, PLoP Conf*, 2003.

[2] D. Alur, J. Crupi, and D. Malks, *Core J2EE Patterns: Best Practices and Design Strategies.* Prentice Hall Professional, 2 ed., 2003.

[3] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security patterns. Integrating security and systems engineering.* 2006.

[4] S. Konrad, B. H. C. Cheng, L. A. Campbell, and R. Wassermann, "Using security patterns to model and analyze security requirements," *Requirements Engineering for High Assurance Systems (RHAS'03)*, p. 11, 2003.

[5] D. M. Kienzle, M. C. Elder, D. Tyree, and J. Edwards-Hewitt, "Security patterns repository version 1.0," *DARPA, Washington DC*, 2002.

[6] J. Yoder and J. Barcalow, "Architectural patterns for enabling application security," *Urbana*, vol. 51, p. 61801, 1998.

[7] D. Pilone, *UML 2.0 pocket reference.* O'Reilly Media, 2006.

[8] OMG, "OMG unified modeling Language^{TM} (OMG UML), infrastructure," tech. rep., 2011.

[9] W. Yan and Y. Du, "Research on reverse engineering from formal models to UML models," pp. 406–411, 2010.

[10] W. Kadir, W. M. Nasir, and R. Mohamad, "Formalization of UML class diagram using description logics," 2010.

[11] E. Gasparis, J. Nicholson, and A. Eden, "LePUS3: an object-oriented design description language," in *Diagrammatic Representation and Inference* (G. Stapleton, J. Howse, and J. Lee, eds.), vol. 5223 of *Lecture Notes in Computer Science*, pp. 364–367, Springer Berlin / Heidelberg, 2008.

[12] A. H. Eden and J. Nicholson, *Codecharts: Roadmaps and Blueprints for Object-Oriented Programs.* John Wiley & Sons, May 2011.

[13] J. Nicholson and A. Eden, "TTP toolkit - home - object-oriented design, visual modelling, formal specification, automated verification, reverse engineering, design mining, traceability, scalability," 2009.

[14] M. Bunke and K. Sohr, "An architecture-centric approach to detecting security patterns in software," in *Engineering Secure Software and Systems* (A. Erlingsson, R. Wieringa, and N. Zannone, eds.), vol. 6542 of *Lecture Notes in Computer Science*, pp. 156–166, Springer Berlin / Heidelberg, 2011.

[15] D. Berardi, D. Calvanese, and G. De Giacomo, "Reasoning on UML class diagrams using description logic based systems," in *Proceedings of the "Workshop on Applications of Description Logics"*, vol. 44, 2001.

[16] S. Flake and W. Mueller, "An OCL extension for real-time constraints," in *Object Modeling with the OCL*, pp. 150–171, Springer, 2002.

[17] S. Sengupta and S. Bhattacharya, "Formalization of UML diagrams and their consistency verification: A z notation based approach," in *Proceedings of the 1st India software engineering conference*, ISEC '08, (New York, NY, USA), pp. 151–152, ACM, 2008.

[18] A. Mostafa, M. Ismail, H. El-Bolok, and E. Saad, "Toward a formalization of UML2.0 metamodel using z specifications," in *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007*, vol. 1, pp. 694–701, 2007.

[19] Z. Zhihong and Z. Mingtian, "Some considerations in formalizing UML class diagrams with description logics," in *2003 IEEE International Conference on Robotics, Intelligent Systems and Signal Processing, 2003. Proceedings*, vol. 1, pp. 111–115, 2003.

[20] A. Cali, D. Calvanese, G. D. Giacomo, and M. Lenzerini, "A formal framework for reasoning on UML class diagrams," in *Foundations of Intelligent Systems*, pp. 503–513, Springer, 2002.

[21] H. M. Chavez and W. Shen, "Formalization of UML composition in OCL," in *Computer and Information Science (ICIS), 2012 IEEE/ACIS 11th International Conference on*, pp. 675–680, 2012.

[22] G. Smith, *The Object-Z specification language*, vol. 101. Citeseer, 2000.

[23] I. Horrocks, U. Sattler, and S. Tobies, "Practical reasoning for expressive description logics," in *Logic for Programming and Automated Reasoning*, pp. 161–180, 1999.

[24] V. Haarslev and R. Muller, "RACER system description," in *Automated Reasoning*, pp. 701–705, Springer, 2001.

Authors

Abdullah A H Alzahrani — the 2nd year Ph.D student, School of Computer Science and Electronic Engineering, University of Essex, Colchester, United Kingdom; E-mail: aahalz@essex.ac.uk

Amnon Eden — Lecturer, School of Computer Science and Electronic Engineering, University of Essex, Colchester, United Kingdom; E-mail: eden@essex.ac.uk