A Retrospective on OOADARE as an Automated Object-Based Approach for Requirements Engineering

Amal Khalil, Hajar Lamsellak, Zineb Bougroun, Mohammed Saber, and Mohammed Ghaouth Belkasmi

Abstract In agile methods, the communication between customer and development team is ensured by requirements, most often presented in an unstructured textual format, which frequently involves redundancy or non-precision information. This leads, in practice, to poor system quality, especially if we use classical approaches such as scenario-based approach. Yet, OOADARE approach is introduced in this way, using semi-structured text models in the form of user stories and constraint story cards (CSC), to automate the object-oriented transformation of requirements into a class diagram. However, the approach failed to capture all the elements needed to construct a correct and complete class diagram. This paper, at that point, proposes templates in natural language, which are part of the same perspectives of CSCs proposed by OOADARE, namely semi-structured text, to fill these gaps and ensure the completeness of the class diagrams thus generated.

Keywords Requirements engineering · Object-oriented application · User stories · Constraint story card (CSC) · Transforming requirements · Class diagram

A. Khalil (✉) · H. Lamsellak · Z. Bougroun · M. Saber · M. G. Belkasmi
SmartICT Laboratory, Mohammed First University Oujda, ENSAO, Oujda, Morocco
e-mail: amal.khalil@ump.ac.ma

H. Lamsellak
e-mail: lamsellak.hajar@ump.ac.ma

Z. Bougroun
e-mail: z.bougroun@ump.ac.ma

M. Saber
e-mail: m.saber@ump.ac.ma

M. G. Belkasmi
e-mail: m.belkasmi@ump.ac.ma

H. Bekkay et al. (eds.), *Proceedings of the 3rd International Conference on Electronic Engineering and Renewable Energy Systems*, Lecture Notes in Electrical Engineering 954, https://doi.org/10.1007/978-981-19-6223-3_8

1 Introduction

Requirements engineering is the process of eliciting, analyzing, specifying, validating, and managing software requirements [1]. It is not an isolated front-end activity to a software lifecycle process; rather, it is an integral part of the larger process connected to other parts through continuous feedback loops [2]. The scenario-based approach to expressing requirements is the approach most used by developers [3, 4]. After eliciting requirements, they are structured in the form of user stories, use cases, sequence diagram, etc. Those forms are not automatically mapped to the object model [5]. object-oriented analysis and design approach for the requirements engineering (OOADARE) approach, which is part of the design approach trend aimed at automating the generation of analysis diagrams [6–8], fits to overcome these limits, so it took user stories and constraint story cards (CSC) to automatically generate the objects, their actions, and the rest of the software artifacts (MVC, DAO, Test…) [9].

The purpose of this study is to follow the same approach to ensure its completeness and therefore determine the remaining gaps, which should be filled in order to advance the current research concretely.

2 OOADARE Approach

2.1 Presentation

A sequence of transformations is performed to build a software system, starting from requirements and ending with implementation. However, requirements are mostly in the form of an unstructured text, but not a model that can be easily understood by computers [10]. OOADARE is an approach based on the OOAD approach evolved by Booch [11], which requires, among its constraints, the use of a semi-structured text to extract structured models [12]. OOADARE contributes to the automatic transformation of requirements into a class diagram and offers technical concepts and practices that facilitate the development flow and improve the quality of the software product [5, 9]. It introduced a template that is based on the well-known user story template for solving the disorganization issue [9].

2.2 Overview of the "User Story" Template

After retrieving the requirements, the OOADARE approach requires preparing them in user stories to move on to identifying objects and methods [5].

User stories are a popular method for representing requirements using a simple template [13]. Their adoption is growing [14] and is massive especially in the context of agile software development [15].

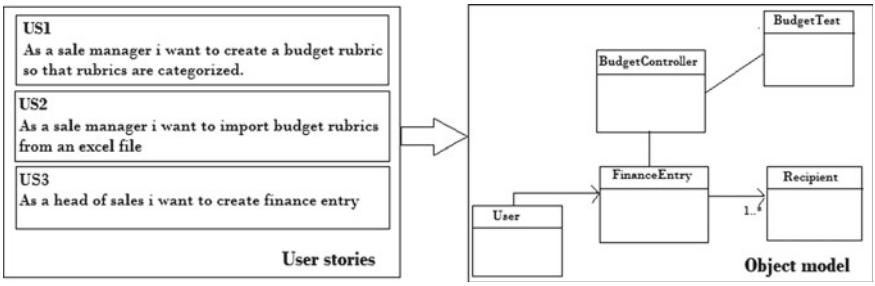

Fig. 1 Using "user stories" to generate an "object model" [9]

The OOADARE approach takes "user stories" as an entry point to automate the identification of objects and actions of the object model, which implies respecting a very precise syntax [9]:

$$As\,a\ <\ role\ >\ I\,can\ <\ \boldsymbol{action}\ ><\ \boldsymbol{object}\ >\ so\,that\ <\ business\,value\ >$$

The diagram thus generated specifies the classes of objects and the methods (Fig. 1).

The "user stories" only allow to generate the classes, and the methods of a class diagram, the "constraint story card" (CSC), are then used to manage the associations.

2.3 Using CSC to Generate Associations

Constraint story card (CSC) is a story card in which we can write the requirement's constraints in a human-like object constraint language (OCL) [16], that represents a precise text language that provides constraint and object query expressions on any meta-object facility model or meta model with a formal specification language [17].

The OOADARE approach suggests four templates to report the different associations within the UML class diagram.

The four templates use a generic formula as shown below [18]:

$$[Role]\ <\ \boldsymbol{Source\,Object}\ ><\ \boldsymbol{Association\,expression\,type}\ >\ \left[\boldsymbol{Cardinality}\right][Role]\ <\ \boldsymbol{Target\,Object}\ >$$

The "Association expression type" argument corresponds to four different values depending on the type of the association [18]:

- [***Has or a verb***]: in this case, it is a simple association
- [***Is a***] is the proposed model for inheritance
- [***Contains***]: to illustrate aggregation between classes
- [***Is composed of***]: for the composition model.

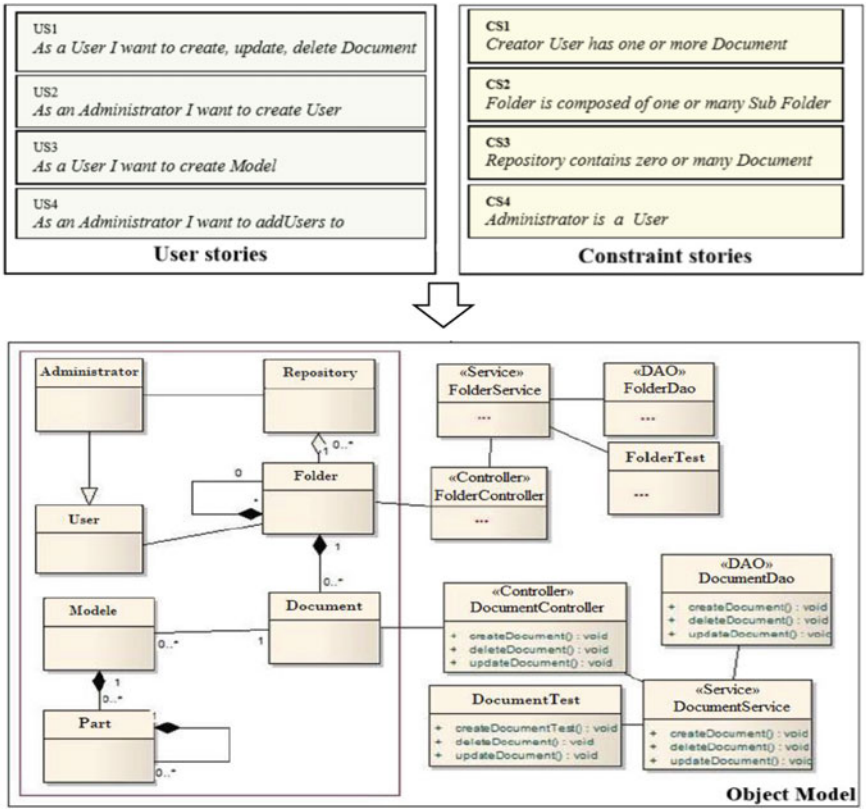

Fig. 2 Class diagram generated from user stories and CSC using the OOADARE approach. *Other artifacts (DAO, Controller …) generated were omitted for a clarity purpose.* [18]

The "Cardinality" can be expressed in ***letters*** (from zero to ten in English), ***numbers*** or by the word "***many***". When the minimum and the maximum are not the same, the two sides are separated by the word "***or***" (Fig. 2) [18].

Nevertheless, the mapping model proposed by OOADARE, in compliance with the constraints and templates, remains incomplete since it does not take into consideration all the elements that can be found in specifications and which are essential for software development.

3 Filling the Gap in the OOADARE Approach

As mentioned above, important elements are missing in the class diagram generated using OOADARE, which take their place in the center of the requirements and without which we cannot speak of a complete class diagram. It lacks, first,

attributes that represent the variables storing state information characterizing the object. Secondly, we notice the absence of association classes in which "attributes" are directly dependent on two other classes and could not be placed in either one or the other. We propose in the following part solutions to fill this gap and ensure the completeness of the method with indispensable features.

3.1 Generating Attributes in a Class Diagram

In domain modeling class diagrams, an attribute represents a data definition for an instance of a classifier. An attribute describes a range of values for that data definition. A classifier can have any number of attributes or none at all. Attributes describe the structure and value of an instance of a class [19].

As a part of CSC, and taking into account textual expressions often used to present the different properties related to an object, we consider the following model to extract the different attributes of a class:

> ***[Each]*** *< Object >* ***has\| is identified by\| is represented with\| is described by*** *< Attribute 1 > , < Attribute 2 > , …**and*** *< Attribute n >*

where

- **Object** represents the entity for which the properties are identified
- **Attribute *n***: the nth single, named fragment of the persistent state.

3.2 Consider Association Classes in Class Diagram Generation

In UML diagrams, an association class is a class that is part of an association relationship between two other classes. It is identical to other classes and can contain operations, attributes, as well as other associations [20]. We can attach an association class to an association relationship to provide additional information about the relationship [17].

This case occurs in a "many to many" association, so we must take into account this type of association before looking for the association attribute(s) through the template that we propose as CSC (Table 1):

Table 1 Formula parts description

Formula argument	Description
Source object	The source object of an association
Attribute	The single, named fragment of the persistent state
Target object	The target object of an association

*[**Each**] < Source Object > **has**| **have** < Attribute > (**for**| **in**) **each** < Target Object >*

3.3 Example of Application to a Case Study

Consider the following set of requirements for a university information system that is used to keep track of student's transcripts: *"…Each department is described by a name, a code, an office number, an office phone and a college. Each course has a course name, description, course number, credit hours, level, and offering department. Each student is identified by a number, a name, a birthdate, a gender, a social security number, current address and phone number. To ensure that the system generates a transcript for students, each student must have a grade for each course.…"*.

To perform the extraction task of UML class diagram concepts, each sentence in the input text will be matched against the different lists of patterns that have been defined by the above models.

If we analyze the sentence "*Each department is described by a name, a code, an office number, an office phone and a college.*', we notice that it matches very well with the attributes pattern (Fig. 3); it begins with a "number of characters" that mentions the concerned "object," followed by the key words "is identified by," then followed by "a number of characters" that list the specified attributes, which gives us the resulting class "Department" with its correspondent attributes.

Then, considering the case of the last sentence presented by the example: '*To ensure that the system generates a transcript for students, each student must have a grade for each course.*', we note its correspondence to the second pattern proposed for the generation of association classes (Fig. 4).

It should be noted that in this case, the association between the two classes (student and course) has on both sides a maximum multiplicity equal to "many" represented by the symbol "*", which means that a "student" studies one or more "courses" and that a "course" is taught for one or more "students".

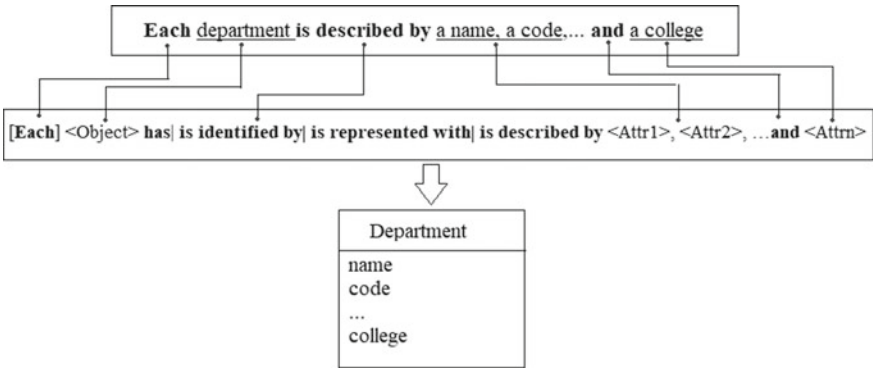

Fig. 3 Diagram class illustrating an entity with attributes as mapped from a CSC template

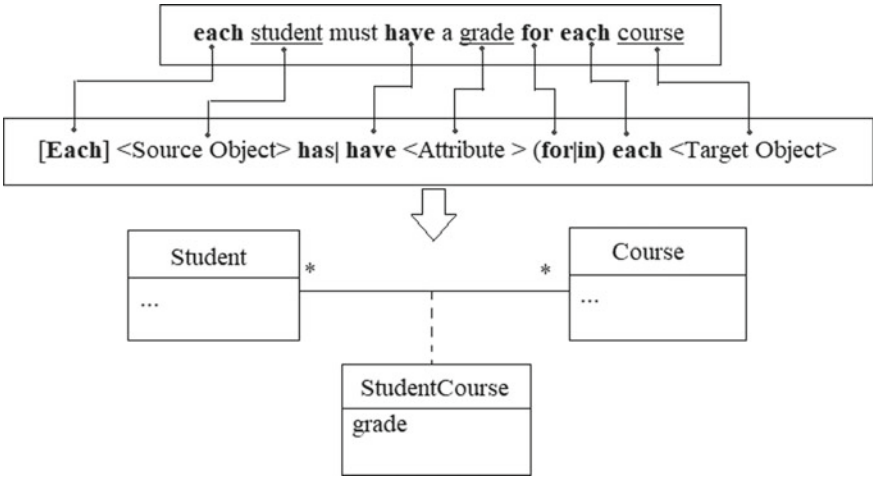

Fig. 4 Class association extraction pattern example

4 Conclusion

More efficient and more flexible technologies are accelerating the growth of fully automated production facilities. This is the context of our study which aims to improve the OOADARE, an approach targeting the automatic generation of class diagrams from requirements presented as semi-structured text templates.

In this paper, we started by presenting the general context of the study, then explaining the approach used by OOADARE which offers an intermediate representation of the requirements in the form of user stories and CSC, thus allowing an automatic generation of class diagrams. Although we discussed the failures of the approach to finally propose solutions that follow the same rules and constraints.

With the aim of eliminating ambiguity and redundancy, the templates proposed by OOADARE in addition to the models proposed in this document are relatively simple to build and to deduce from the requirements expressed either in textual or oral form.

References

1. Lucassen G et al (2016) The use and effectiveness of user stories in practice. In: International working conference on requirements engineering: foundation for software quality. Springer, Cham
2. Macaulay LA (2012) Requirements engineering. Springer Science & Business Media
3. Sutcliffe A (2003) Scenario-based requirements engineering. In: Proceedings of the 11th IEEE international requirements engineering conference, 2003. IEEE
4. Kim M et al (2007) Managing requirements conflicts in software product lines: a goal and scenario based approach. Data Knowl Eng 61(3):417–432
5. Zeaaraoui A, Bougroun Z, Belkasmi MG, Bouchentouf T (2012) Object-oriented analysis and design approach for the requirements engineering. J Electr Syst 2(4):147–153
6. Elallaoui M, Nafil K, Touahni R (2018) Automatic transformation of user stories into UML use case diagrams using NLP techniques. Procedia Comput Sci 130:42–49
7. Fischbach J, Frattini J, Vogelsang A (2021) Cira: a tool for the automatic detection of causal relationships in requirements artifacts. arXiv Preprint. arXiv:2103.06768
8. Yang Y et al (2019) RM2PT: a tool for automated prototype generation from requirements model. In: 2019 IEEE/ACM 41st international conference on software engineering: companion proceedings (ICSE-Companion). IEEE
9. Zeaaraoui A, Bougroun Z, Belkasmi MG, Bouchentouf T (2013) User stories template for object-oriented applications. In: Third international conference on innovative computing technology (INTECH 2013)
10. Yue T, Briand LC, Labiche Y (2011) A systematic review of transformation approaches between user requirements and analysis models. Requirements Eng 16:75–99
11. Booch G et al (2008) Object-oriented analysis and design with applications. In: ACM SIGSOFT software engineering notes, vol 33, no 5, pp 29–29
12. Sanyal R, Ghoshal B (2018) Automatic extraction of structural model from semi structured software requirement specification. In: 2018 IEEE/ACIS 17th international conference on computer and information science (ICIS). IEEE
13. Cohn M (2004) User story applied: for agile software development. Addison-Wesley, Boston, MA
14. Kassab M (2015) The changing landscape of requirements engineering practices over the past decade. In: Proceedings of EmpiRE. IEEE, pp 1–8
15. Wang X, Zhao L, Wang Y, Sun J (2014) The role of requirements engineering practices in agile development: an empirical study. In: Zowghi D, Jin Z (eds) APRES 2014, vol 432. CCIS. Springer, Heidelberg, pp 195–209
16. Warmer J, Kleppe A (2003) The object constraint language—getting your models ready for MDA, 2nd edn. Addison Wesley
17. OMG (2017) Unified modeling language specification version 2.5.1
18. Dahhane W et al (2014) An automated object-based approach to transforming requirements to class diagrams. In: 2014 Second world conference on complex systems (WCCS). IEEE
19. IBM Documentation. Rational software architect standard edition (Version 7.5.0)
20. IBM Documentation. Rational software modeler (Version 7.5.0)