An automated object-based approach to transforming requirements to class diagrams

Walid DAHHANE*, Adil ZEAARAOUI, El Hassane ETTIFOURI, Toumi BOUCHENTOUF
Team SIQL, ENSA Oujda
Mohamed First University
Oujda postal 60000
MOROCCO
dahhane.walid@gmail.com

***Abstract*—Requirements Engineering (RE) is a set of activities concerned with identifying and communicating the purpose of a software system and its contexts of use as well. The requirements explain the purpose of a particular system as well as its suitability in fitting the desired purpose. Hence, the loss of information and precision, during the construction phase, leads to a poor quality zone of the system's target. Moreover, this problem becomes obvious in practice when using the classical approaches (e.g. scenario-based approach). Thus, OOADA-RE (Object-oriented Analysis and Design Approach for Requirements Engineering) has been introduced with the aim of breaking the gap between RE and other phases. It demonstrates a way to handling object classes based on the classical template of the user story, which is enhanced as follows: *'"As a <role>, I want to <action> <object>, so that <business value>"'*; this is to allow an automated creation of classes with objects and actions mapped and gathered straight from the requirements. User stories are also used as a communication mean with customer.**

However, the approach has not offered a way yet to handling associations between the mapped class diagram's objects. This paper aims to tackle this issue and introduce the Constraint Story Card (CSC) upon which we suggest four templates in a natural language so as to manage the standard UML associations (Simple association, Inheritance, Aggregation and Composition) including association's roles and cardinality management. We conclude this paper with an illustrating example to clarify our main idea.

Keywords—Constraint Story Card (CSC); OOADA-RE; Associations; Requirement Engineering; User Story;User Story Template; Model Transformation;Transforming requirements to class diagrams

I. Introduction

Requirements engineering bridges the gap between an initial vague recognition of some problems to which we can apply computer technology and the task of building a system to address the problem, which is designed and has an intended purpose ([3],[4]). Hence, if inputs (requirements) are poor, the final software is automatically poor ([1],[2]).

Our approach ([1],[2]) intends to early involve the customer in the system construction. It also insists on the mechanisms that allows a shared understanding of the requirements between different stakeholders. It's so important in our vision that the customer validates the requirements as early as possible so that the mapping to the object-oriented universe reflects the desired system. The approach is based on agile concepts and more precisely on user story template. Nevertheless, the proposed mapping in OOADA-RE to object oriented universe complied with classes and methods; yet, it did not present a way to handling associations between classes.

In this paper, we are explaining our approach of automating transformation from requirements to class diagrams. We suggest a new story card (CSC) with four templates in a natural language easy to understand and even to write by the system's customer. These templates cover the standard UML associations (Simple association, Inheritance, Aggregation and Composition) and deals with roles and cardinality management. The following section presents some related works of the main subject. Next, we return back to the OOADA-RE approach and the use of a user story template so as to map objects from requirements, and we present the difficulty of enhancing this template to handle associations in the suggested template without influencing its simplicity. Section 4 discusses the suggested story card and presents the four templates. In section 5, we present an example in order to illustrate the purpose of the study. Finally, section 6 presents some concluding remarks and future works.

II. Related work

Several researches have attempted to find a way to automate transformation from software requirements to class diagrams ([12],[14],[15],[18],[19],[20],[21],[22],[23],[24],[25]). Some researches essentially use formal language with natural appearance to express requirements, whereas, other ones try to process natural language using opportunistic parsing techniques. Nonetheless, we have not seen those researches dealing with agile manifesto, which is our main aim.

Zhou and Zhou [17] suggest an approach that uses NLP and domain ontology to automate class diagram generation from free-text requirements. First, some initial processing of the natural language requirements is done with the help of some linguistic patterns. This step extracts candidate classes. Then, a dictionary of domain ontology, provided in some specific format as input, is used to refine the result.

Mich L. [20] suggests a NLP system called LOLITA, which generates an object model automatically from natural language. This system considers nouns as objects and use links to find relationships amongst objects. LOLITA system is built on a large scale Semantic Network (SN) that does not distinguish between classes, attributes, and objects. This approach is limited to extract objects and cannot identify classes.

Another interesting work has been done by Christiansen et al. [25] They developed a system to transform use case diagram to class diagram using Definite Clause Grammars extended with Constraint Handling Rules. The grammar captures information about static words (classes and their relations) and subsequently the system generates the adequate class diagram.

Our approach tends to be more agile. It is based on user stories to handle functional requirements. We suggest an intermediate level for requirements representation in the form of story cards that allow us to automate class diagram generation. Those cards connect the natural language level of the end user to object model level produced by engineers.

III. OOADA-RE WITH USER STORY TEMPLATE OVERVIEW

A. Introduction

The OOADA-RE discusses the quality issues, which can occur when moving from the RE phase to the construction phase, due to requirements' misunderstandings([1],[2]). It tackles the difference between the human and the models/diagrams languages which are used in the two previous phases.

In the next paragraphs, we will discuss our work on OOADA-RE presented in our previous papers ([1],[2]).

At first, OOADA-RE suggested an ontology database use which represent different kinds of categories and domain models. To illustrate, during object analysis, the approach requires the Business Analyst to identify objects as well as their corresponding actions. Once objects and actions are identified, the rest of software artifacts (MVC, DAO, Test engineering) are automatically generated, based on the previous defined database [1]. The idea was limited to the main raison that the user inputs can be written in a disorganized manner. Besides, no guidance was proposed so as to get a real oriented-object specification. Thus, this solution was impractical.

B. The intent of using the user story and the associations' issue

In the next paper [2], OOADA-RE introduced a template that it based on the well known user story template (*As a <Role> I want to <Activity > so that <Business value>)* for solving the disorganization issue. The choice of the user story was taken into account, thanks to the cards given power, in order to facilitate the communication with the customer [11]. The proposed template focuses on the middle part (<Activity>) , whose aim is providing the action, that is to be treated within the system. The template was enhanced as follows:

As a <role>, I want to **<action> <object>**, so that <business value>.

This technique allowed a straight forward mapping between the requirements and the class diagram. Nevertheless, the extracted classes had no relationship between each other. Therefore, the Business Analyst was responsible for finalizing associations within the generated diagram's classes. So adding the associations to the template would add more complexities within the user stories; Thus, looking for another technique was not an option.

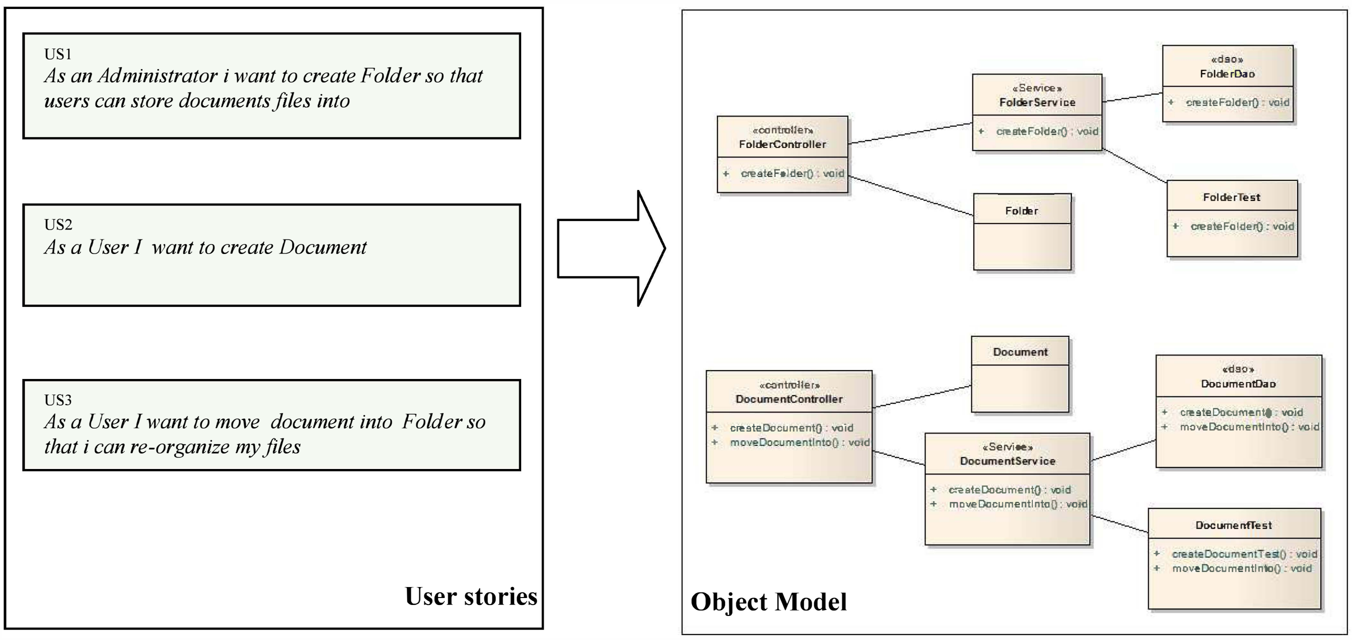

US1
As an Administrator i want to create Folder so that users can store documents files into
US2
As a User I want to create Document
US3
As a User I want to move document into Folder so that i can re-organize my files
User stories
«controller» FolderController
+ createFolder() : void
«Service» FolderService
+ createFolder() : void
«dao» FolderDao
+ createFolder() : void
Folder
FolderTest
+ createFolder() : void
Document
«controller» DocumentController
+ createDocument() : void
+ moveDocumentInto() : void
«Service» DocumentService
+ createDocument() : void
+ moveDocumentInto() : void
«dao» DocumentDao
+ createDocument() : void
+ moveDocumentInto() : void
DocumentTest
+ createDocumentTest() : void
+ moveDocumentInto() : void
Object Model

Fig. 1. Transformation of user stories to object model using the first version of OOADA-RE approach. We can notice that the class Folder and Document have no associations to one another.

IV. CONSTRAINT STORY CARD TEMPLATES

A. Overview

As discussed in the previous section, enhancing the user story template to include associations is inefficient without altering the user stories communication facilities with the customer. Therefore, another technique is needed to address this lack of mechanisms in the mapping between the requirements and the object-oriented diagrams. When we look at UML extensions, we come across the Object Constraint Language (OCL). The OCL is a precise text language that provides constraint and object query expressions on any MOF (Meta-Object Facility) model or meta-model with a formal specification language [9]. Associations can be viewed as requirement constraints (e.g. a user has one and only one email address). Based on the OCL language, we can handle the lack of associations using a new story card that we named it after CSC.

Constraint Story Card (CSC) is a story card in which we can write the requirement's constraints in a human-like OCL language ([5],[6],[7],[16]). We suggest four templates to address the different associations within the UML class diagram. The four templates use a generic formula which has as arguments roles, cardinalities, etc as shown below:

[Role] **<Source Object> <Association expression type>** [Cardinality] [Role] **<Target Object>**

TABLE I. GENERIC FORMULA PARTS DESCRIPTION

Formula argument	**Description**
Source Object	The source object of the association
Role	The association role
Association expression type	*'has' (or a verb),' is a',' contains',' is composed of'*
Cardinality	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, we separate the two sides by the word '*or*'
Taget Object	The target object of the association

More explanations and examples are detailed in the following paragraphs.

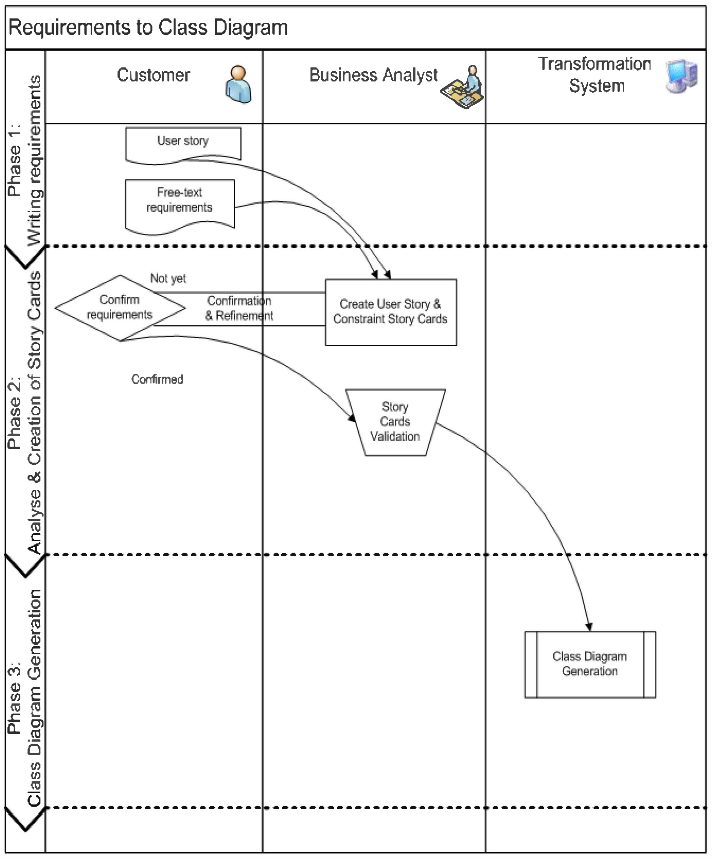
To simplify the workflow of writing and organizing requirements for the customer, we suggest the following process:

1. The customer provides the requirements in the form of free-text specifications or preferably as user stories.

2. The Business Analyst creates adequate story cards based on the customer's inputs.

3. The customer validates and/or question all stories with the Business Analyst in order to avoid ambiguities.

4. Once the story cards are confirmed and validated by the customer and Business Analyst, the system automatically transforms them into a class diagram and generates the appropriate source code.



Requirements to Class Diagram
Customer
Business Analyst
Transformation System
Phase 1: Writing requirements
Phase 2: Analyse & Creation of Story Cards
Phase 3: Class Diagram Generation
User story
Free-text requirements
Not yet
Confirm requirements
Confirmation & Refinement
Create User Story & Constraint Story Cards
Confirmed
Story Cards Validation
Class Diagram Generation

Fig. 2. From requirements to a class diagram using OOADA-RE Process

B. Simple association template

By simple association, we mean a relationship where all objects have their own lifecycle and there is no owner ([8], [9]). For example, let's take a teacher and a student. Many students can be associated with a single teacher, and a single student can be associated with multiple teachers. Yet, there is no ownership between a teacher and a student, and both have their own lifecycle. Both can be created and deleted independently. For those kinds of associations, we can express them as follows:

[Role] <Source Object> **<[Has or a verb]**> <Cardinality> [Role] <Target Object>

For the teacher and the student example, we can fill the template as shown below:

Teacher teaches |has *one or many* **Student**

TABLE II. EXAMPLE OF ASSOCIATION MAPPING IN SIMPLE ASSOCIATION

Template argument	**Mapping**
Source object	Teacher
Role	none
Association expression type	*Simple association using the verb 'teaches' (or the 'has' word)*
Cardinality	Minimum : One student, Maximum: Many
Taget Object	Student

The sentence can be expressed in a different way to introduce roles or to express cardinalities in both sides of the relationship, for example:

Mathematics Teacher teaches five or many Student

and

Student has one **Mathematic** Teacher

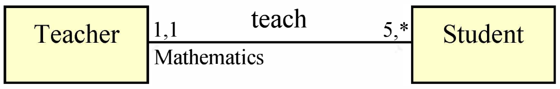


Fig. 3. Diagram class with simple association, cardinalty and role as mapped from the CSC card

C. *Inheritence template*

Inheritance is a parent-child relationship where we create a new class by using an existing class code (i.e. reusing methods, properties and other variables). Inheritance allows us to share some common characteristics or behaviors between classes. In object-oriented paradigm, we identify the inheritance based on IS-A Relationship ([8],[9]). Therefore, the proposed inheritance template is as simple as follows:

<Source Object> **<Is a>** <Target Object>

For example:

Teacher **is a** Person

Student **is a** Person

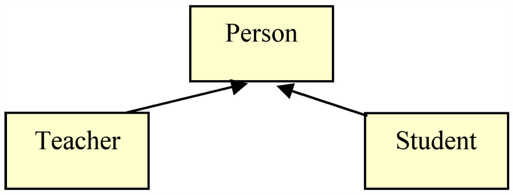


Fig. 4. Diagram class illustrating inhertence of Student and Teacher from Person as mapped from the CSC card

D. *Aggregation template*

In cases where there's a part-of relationship between class A (whole) and class B (part), taking into account that the class B can be also aggregated with other classes in the application, then we can talk about "weak" aggregation (shared association) [8][9].

To illustrate this kind of relationships between classes, we suggest the below template :

[Role] <Source Object> **<Contains>** <Cardinality> [Role] <Target Object>

For example :

Application **contains** one or many Module

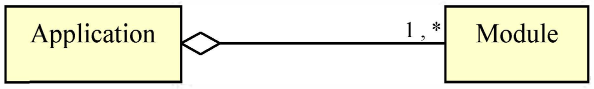


Fig.5 . Class diagram illustrating aggregation between Application & Module as mapped from the CSC card

E. *Composition template*

In cases where, in addition to the part-of relationship between class A and class B, knowing that there's a strong life cycle dependency between the two classes, we should be more specific to use the composition relationship instead of aggregation or association ([8],[9]). That is to say, when the class A is deleted, the class B is automatically deleted. As a result, we suggest the following template:

[Role] <Source Object> **<is composed of>** <Cardinality> [Role] <Target Object>

For example:

Folder **is composed of** zero or many File

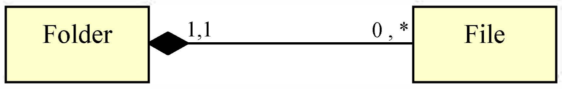


Fig.6 . Class diagram illustrating the composition between Folder & File as mapped from the CSC card

V. ILLUSTRATING EXAMPLE

A. *Customer input*

Let's take the following requirements from a customer: "A user can create documents and share them with other users. A user can modify documents, but cannot delete them only if he is the documents' owner. A document is organized into folders which can have sub folders or documents, and they are created by a user. A repository holds one or many users and one or many folders and documents. A repository has one administrator who is a user system with powerful capabilities like managing users. A Document can be created based on a model, and this model is composed of one or more parts. Every part is composed of one or more sub-parts."

Based on these simple requirements, we will try to apply OOADA-RE approach to generate a class diagram which would reflect the desired system.

B. *Expressing requirements using the OOADA-RE approach*

The first step is to formalize our user story based on the customer requirements; We can already deduct some of them:

As a User I want to create, update, delete Document

As an Administrator I want to create User

As a User I want to create Model

As an Administrator I want to addUsersTo Repository

We continue our formalization process until we have a descriptive user stories of the requested specifications; Then, in the second step, we identify the system constraints that can be modeled as associations. For example:

User has one or more Document

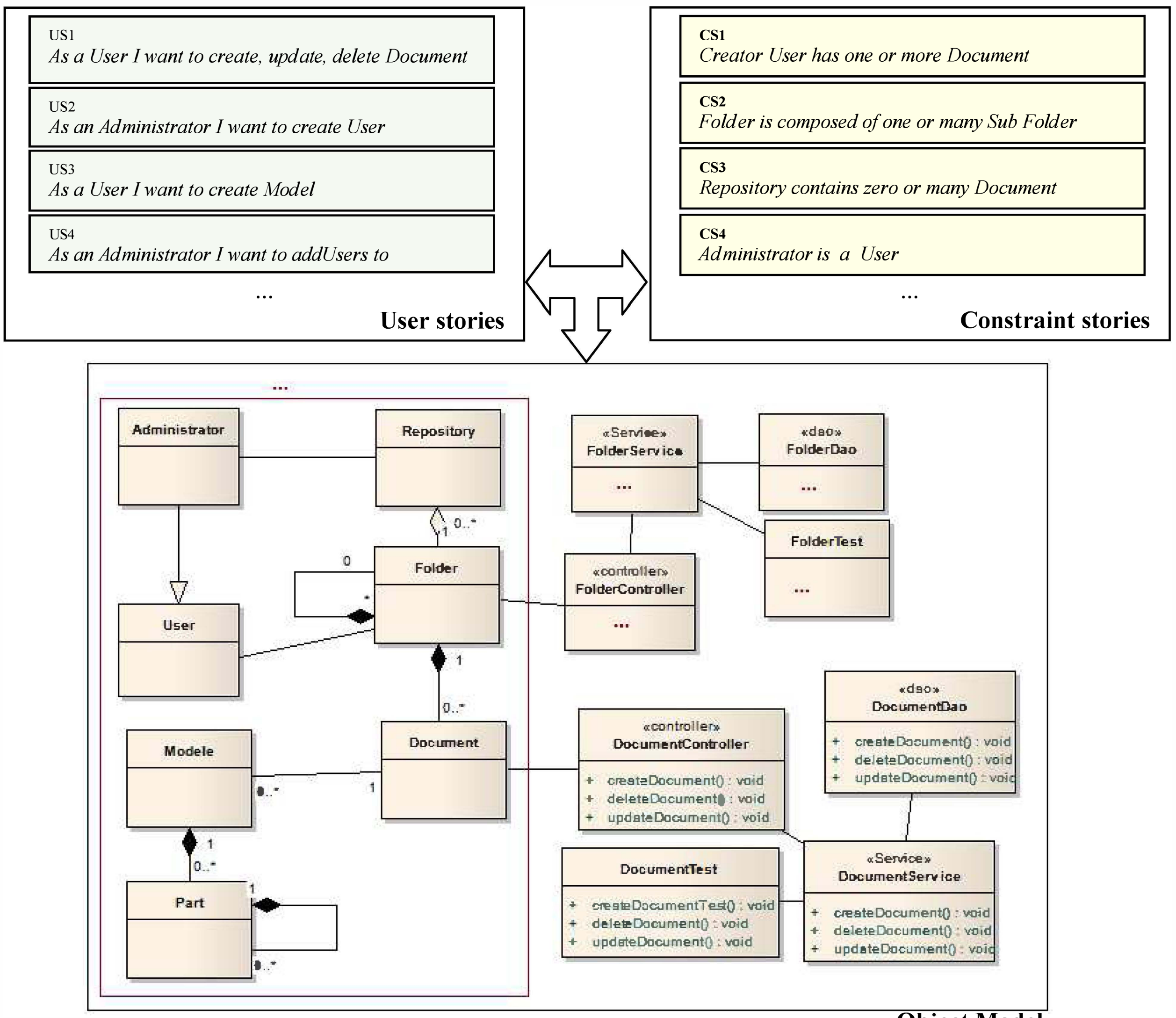
Folder is composed of one or many Sub Folder

Repository contains zero or many Document

Administrator is a User

As illustrated in the generated class diagram fig. 5, we can see that mapping is reflecting what is indicated in the story cards (User & Constraint story). Therefore, if the customer validates these cards, we can say that the future system will be reflected as desired.

To conclude our example, we can say that our approach was successfully applied on the presented requirements and was able to map them in a straight forward manner to a class diagram. The extracted objects from the User Story Template has been matched with those extracted from Constraint Story Card to handle all kinds of associations which are presented in this paper. For instance, we have simple association between objects (e.g. Document and User), self-association (e.g. Folder), inheritance (e.g. User and Administrator), aggregation (e.g. Repository and Document) and Composition (e.g. Folder and Document).


US1
As a User I want to create, update, delete Document

US2
As an Administrator I want to create User

US3
As a User I want to create Model

US4
As an Administrator I want to addUsers to

...

User stories

CS1
Creator User has one or more Document

CS2
Folder is composed of one or many Sub Folder

CS3
Repository contains zero or many Document

CS4
Administrator is a User

...

Constraint stories

...
Administrator
Repository
0..*
1
0
Folder
*
User
1
0..*
Modele
Document
0..*
1
1
0..*
Part
1
0..*
«Service»
FolderService
...
«dao»
FolderDao
...
FolderTest
...
«controller»
FolderController
...
«controller»
DocumentController
+ createDocument() : void
+ deleteDocument() : void
+ updateDocument() : void
«dao»
DocumentDao
+ createDocument() : void
+ deleteDocument() : void
+ updateDocument() : void
DocumentTest
+ createDocumentTest() : void
+ deleteDocument() : void
+ updateDocument() : void
«Service»
DocumentService
+ createDocument() : void
+ deleteDocument() : void
+ updateDocument() : void
Object Model

Fig.7 . Generated class diagram from Users Stories & Constraint Stories Cards

Other artifacts (DAO, Controller...) generated from user story template were omitted for a clarity purpose.

VI. Conclusion

The user story template suggested in the OOADA-RE approach and the four templates presented previously in this paper - Simple association template; Inheritance template; Aggregation template; and Composition template - are relatively easy to write and require few instructional coaching to think object in the RE phase. Those cards in our vision are written by Business Analyst, and we claim that they are easy to write even by customers. However, agreeing with the importance of oral language so as to express requirements in an object manner is different from having the expertise and skills to effectively handle requirements. Additional researches need to be conducted in order to determine how easy it is for a non-software person to be coached in object-oriented thinking and how well his output story will be in approving or disapproving our claim.

In this paper we have presented the importance of having a shared understanding between stakeholders, specially between customer and developers, and the importance of having a technique to map requirements straight to other phases without affecting the facility of communication offered by standard specifications techniques (e.g. User Story). We reminded the intent of the OOADA-RE approach which offers an intermediate level for requirements representation in the form of story cards thing that allow us to automate class diagram generation. We discussed also the new story card called CSC and the four templates aimed to handle associations. Finally, we presented an example illustrating the whole technique of the OOADA-RE approach, including our enhancement, to map requirements with class diagram. As future work we aim to evaluate our approach by a real case study. We are working also on an MDA [13] supporting tool for the OOADA-RE approach, and intend to apply the approach to Agent-Oriented Paradigm [10].

References

[1] A. Zeaaraoui, Z. Bougroun, M.G Belkasmi, T. Bouchentouf, “Object-oriented Analysis and Design Approach for the Requirements Engineering”, Journal of Electronic Systems, Vol. 2, No. 4, pp. 147-153, December 2012.

[2] Zeaaraoui, A.; Bougroun, Z.; Belkasmi, M.G.; Bouchentouf, T., "User story template for object-oriented applications," *Innovative Computing Technology (INTECH), 2013 Third International Conference on*, pp.407,410, 29-31 Aug. 2013

[3] S. M. Easterbrook, "What is Requirements Engineering?" (Draft book chapter) , 2004

[4] Bashar Nuseibeh , Steve Easterbrook, Requirements engineering: a roadmap, Proceedings of the Conference on The Future of Software Engineering, p.35-46, June 04-11, 2000, Limerick, Ireland

[5] OMG. 2006. Object Constraint Language (OCL), OMG Standard, v. 2.0.

[6] OMG. 2007. Unified Modeling Language (UML), OMG Standard, v. 2.3.

[7] Bajwa I., Behzad B., Lee M., OCL Constraints Generation from Natural Language Specification. EDOC 2010 - 14th IEEE EDOC Conference, Vitoria, Brazil, pp. 204-213. (2010)

[8] Grady Booch , James Rumbaugh , Ivar Jacobson: "Unified Modeling Language User Guide", The (2nd Edition) (Addison-Wesley Object Technology Series), Addison-Wesley Professional, 2005

[9] OMG. "Unified Modeling Language Specification version 2.4.1", August 2011. OMG document available from www.omg.org

[10] Gaur, V. Soni, A. Bedi, P., "An agent-oriented approach to requirements engineering", Proc. of IEEE 2nd International Conference on Advance Computing, 449-454, 2010.

[11] Cohn, Mike, “User Story Applied: For Agile Software Development”, Boston, MA: Addison-Wesley, 2004.

[12] Raj A., Prabharkar T., Hendryx S.: "Transformation of SBVR Business Design to UML Models". In ACM Conference on India software engineering, pp.29--38 (2008)

[13] Kleppe, A., Warmer, J., Bast, W.: "MDA Explained: The Model Driven Architecture. Practice and Promise". The Addison-Wesley Object Technology Se-ries. Addison-Wesley (2003)

[14] OMG: "Semantics of Business vocabulary and Rules (SBVR)", OMG Standard, v. 1.0, (2008)

[15] Silvie S., Keri A. :"SBVR's Approach to Controlled Natural Language, Workshop on Controlled Natural Language", 8-10 June, 2009, Marettimo Island, Italy (2009)

[16] Warmer Jos, Kleppe A.: "The Object Constraint Language - Getting Your Models Ready for MDA". Second Edition, Addison Wesley (2003)

[17] Zhou N. et Zhou X. (2004). Automatic Acquisition of Linguistic Patterns for Conceptual Modelin. *INFO 629 : Artificial Intelligence.*

[18] Ilieva M., Olga O.: "Automatic Transition of Natural Language Software requirements Specification into Formal Presentation". Springer LNCS Vol. 3513, pp.392--397 (2005)

[19] Bryant B., et al.: "From Natural Language Requirements to Executable Models of Software Components". In Workshop on S. E. for Embedded Systems pp.51-58 (2008)

[20] Mich L. (1996). NL-OOPS: from natural language to object oriented requirements using the natural language processing system LOLITA *.Natural Language Engineering, Vol. 2, No. 2, p. 161-187.*

[21] Highsmith, J. (2002); Agile Software Development Ecosystems; Publisher: Addison Wesley; Pub Date: May 26, 2002; ISBN: 0-201-76043-6; Pages: 448

[22] K. Li, R.G.Dewar, R.J.Pooley, [2003] “Object-Oriented Analysis Using Natural Language Processing”,

[23] Mala, G. A.,Uma, G. V. (2006). Automatic construction of object oriented design models [UML diagrams] from natural language requirements specification. PRICAI 2006: Trends in Artificial Intelligence, p.1155-1159.

[24] Deeptimahanti, D.K., Sanyal, R. (2011): Semi-automatic Generation of UML Models from Natural Language Requirements. Proceeding ISEC '11 Proceedings of the 4th India Software Engineering Conference, p. 165-174.

[25] HC, C.T. Have, K. Tveitane. "From use cases to UML class diagrams using logic grammars and constraints." RANLP 20