From User Stories to UML Diagrams Driven by Ontological and Production Model

Samia Nasiri[1], Yassine Rhazali[2], Mohammed Lahmer[3], Amina Adadi[4]
LMMI Laboratory of ENSAM, Moulay Ismail University
ISIC Research Team of ESTM
Meknes, Morocco

***Abstract*—The User Story format has become the most popular way of expressing requirements in Agile methods. However, a requirement does not state how a solution will be physically achieved. The purpose of this paper is to present a new approach that automatically transforms user stories into UML diagrams. Our approach aims to automatically generate UML diagrams, namely class, use cases, and package diagrams. User stories are written in natural language (English), so the use of a natural language processing tool was necessary for their processing. In our case, we have used Stanford core NLP. The automation approach consists of the combination of rules formulated as a predicate and an ontological model. Prolog rules are used to extract relationships between classes and eliminate those that are at risk of error. To extract the design elements, the prolog rules used dependencies offered by Stanford core NLP. An ontology representing the components of the user stories was created to identify equivalent relationships and inclusion use cases. The tool developed was implemented in the Python programming language and has been validated by several case studies.**

Keywords*—Ontology; prolog rules; natural language processing; UML diagrams; user stories

I. Introduction

Requirements engineering (RE) represents an important role in all types of software development processes. They aim to define the scope of development together with customers [1]. In agile software development, requirements are presented in documents named user stories. These documents are an efficient way to express requirements from the user. User stories are written in natural language that renders them easily understandable to stakeholders, indeed they are short text that depicts a semi-structured specification. A user story often uses the following format type: As <role>, I want <feature> to <reason> [2, 3].

Recently, agile software development has become more and more widely used. However, unlike the extensive automation research on RE in traditional software development, the automation of RE in agile development has not yet been investigated sufficiently, especially in the area of requirements modeling [4]. Requirements modeling is a critical process in the software engineering life cycle. It is a multi-faceted and time-consuming process. However, it provides a solid guide for the final product. The success of software projects depends mainly on careful and timely analysis and modeling of system requirements. In [5], the authors propose an approach to generate a conceptual model using heuristic rules and the NLP tool, but this model is not complete as it lacks the attributes of each entity. In [6], the authors also analyzed user stories in order to generate a UML use case diagram, but their approach is limited as they did not extract the relationships between the use cases. Furthermore, the authors of both approaches have not refined the relationships in the conceptual model or in the use cases in the use case diagram.

Our contribution aims to automate RE in agile development in order to generate automatically three UML diagrams which are class diagram, use case diagram, and package diagram with the refinement of results. To achieve the refinement task, at first, ontology engineering is created for defining synonyms and relationships between actions, given that action is a relationship or part of the use case, secondly, Prolog rules are used to eliminate the relationships that are at risk of error in the class diagram. Our purpose is to minimize the errors of the relationships extraction and to avoid the redundancy of the associations not taken into account by Wordnet in the class diagram. Prolog rules are applied at first to determine the relationship between engineering requirements. All statements are converted to rules and facts in the SWI-Prolog language. A user story is made up of three elements: the role which represents the actor who acts, then the action represented by a verb, and finally the object which has undergone the action. Ontology is created to describe the components of user stories as the role, the action and the object. This ontology represents the field of agile methods by focusing on the part of user stories.

After the creation of the classes in ontology editor, we proceeded to the stage of filling the ontology by enriching it with vocabulary and equivalent of class instances; we concentrated on the class "action" which represents the relation between the classes in a diagram of UML. The object properties reflect the relations that can be established between instances of the ontology classes.

The structure of this paper is as follows: This section introduces agile methods and our proposed approach. In Section 2, we present the related work of our proposal. However, we detail our proposal approach, and we present the main of the platform in Section 3. Then, we present a generated UML diagrams in Section 4. In Section 5, we present the discussion and analysis. At finally, conclusion is presented in Section 6.

II. Related Work

Several research projects have been carried out to automate the requirement engineering, but few researchers have developed a tool to automate the agile requirement presented in user stories. Since the agile method is the most used in software engineering, it was necessary to think about developing a solution to automate the design phase in the software development Life cycle. A user story is a very effective means of communication between future users of the software and developers and designers.

Our approach automates the design phase, i.e. from several user stories; our tool generates several diagrams as output: the class diagram, package diagram, and use cases. The tool is carried out by developing prolog language rules allowing the extraction of design elements such as actors and associations between classes, and subsequently refining the associations obtained by using ontology. The created ontology represents the user stories and based on looking for the synonyms of the associations that are found in the ontology. The use of ontology was primordial, firstly, in order to avoid the redundancy of the associations in a generated class diagram, secondly, in order to detect the inclusion between use cases in the use case diagram.

To analyze requirement engineering most of the researchers used the ontology domain to achieve their goal. Our approach combines prolog rules and the domain of ontology. The majority of the researchers have tended to analyze the requirement engineering [1] but in our approach, we start with the extraction of the design elements constituting the UML diagrams, and then we analyze the requirements using the ontology that represents the strong point of our approach. In [7], the authors propose the business process ontology design scheme. The built ontology is considered as a knowledge base by collecting the user stories to reuse them from previous projects. Classes are created in ontology according to Role-Action-Object relations. In [8], the authors used an NLP tool named “OpenNLP Parser” and Wordnet in order to analyze the requirements. Their aim is to extract concepts to constitute a class diagram. The authors developed a desktop tool named “RAPID”, the limitation of this tool is that each sentence in the requirements must match a specific structure. In [9], the authors develop a formal Web Ontology Language ontology for the standard representation of engineering requirements. The proposed ontology uses explicit semantics that makes the ontology amenable to automated reasoning. The approach allows the evaluation and classification of engineering requirements. In [10], the author’s Approach allows to Test Case Generation Based on Inference Rules. In [11], the authors are built an automated tool named “ABCD” for class diagram generation from user requirements. They used NLP techniques to extract class diagram concepts and generate an XMI file representing a class diagram. The limitation of their tool is that the system does not focus on the problem of concept redundancy. In [12], the authors have developed a framework to automate the documentation by elaborating the ontology. 60 percent is a percentage of their automation. In [5], a conceptual model is generated from a set of user stories, their tool named visual narrator, this tool does not extract attributes of entities in the conceptual model, and they focus on detecting entities and relationships. In [13], the conceptual model is generated from an unrestricted format such as general requirements, user stories, or use cases; but the attribute extraction rule is based on a set of previously designed verbs. In [14], the searchers suggest an approach that generates a class diagram from use case specifications, parts of speech tags (POS tags), and typed dependencies (TD), were used to reach their objective, however, the developed tool analyses simple sentences. The rules used to extract attributes are not valid in most sentence structures, due to the failure of consecutive names processing. In [6], the authors used the NLP tool named TreeTagger analyzer and developed a JAVA plugin to generate the use case diagram from the user stories; their tool does not handle sentences containing compound nouns. Also; it does not support inclusion or exclusion relationships between use cases. In [15], the authors analyze the requirements by combining the ontological model with prolog rules. This analysis relies on tracing the requirements, eliminating duplicate requirements, and identifying conflicting requirements. The authors used agile requirements. In [16], an NLP-based tool is implemented to generate an Entity Relationship diagram from requirement specification. The machine-learning module is implemented by using a supervised learning mechanism. In [17], through a linguistic analysis of sentence structures and action verbs in user stories, the authors discover patterns of labeling refinements. The refinement goal is a transformation of User Stories into Backlog Items. In [18] the authors propose a technique to automatically transform textual user stories into visual use case scenarios.

III. An Approach to Extract Design Elements and Analyse Relation between the Classes and use Cases

In our previous approach [19], our objective was to define the extraction rules of the object-oriented design elements, such as actors, classes, attributes, operations of classes, and associations. These components were essential to generate a class diagram, presented in an XMI file and also in a PNG image. We used a natural language processing (NLP) tool named "Stanford CoreNLP" and python language to achieve our goal. After extraction of associations, we have used Wordnet to delete the redundancy associations between the two classes. The use of Wordnet was not sufficient to avoid Redundancy that’s why we have thought of another approach that integrates artificial intelligence materialized firstly in the use of prolog language for the definition of production rules to generate associations. Secondly in the use of requirement ontology in which we have defined synonyms of verbs presenting associations. The ontology is created in Protégé editor.

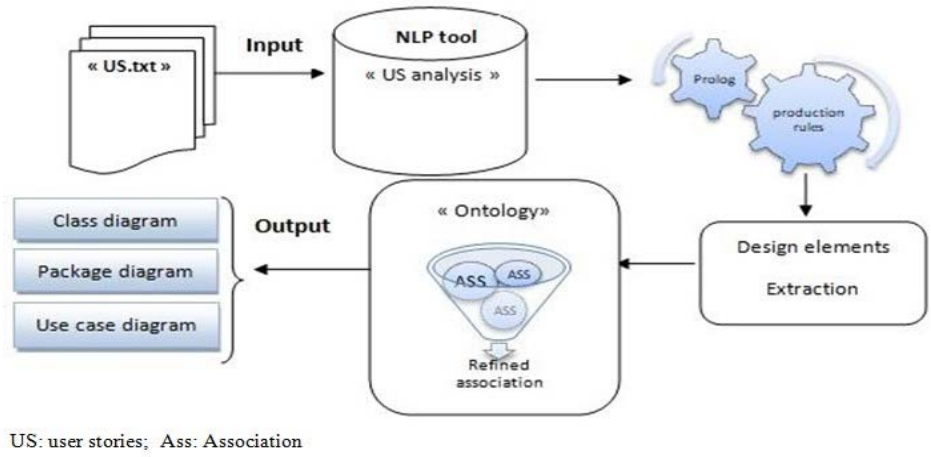


Fig. 1. Architecture of the Proposed Approach.

All treating was done in python language, even access to ontology to search for synonyms. Prolog language is used firstly to define the Production rules for extraction of the design elements. Secondly to define rules to detect errors in association extraction. The output of our framework is three diagrams: class diagram presented in XMI file, use case, and package diagrams presented in a PNG image. This image is carried out by using Plant UML.

The processing of a text in user stories goes through several steps: Splitting, Tokenizing, POS, Lemmatization, and typed dependencies. The user stories analysis was done using the NLP tool named Stanford core NLP. Fig. 1 shows the architecture of the proposed approach.

A. Prolong Rules for Extracting Relations

To extract the design elements which constitute the class diagram, from a set of user stories, we followed the steps described in the algorithm presented below:

Algorithm: Design elements extraction

1: Procedure (Stories S, Actors A, Classes C, Rels R, Attributes ATT, Operations Op)

2: for each s in S

3: p=POS(s)

4: Word_tokenize(s)

5: Dependency_parse(s)

6: Extract_Nouns(s)

7: Extract_Verbs(s)

8: A=Extract_Actor(s)

9: C=extract_class(s)

10: R= Extract_all_relationships(s) [prolog rules: Association(X,Y,Z); X is association name, Y and Z are classes]

11: Comp= filtrate_composite_rel (R)

12: for each c in C

13: if c in comp then ATT=extract_attribute (Comp)

14: for each r in R

15: If ATT in r

16: Op=r

Based on rules previously defined in [19], we have defined a production rules written in prolog language to extract actors, classes, and associations which connect classes (lines 8-10). The facts are provided from NLP tool that provide the nouns, the verbs and typed dependencies (lines 3-7). To extract attributes of classes we have followed the same approach of [19] i.e. from the resulting classes we do a refinement; some classes become attributes and thereafter some associations become operations of a class (lines 10-16).

The rules are presented in this form: Association(X, Y, Z); The objective of these prolog rules is to extract X which represents the association name, and Y and Z which are classes in the UML class diagram.

B. Prolog Rules for Detecting Errors in Relation Extraction

After extraction of association between classes, we proceed to the refining step by applying some prolog rules which detect errors in the list of association.

Rule1: if two or more actors have the same action to execute.

Rule2: if there is an association between A class and B class and the same association between B class and C class then there isn't an association between A class and C class. This rule avoids transitivity between associations which can clutter the class diagram with several unsuccessful relations.

C. Ontology for Analysis of Relations between the Classes in Class Diagram and use Cases in use case Diagram

Our ontology is important to represent knowledge of the application domain and to identify the relations between requirements such as composition or synonyms. USOn is an ontology that describes an agile requirement; we have created ontology classes and instances through Protégé ontology editor.

Fig. 2 shows the hierarchy of the ontology USon. Table I presents the description of some classes.

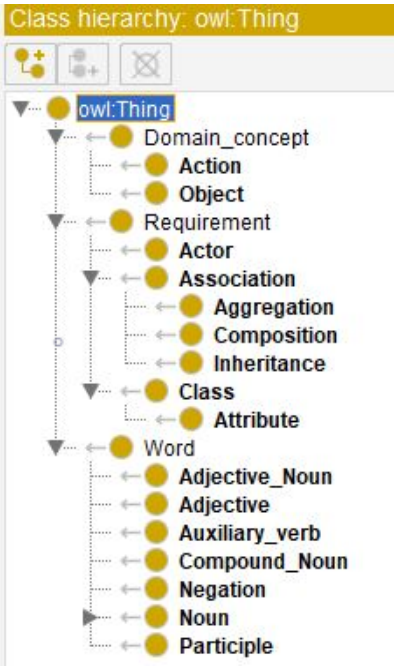


Fig. 2. Hierarchy of the Ontology USon.

TABLE I. DESCRIPTION OF SOME CLASSES

Class name	Description
Action	class whose elements are verbs which represent the associations in class diagram
Object	class whose elements are nouns which represent the classes in class diagram
Actor	Class whose elements are nouns which represent the role in user story (As role,…). The actor is who perform the action. Actors are present in the use case diagram.
Association	Symantec link between classes in the class diagram
Class	class whose elements are nouns which represent the classes in class diagram
Attribute	class whose elements are nouns which represent the attributes of classes in class diagram
Word	class whose elements are tokens which represent part of user story

We have defined a set of Synonyms to *Action* class in order to refine association name. The same process is for the *Actor* class. Fig. 3 shows an example of defined synonyms and inclusion action.

Our tool accesses the *USon* ontology in order to compare the names of the associations obtained using the prolog rules with those defined as synonym of the *Action* class. Subsequently, the associations will be refined. The refinement of actors in use case diagram is done by browsing synonyms of Actor instances.

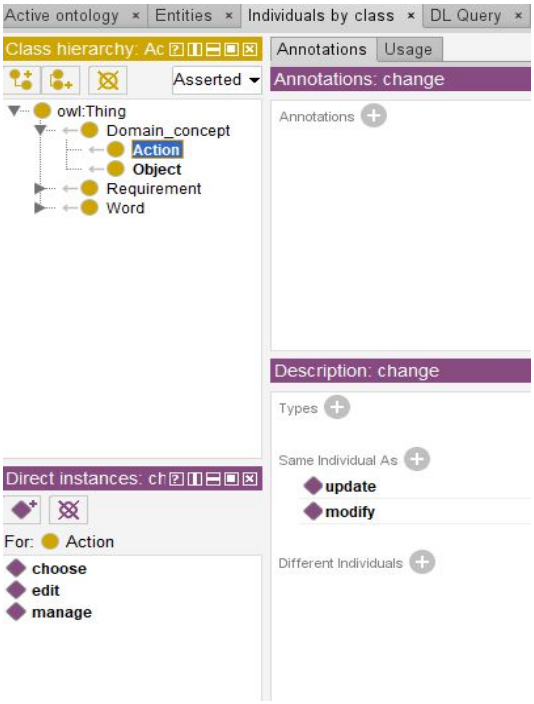


Fig. 3. Synonyms of Change Action.

Consider an example of user stories:

US1: As a manager, I want to manage account of users.

US2: As a manager, I can create a new account.

US3: As a user, I can modify login account.

US4: As a user, I can update my login.

In US1 the action is the verb “manage”, according to our approach the following association is extracted: Manage (manager, account).

In US2 the action is the verb “create”, according to our approach the following association is extracted: Create (manager, account).

Our tool, at first looks for the relationships between the same classes as in US1 and US2, and US3 and US4, after Wordnet is used in order to avoid redundancies, then browsing of USon ontology is mandatory to detect synonym and inclusion relations between use cases; in the example, the actions modify and update are synonyms as shown in Fig. 3 so an association will be removed from the list of associations in order to avoid duplicate associations.

In USon Create is part of Manage as shown in Fig. 4, then there is an inclusion between two use cases extracted from US1 and US2: “manage account” and “create account”.

Consider these user stories:

As a user, I can change the account information.

As a user, I am able to edit account information.

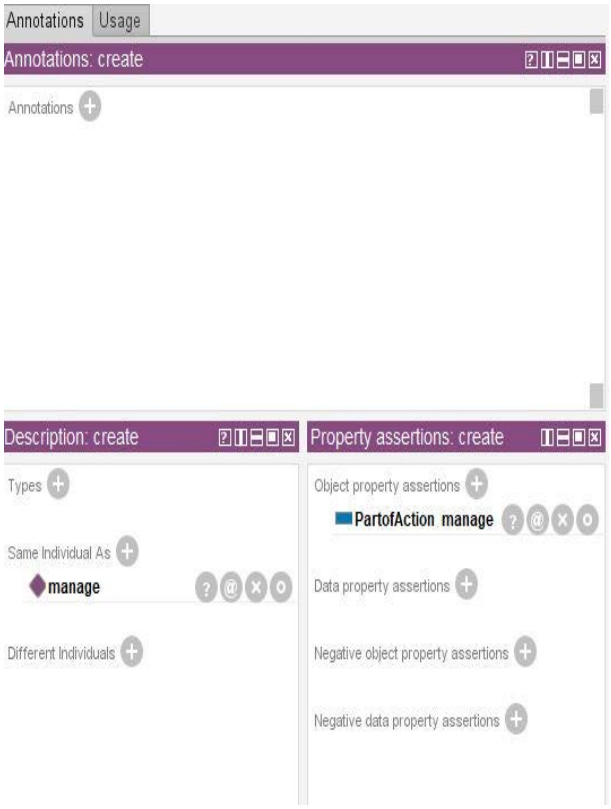


Fig. 4. Composition of Manage Action.

Our tool with the help of the Stanford NLP tool and prolog rules allows extracting two use cases: “change account information” and “edit account information”.

According to the ontology USon, the update action is part of Edit. In Fig. 3 the update action is a synonym of change action then change is part of the Edit action.

We can deduce from this combination between ontology and prolog rules that there is an inclusion between two use cases: “change account information” and “edit account information”.

IV. Generation of UML Diagrams

A. Generated Class Diagram

After extracting the design elements, the next task was to regroup these elements to constitute the class diagram. The developed tool generates an XMI file which is an Ecore file. Ecore file is the Eclipse Modeling Framework (EMF) meta-model, which illustrates the names of the classes, their attributes, and their types, as well as the methods and relationships with their classifications. Also, PlantUML API is used to visualize the class diagram. These treatings were done in python language. To implement our new approach, we used the same case studies[1,2] from article [19] and compared the results. We found that in our old approach, the class "people" (case study[1] number 2) was detected, yet in our approach; there is an association of inheritance between "people" and all the actors thanks to our ontology. This association has been added to the generated class diagram.

Regarding the first case study[2], there is redundancy in the operations obtained (filtrate (type) and choose (type)) which are at the origin of associations before refinement. Wordnet could not detect that these verbs are equivalent, so we used the ontology.

We noted that the associations obtained from the old approach are all obtained using the extended rules of our second approach.

B. Generated Package and use Case Diagrams

A package diagram offers many advantages to designers who want to create a graphical representation of their UML system or project. This diagram simplifies the complex class diagram into a tidy visual form. In our case, we used the package diagram to organize the class diagram.

After generating the class diagram, the next task was to generate the package and use case diagrams. To do this, we based on the design elements already extracted such as: classes, associations, and actors.

To extract a package, we first use the associations that link the actor and another class, and secondly, we add the term "manage" before each class to form a package. To detect the dependency between the packages, we take into consideration the relationship between the classes that make up the packages. The use cases are formed from the associations between classes provided that one of the classes is an actor. PlantUML API is used to visualize the package and use case diagrams. All treatings were done in python. Fig. 5 shows the generated package diagram of the case study[1] which represents inline course management: videos, quizzes, and others.

The generated package diagram is based on associations and classes of the class diagram. The red arrows between the packages represent the dependencies between them. Table II represents some use case diagrams for each package presented in Fig. 5.

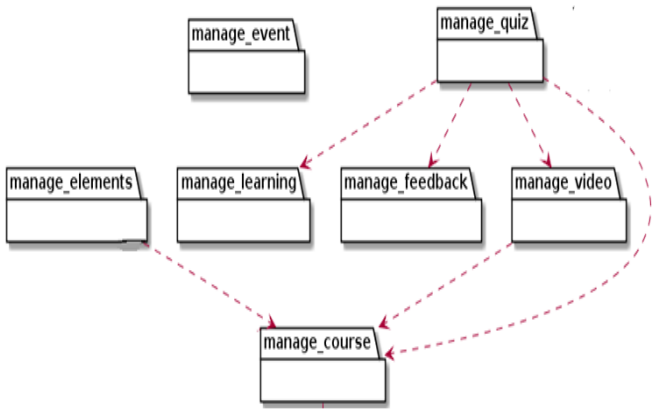
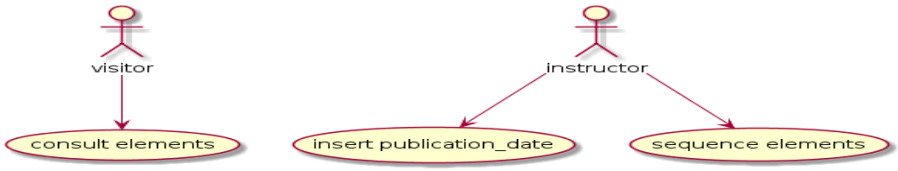
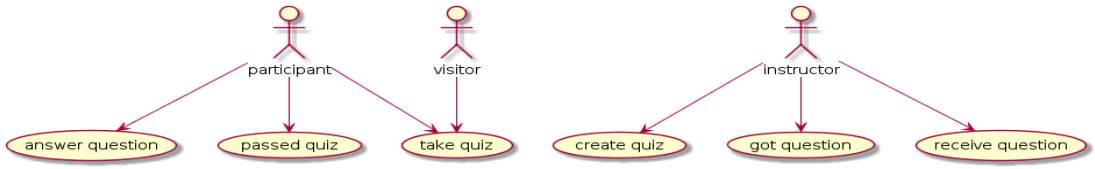
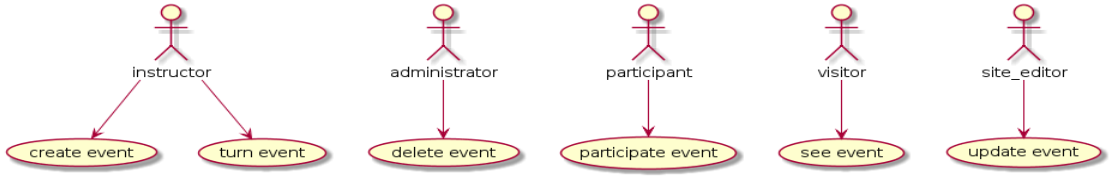
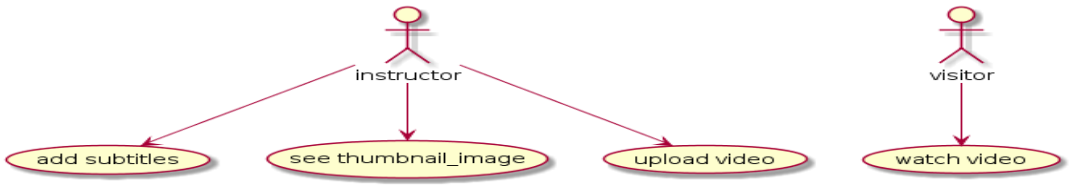


Fig. 5. The Generated Package Diagram.

[1] https://drive.google.com/file/d/1wk8yCa9wS12ooeEwR0UjWDsK0_b4kaKG / view?usp=sharing

[2] https://github.com/MarcelRobeer/StoryMiner/blob/master/example_stories.txt

TABLE II. USE CASE DIAGRAMS OF PACKAGES

Package name	Use case diagrams
Manage Course	
Manage Element	
Manage Quiz	
Manage Event	
Manage Video	

By comparing the results obtained manually with those which are automatic, we noted that our approach extracts 99% of the relationships. Generating the package and use case diagrams based on the associations of the class diagram has revealed its effectiveness.

Regarding the class diagram, the USon ontology allows firstly detecting inheritance association between actors such as actor named People and other actors, secondly determining the synonyms of associations.

Regarding the use case diagram, the USon ontology allows firstly determining the synonyms of use cases, secondly the inclusion relationships between two use cases.

Our approach remains very effective thanks to the strong point of the combination of the domain of ontology and prolog rules. We can say that the ontology we created complements the prolog rules in order to obtain better results.

Table III shows a comparison between my old approach [19] and my proposal. However, Table IV depicts a comparison between related work and my proposed.

Our approach is the unique method that defined extraction rules for associations of class diagram using prolog language. Subsequently, the association and use cases are analyzed and refined using our built ontology named “USon”. The associations are the key for building the UML diagrams: use case and package diagram.

TABLE III. COMPARISON BETWEEN MY OLD APPROACH AND MY PROPOSAL

	Association		**Composition relationship**	**Inheritance relationship**	**Synonyms**	**Output**
Old approach [19]	NLP tool: Stanford coreNLP	Python heuristic rules	Python heuristic Rule s: H1, H2, H3	Python heuristic H4 and H5	Wordnet: Synonym of associations (verb which link two classes)	**Class diagram**: XMI file
Proposal approach		- python language - Prolog rules	- Prolog rules - Ontology USon	Ontology USon	- Wordnet - Ontology USon: 1. synonym of use cases (action in use case) 2. synonym of associations	- **Class diagram**: XMI file and PNG image (using plant UML) - **Package diagram**: PNG image(using plant UML) - **Use case diagram**: PNG image(using plant UML)

TABLE IV. COMPARISON BETWEEN SOME RELATED WORK AND MY PROPOSED

Related work	**Input**	**Output**	**Approach and tool**
[7]	User stories	Business process ontology	Reuse of user stories
[8]	Business document	Concept-classes	- Wordnet - Linguistic rules
[9]	Engineering requirements	Formal Web Ontology	Evaluation and classification of Engineering requirements
[5]	User stories	Conceptual model without attributes	Visual narrator tool
[14]	Text requirement and user stories	Conceptual model	Visual C # language and Stanford CoreNLP
[15]	User stories	- Elimination of duplicate requirements, and identification conflicting requirements - not any generation of the UML diagram or conceptual model	- Prolog and python language - Ontology
[19]	User stories	Class diagram	Python language and Stanford CoreNLP
My proposal	User stories	- Class diagram - Package diagram - Use case diagram	- Standford coreNLP - Python and prolog language - Ontology

V. CONCLUSION

This paper have proposed an approach to automate the analysis phase in an agile context, to extract the design elements which are essential to constitute the generated UML diagrams: the class diagram, the package and use case diagrams.

Our approach is based on the combination of prolog rules and an ontology which present the user stories. The prolog rules used dependencies offered by Stanford core NLP. This combination is the strong point of our approach. The main advantages of the proposed technique are:

- Improvement of the results obtained from our previous approach by Applying artificial intelligence presented in prolog rules and ontology.
- Generation of three UML diagrams which facilitate the design of analytical tasks in the team.
- Refined classes have been obtained following a transformation of some classes into attributes using composition relationships, and some relationships to operations.
- Definition of prolog rules for detecting errors in relation extraction.

Our proposed approach is very useful to ease the analytical tasks in the design team. Next, minimize time and costs. The benefits of our approach are the utilization of agile requirement to automate them, these requirement named user stories are the best way to describe the engineering requirement. In the future, our work will be completed by generating user interfaces and code of the software.

REFERENCES

[1] D. Pandey and V. Pandey, “Requirement Engineering: An Approach to Quality Software Development,” Journal of G--lobal Research in Computer Science, vol. 3, no. 9, pp. 31-33, 2012.

[2] M. Cohn, User Stories Applied: for Agile Software Development. Redwood City, CA, USA: Addison-Wesley Professional, 2004.

[3] Y. Wautelet, S. Heng, M. Kolp, and I. Mirbel, “Unifying and extending user story models,” Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 8484 LNCS, pp. 211–225, 2014, doi: 10.1007/978-3-319-07881-6_15.

[4] I. K. Raharjana, D. Siahaan, and C. Fatichah, “User Stories and Natural Language Processing: A Systematic Literature Review,” IEEE Access, vol. 9, pp. 53811–53826, 2021, doi: 10.1109/ACCESS.2021.3070606.

[5] G. Lucassen, M. Robeer, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper, “Extracting conceptual models from user stories with Visual Narrator,” Requir. Eng., 2017, doi: 10.1007/s00766-017-0270-1.

[6] M. Elallaoui, K. Nafil, and R. Touahni, “Automatic Transformation of User Stories into UML Use Case Diagrams using NLP Techniques,” in Procedia Computer Science, 2018, vol. 130, pp. 42–49, doi: 10.1016/j.procs.2018.04.010.

[7] C. Thamrongchote and W. Vatanawood, “Business process ontology for defining user story,” 2016 IEEE/ACIS 15th Int. Conf. Comput. Inf. Sci. ICIS 2016 - Proc., pp. 1–4, 2016, doi: 10.1109/ICIS.2016.7550829.

[8] P. More and R. Phalnikar, “Generating UML Diagrams from Natural Language Specifications,” Int. J. Appl. Inf. Syst., vol. 1, no. 8, pp. 19–23, 2012, doi: 10.5120/ijais12-450222.

[9] A. Mukhopadhyay and F. Ameri, “An ontological approach to engineering requirement representation and analysis,” Artif. Intell. Eng. Des. Anal. Manuf. AIEDAM, vol. 30, no. 4, pp. 337–352, 2016, doi: 10.1017/S0890060416000330.

[10] H. Tan, M. Ismail, V. Tarasov, A. Adlemo, and M. Johansson, “Development and Evaluation of a Software Requirements Ontology,” SKY 2016 - 7th Int. Work. Softw. Knowledge, Proc. - conjuction with IC3K 2016, pp. 11–18, 2016, doi: 10.5220/0006079300110018.

[11] W.B. A. Karaa, Z. B. Azzouz, A. Singh, N. Dey, A. S. Ashour, H. B. Ghazala, “Automatic Builder of Class Diagram (ABCD): an Application of UML Generation From Functional Requirements,” Journal of Software Practice and Experience, vol. 46, no.12, pp. 1443–1458. 2016, doi: 10.1002/spe.2384.

[12] M. P. S. Bhatia, A. Kumar, and R. Beniwal, “Ontology driven software development for automated documentation,” Webology, vol. 15, no. 2, pp. 86–112, 2018.

[13] M. Javed and Y. Lin, “Iterative process for generating ER diagram from unrestricted requirements,” ENASE 2018 - Proc. 13th Int. Conf. Eval. Nov. Approaches to Softw. Eng., vol. 2018-March, no. Enase, pp. 192–204, 2018, doi: 10.5220/0006778701920204.

[14] J. S. Thakur and A. Gupta, “Automatic generation of analysis class diagrams from use case specifications,” 2017.

[15] M. S. Murtazina and T. V. Avdeenko, “Requirements analysis driven by ontological and production models,” CEUR Workshop Proc., vol. 2500, pp. 1–10, 2019.

[16] P. G. T. H. Kashmira and S. Sumathipala, “Generating Entity Relationship Diagram from Requirement Specification based on NLP,” 2018 3rd Int. Conf. Inf. Technol. Res. ICITR 2018, pp. 1–4, 2018, doi: 10.1109/ICITR.2018.8736146.

[17] L. Müter, T. Deoskar, M. Mathijssen, S. Brinkkemper, and F. Dalpiaz,``Re_nement of user stories into backlog items: Linguistic structure and action verbs," in Requirements Engineering: Foundation for Software Quality (Lecture Notes in Computer Science), vol. 11412. New York, NY, USA: Springer, 2019, pp. 109_116.

[18] F. Gilson, M. Galster, and F. Georis, ‘‘Generating use case scenarios from user stories,’’ in Proc. Int. Conf. Softw. Syst. Processes, Jun. 2020, pp. 31–40, doi: 10.1145/3379177.3388895.

[19] S. Nasiri, Y. Rhazali, M. Lahmer, and N. Chenfour, “Towards a Generation of Class Diagram from User Stories in Agile Methods,” Procedia Comput. Sci., vol. 170, pp. 831–837, 2020, doi: 10.1016/j.procs.2020.03.148.