ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/324735825

Automatic Transformation of User Stories into UML Use Case Diagrams using NLP Techniques

Article *in* Procedia Computer Science · January 2018

DOI: 10.1016/j.procs.2018.04.010

CITATIONS
17

READS
238

3 authors:

Elallaoui Meryem
Université Ibn Tofail
6 PUBLICATIONS **43** CITATIONS
SEE PROFILE

Khalid Nafil
Mohammed V University in Rabat, Morocco
23 PUBLICATIONS **71** CITATIONS
SEE PROFILE

Raja Touahni
Université Ibn Tofail
90 PUBLICATIONS **329** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Protein identification accuracy improvement View project

medical image processing and analysis View project

All content following this page was uploaded by Khalid Nafil on 07 February 2019.

The user has requested enhancement of the downloaded file.

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 130 (2018) 42–49

Procedia Computer Science

www.elsevier.com/locate/procedia

The 8th International Conference on Ambient Systems, Networks and Technologies (ANT 2018)

Automatic Transformation of User Stories into UML Use Case Diagrams using NLP Techniques

Meryem Elallaoui[a,*], Khalid Nafil[b], Raja Touahni[a]

[a]*Department of Physics, Faculty of Sciences, Ibn Tofail University, Kenitra 14000, Morocco*
[b]*Software Project Management Research Team, ENSIAS, University Mohammed V, Rabat, 1000, Morocco*

Abstract

Agile methods in general and the Scrum method in particular are gaining more and more trust from the software developer community. When it comes to writing a functional requirement, user stories become more and more usable by the community. Furthermore, a considerable effort has already been made by the community in relation to the use of the use case tool when drafting requirements and in terms of model transformation. We have reached a certain stage of maturity at this level. The idea of our paper is to profit from these richness and to invest it in the drafting of user stories. In this paper, we propose a process of transforming user stories into use cases and we will be able to benefit from all the work done in the transformation of the models according to the MDA approach. To do this, we used natural language processing (NLP) techniques, by applying TreeTagger parser. Our work was validated by a case study where we were able to obtain very positive precisions between 87% and 98%.

© 2018 The Authors. Published by Elsevier B.V.
Peer-review under responsibility of the Conference Program Chairs.

Keywords: User Stories; UML; Use Case; NLP; MDA

1. Introduction

Scrum is an agile methodology[1], which is based on dividing projects in iterations (sprints). Each sprint includes five phases, which are: initialize, analyze, design, realize and test a set of user stories. User stories are a short notation used to describe agile requirements[2]. Indeed, many software development companies have adopted this notation in agile development process[3,4]. In the context of Model-Driven Architecture (MDA)[5], user stories are described in a Computational Independent Model (CIM) level. This model represents the highest level of abstraction and describes the requirements of the system in understandable way by domain experts. MDA promotes the transformation approach by transforming the (CIM) into the Platform Independent Model (PIM) and the (PIM) into the Platform Specific Model (PSM). Several research studies addressed the problem of transforming the (PIM) into (PSM) and the (PSM) into code[6]. The work done on the (CIM) level uses the use case diagrams, BPMN or activity diagrams[7,8,9,10]. In this paper we want to take advantage of this previous work and apply it in the agile context. Our proposal aims to

* Corresponding author. Tel.: +212-642-640-814.
E-mail address: elallaoui.meryem@univ-ibntofail.ac.ma

1877-0509 © 2018 The Authors. Published by Elsevier B.V.
Peer-review under responsibility of the Conference Program Chairs.
10.1016/j.procs.2018.04.010

bring a new solution related to the problem of constructing the (CIM) model by automatically transforming a set of user stories into UML use case diagrams. To address this challenge, and on the basis of natural language processing (NLP) techniques, we suggest the development of a new plugin for automatic transformation of user stories into UML use case diagram. Unified Modeling Language (UML) is generally used to model requirements in a more effective and organized manner. It is frequently the best modeling language used for designing all parts of the system[11]. It is also considered as a good way to reduce the ambiguity between the needs analysis and design phase, on the basis of diagrams such as class, sequence and use cases diagrams[12].
Natural language processing (NLP) is the automation processing of human natural language, which can be defined also as semi-automatic processing. NLP is mainly multidisciplinary and associated to linguistics[13]. Usually the requirements are expressed by sentences written in natural language, these sentences may be incomplete and inconsistent. The analysis of these requirements is carried out using natural language processing (NLP) tools, which allows linguistic analysis and provides automated assistance[14].
The advantage of our technique is to be able to benefit from both the power of expression of user stories as well as the transformation work already carried out in the MDA approach. The benefit in applying an MDA approach lies in its ability to facilitate the work of the development team and Product Owner. In general, this technique can be applied in a software development process, and in an MDA context in particular. Very positive results have already been achieved for ”actors”, which represents 98 percent for accuracy and recall. For use cases and their relationships, we have acquired 87 percent for accuracy and 85 percent for recall. The structure of this paper is as follows. This section introduces UML and natural language processing (NLP). Section 2 concludes and compares the research work in the area of textual requirements to UML. In Section 3 we present the architecture of the proposed approach that serves as the baseline for our work, we detail the steps followed in the transformation of user stories into a UML use case diagrams, and we present the main algorithm of the plugin. Section 4 discusses our approach and report on the case study evaluation. Section 5 draws some conclusion and presents future lines of work.

2. Related Work

In this section, we review some works in the area of textual requirements transformation into UML models generally, and model transformation from (CIM) to (PIM) particularly.

2.1. Textual Requirements Transformation

In recent years, several approaches have been studied for the natural language requirements transformation into UML diagrams, but few researchers have specifically focused on agile requirements and in particular user stories. Kumar[15,16], introduces a new technique for generating static and dynamic UML models from natural language requirements (UMGAR). This technique is semi-automatic. In another previous work, they have presented a tool called (SUGAR), which consist in generating the use case and class diagrams from natural language requirements. SUGAR focus on generating only static UML models from requirements, following the Rational Unified Process (RUP)[17]. More and Phalnikar[18] have used an algorithm and implemented a prototype tool called RAPID, in order to generate UML diagrams from NL specifications. Herchi[19] suggest an approach for extracting class diagrams from textual requirements, their approach is based on natural language processing (NLP) technique. The textual data and user requirements represent the input, and the selected class names, their associations and attributes are structured in XML file. Zhou[20] introduces an approach for automatic generation of class diagram from NL requirement documents. To understand written requirements, the system uses NLP techniques and applies domain ontology to enhance the performance of class identification. They used a domain ontology to refine the result. The link grammar parser and part of speech (POS) tagger are used for extracting the candidate classes. Tool named LOLITA (Large-scale Object-based Language Interactor, translator and Analyser) for NLP was introduced by Mich[21]. LOLITA consists in pre-processing the user requirements, and including all the tasks for NL analysis. The limitation of the proposition is that the tool cannot differentiate between objects, classes and attributes, but only the extraction of objects from NL. Vrushali[22] have implemented a system called RACE Requirements Analysis and class diagram Extraction. Their technique consist in translating software requirements expressed in NL into class diagram. The architecture of RACE is quite similar to RAPID. In[23], the authors define a semi-automatic technique for generating analysis class models, design

class models, collaboration and use case diagrams from naturel language (NL) requirements. But the authors do not detail the transformations. The identification of irrelevant classes is done manually. Madanayake[24] carried out a survey, in order to define the most used modeling techniques for software engineering. The authors claim that Entity Relationship diagrams, class diagrams, use case diagrams and user stories are the most used. Also, they discussed the inter-compatibility between use cases and user stories. Robeer[25] propose a technique for generating conceptual models automatically from user stories. They have implemented Visual Narrator tool based on a selection of state-of-the-art NL processing heuristics. To calculate the precision and recall, the authors ignore attributes and cardinality, and focus on the details of user stories. Letsholo[26] introduce a new software tool named (TRAM) for the automatic construction of analysis models from the natural language specification of requirements (NLR). Analysis models are represented by UML class diagram. To improve the model quality, TRAM allows the exchange with the software analyst or domain expert. Also, the knowledge of the expert modeler is integrated to the model construction. Prasad[27] propose a technique to reduce complication of how to derive design model from analysis model. This transformation is based on interface representation, data structure, architecture of system and component level detail. Concepts and principles are applied during each activity, in order to produce high quality software. Sagar[28] propose the creation of conceptual modeling from functional requirements (FRs) based on linguistic aspects of the English language. For visualizing the functional requirements, the authors focus on the automatic extraction of concepts and their relationships. For identifying entities within a post-processing step, they have applied the optimization. Yue[29] present systematic review of the textual requirements transformation into analysis models. The authors aim to evaluate existing literature works and hence provide suggestions. Pre-processing methods such as categorization, semantic, lexical, pragmatic analyses and syntactic are used in isolation or in combination. The authors conducted a survey of 20 works that are classified into 16 transformation approaches. Karaa[30] define a new automated tool named Automatic builder of class diagram (ABCD), which consists in generating UML class diagrams from natural language NL (user requirements). They have used Stanford NLP toolkit for NL requirements processing in order to extract syntactical and lexical elements. The authors use a pattern-matching NLP technique to extract elements of the class diagram such as generalization, aggregation, association multiplicity and composition. A survey works related to the transformation of requirements into UML diagrams is given in[31]. The study covers works carried out manually from 1976, to automatic tools in 2015. The authors present a comparative study and show the weaknesses and strengths of each technique. Also, a discussion around the combination of requirements engineering and artificial intelligence has taken place. Osman[32] suggest a summary of existing tools in order to find out data and process models from text expressed in natural language. For each proposed tools, they have analyzed the related issues. The degree of automation, completeness and efficiency of the transformation are examined for data models extracted. They also present various case studies in archaeological and medical fields.

2.2. Model Transformation from (CIM) to (PIM)

Despite the popularity of MDA approach, the number of methods for generating UML diagrams from requirements in (CIM) level is few and far between.

Work has been proposed by[33] for modeling the (CIM) model from artifacts and concepts of RUP methodology. Their (CIM) model represents both the business process and requirements, and it contains three models: a business analyses, a business use cases and use cases model. Other work focuses on the derivation of the (PIM) model from the (CIM) model. The authors in[7] introduce a new technique for generating the (PIM) model from (CIM) model. The (CIM) model is represented by two activity diagrams in order to define system requirement and business processes. The activities of the organization are defined by business process model, and the system that supports these activities is represented by the requirements model. On the basis of this requirement model, a class diagram is obtained which defines the (PIM) model. In[6], author Proposes a technique for transforming the (PIM) model from the (CIM) model based on QVT transformation rules. Their (CIM) model is defined by use case diagrams for describing the system, and a business process model for representing the static and the behavioral views of the system. A new technique has been proposed by[34] to transform the (CIM) model into a (PIM) model using oriented functionalities and components method. The authors use an intermediate model in their method, this model does not take into account the business processes. In previous works[38] the authors proposed an algorithm that takes as input a set of user stories stored in a text file, then analyzed the first line of the text file and searched the actor of the first user story to insert it in the XML file generated. In the same way action and benefit were searched. The XML file generated is transformed into

a sequence diagram by using the UML 2 tool SDK plugin for Eclipse. The generated sequence diagrams are used as input for androMDA tool[39].

3. Transformation Process

Given that the user stories are expressed by sentences that represent the customer needs at a high level. And, in order to improve the understanding and collaboration between the business, the Product Owner, developers and testers, and to significantly reduce the time and efficiently implementing the system, we propose the automatic transformation of user stories into UML use case diagrams.

3.1. TreeTagger Processing and Plugin Use Case

The transformation of user stories into use case diagrams has the following main features: the first step consists of a preprocessing of text file containing a set of user stories. This is carried out using an algorithm that removes all unnecessary words. Then, the new file is parsed using TreeTagger parser which produces parse tree for each user story, through which noun (NN), proper noun (NP), determiner (DT) and verb (VV) can be selected. This parse tree facilitates the extraction of actors, use cases and their associations relationship on the basis of plugin that we have implemented. Use case diagram is built using the Java technology. The generated use case model is visualized using the Visual Paradigm tool. This subsection defines the generation process of use case diagrams from user stories using

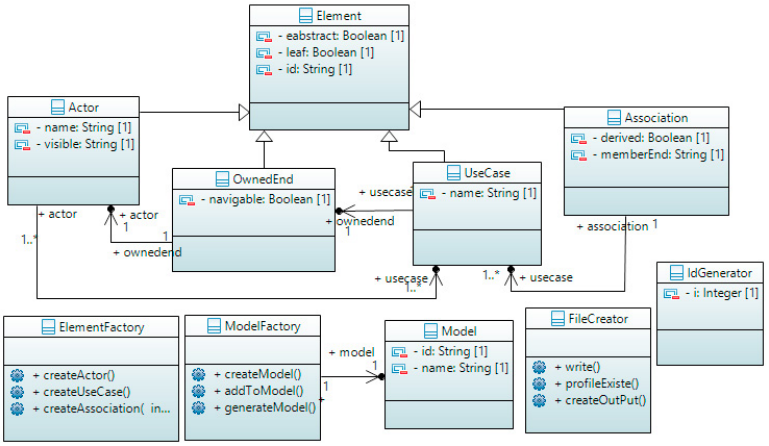

Fig. 1. Meta-model of Use Case Plugin

TreeTagger parser[35]. Applying POS tags, terms are categorized into proper noun, personal noun, noun (singular or plural) or verbs, etc. Each term in the user story is classified into a single part of speech. For the development of our use case generation plugin, we have implemented 10 java classes that we present as follows. The FileCreator class consists in the creation of physical output files. The generation of ids is performed using the IdGenerator class. For each element of the model, we insert automatically a basic Id. The ElementFactory class aims to create all elements of the use case diagram (Creation of actors, use cases and their associations). We have represented associations with ends (sEnd, eEnd), to associate the actor with the corresponding use case (sEnd= start End and eEnd= finish End). The ModelFactory class allows creation of models, adding elements, which can be an actor, association relationship or use case, and finally generating the model as (.uml) file. Actor, Association relationship and use case classes describe models for actor, associations and use cases. The Model class contains a collection of actors, associations and use cases. The OwnedEnd class specifies end for the actor or use case. Fig 1 illustrates the set of classes implemented that represent the use case plugin. To extract UML use cases diagrams automatically from user stories, our plugin accepts only sentences that respect the syntax proposed by Wautelet[36]: (As a(n)) for the actors, (I want to / I can / I am able) for the actions, and (so that) for the benefit. Actor represents the role that a user can play within a system. Before creating an actor, we first check whether he already exists or not. Use case allows us to describe functional

requirements. Associations represent the relationship between each actor and use case. In the following we present the transformation rules applied.
R1. Each first noun (NN), noun plural (NNS) or compound noun (NNs && NNs) is an Actor. **R2.** The identification of use cases is generally linked to actors. The use cases detected are predicates (Verbs [base form (VV) or past participle (VVN) or participle (VVG) or past tense (VVD)] + Noun [singular (NN) or plural (NNS)]) in each user story, which are associated with an actor (NNs). **R3.** In each sentence, an association relationship will be between R1 and R2, i.e., the associations relationship are triggered between each actor and use case.

3.2. Extraction Algorithm

To extract UML Use Case Diagram from a set of user stories, we have followed the steps in the pseudo-code presented below. The algorithm takes as input a set of user stories stored in a text file. After filtering the text file and eliminating unnecessary terms, we get a new file that contains only (nouns/ compound noun) and (verbs). Therefore, the system takes as input the new file. We have defined two Booleans (line 2). The first Boolean named (isFirst) detects the first (nouns/compound noun) in each user story, and the second Boolean searches in the history if the (nouns/compound noun) already exists (line 8) or not. If the first (nouns/compound noun) is detected, the loop checks if the word was already detected in previous sentences (lines 9-11). If founded, the actor will not be created (line 12). Otherwise, new actor will be created (line 16). Lines (17-21) allow to find all the (verbs) followed by (nouns/compound noun) in each user story to create use cases. Each actor is associated with its use cases by an association relationship (line 22). For the creation of these elements, we use the plugin classes that we have already implemented, and the output generates (.uml) files containing the overall UML Use Case diagram.

Algorithm 1 Pseudo-code that builds use case diagrams from user stories by applying NLP techniques

```
1:  Input: {As a...., I want to..., so that....}
2:  Boolean: isFirst, isThere
3:      store ← actors that already exist
4:      currentActor ← the current actor
5:      POS ← part-of-speech
6:        isFirst ← True
7:        isThere ← False
8:      if(((POS==NNs)——((POS==NNs)&&(POS==NNs)))&& isFirst)
9:          for each (Actor a)
10:           if(isThere ← True)then
11:             search a
12:             currentActor ← store
13:         end for
14:           else
15:          if(isThere ← False) then
16:             a ← createActor
17:     if((POS==VV)——(POS==VVN)——(POS==VVG)——(POS==VVD))
18:       verbe ← add(token)
19:     if(((POS==NNs)——((POS==NNs)&&(POS==NNs)))&& !isFirst)
20:       verbe ← add(token)
21:       UseCase u ← ElementFactory.createUsecase(verbe)
22:       Association assoc ← ElementFactory.createAssociation(CurrentActor, u)
```

4. Simulation Work

4.1. Accuracy Assessment

We assessed the feasibility and accuracy of our approach by applying the plugin on a dataset obtained from the case study WebCompany which is available online[37]. We have made some minor changes to address the cases of

compound noun (for actors). The evaluation was based on a comparison of the outputs automatically generated by the plugin and the manual modeling of each user story. The objective of this evaluation is to check the error rate in bad tagging user stories by NLP tools and its impact on the generation of UML models. The elements true positive (TP), false negative (FN) and false positive (FP), that apply to Actor, Use Cases and their association relationship are defined as follows: **True positive (TP):** The actors, use cases and their relationship are identified both manually and by the plugin. **False positive (FP):** The actors, use cases and their relationship are identified by the plugin but not manually. **False negative (FN):** The actors, use cases and their relationship are identified manually but not by the plugin. Accuracy is assessed by comparing the calculation results of the number of true positives, false positives and false negatives for actors, use cases and their relationship obtained automatically compared to those obtained manually. Tables 1 and 2 show all the actors, use cases and their relationships that we have found manually and automatically from 90 user stories. Table 3 shows the calculation results. We divided the table 3 into three columns: actors, use cases and relationships. Each column has three sub-columns that specifies true positive (TP), false positive (FP) and false negative (FN).

Table 1. Total of items detected manually

Actors	Use Cases	Relationship
90	168	168

Table 2. Total of items detected automatically

Actors	Use Cases	Relationship
89	163	163

The manual UML use case diagram contains 9 actors, because a set of user stories contains (Administrator is repeated 30 times), (Visitor is repeated 39 times), (User is repeated 15 times), (newly registered user is repeated once), (Trainer is repeated once), (ATM User is repeated once), (ATM Operator is repeated once), (Site editor is repeated once) and (Developer is repeated once). The automatic UML use case diagram generation contains 8 actors, because a set of user stories contains (Administrator is repeated 30 times), (Visitor is repeated 39 times), (User is repeated 15 times), (Trainer is repeated once), (ATM User is repeated once), (ATM Operator is repeated once), (Site editor is repeated once) and (Developer is repeated once). The only actor not detected is (newly registered user), because the syntax does not meet the condition. For the actors, the results are quite positive, the precision and recall are equal to 98%. For the use cases and their relationships the precision is equal to 87% and the recall is equal to 85%. These values are the results of the problems that we have encountered with the sentences requiring inclusion and exclusion relationships, which are not supported by the plugin. Fig 2 illustrates a part of the output of the generated.

Table 3. Accuracy of the generated uml use cases diagrams for the web-company case

Actors	Use cases	Relationships
TP FP FN	TP FP FN	TP FP FN
90 1 1	143 20 25	81 20 25
Precision= 98%	Precision= 87%	Precision= 87%
Recall= 98%	Recall= 85%	Recall= 85%

5. Conclusion

The majority of research work in the literature focuses on applying NLP tools for generating UML models from requirements document. However, the integration of NLP tools in agile requirements in particular user stories for generating UML models is few and far between. The work represented by Robeer[25] is quite similar to the present one, but they do not generate UML models. They generated conceptual models automatically from a set of user stories in the form of OWL ontologies, while we generated UML use case diagram automatically from a set of user stories. The advantage of this technique lies in its ability to facilitate the work of the development team and Product

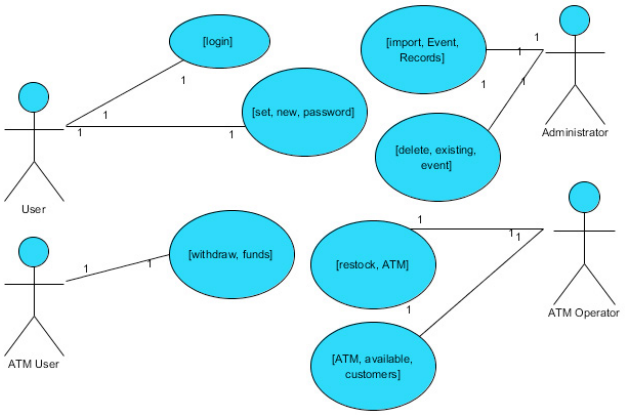

Fig. 2. UML Use Case Diagram generated

Owner in reducing ambiguity in requirements specifications, and generating automatically design models. The benefit of generating UML use case diagram automatically make easy the understanding and help designers to interpret the same user story in one way, which combines the teams in the design process. Furthermore, it allows gaining time because designers can generate UML models from a set of user stories in a short time. TreeTagger parser was chosen because it represents a good way to categorized terms into proper noun, personal noun, noun (singular or plural) or verbs, etc. Given that the syntax of user stories is simple and semi-structured, which allows to classify each term in the user story into a single part of speech easily. Visual Paradigm tool was used to display diagrams in (.uml) format. The results that we have obtained are positive. User stories have a simple and understandable syntax, which makes NLP analysis and the treatment sentences an easy task to achieve. The problems encountered in the case study are derived from complicated sentences, which often require inclusion or exclusion relationships between use cases. This type of relationship is not yet supported by our plugin. Also, our plugin does not support sentences containing more than one compound noun, such as (Administrator database manager). Other types of relationship, such as generalization and specialization between actors and use cases will be addressed in our future work. We can notice that the manual modeling of 90 user stories can take a lot of time, while it's much faster through the plugin. The benefit of integrating the NLP techniques for analyzing user stories lies in its ability to classify each term of user story into a single part of speech, which facilitates the connection of requirements to design and the generation of the results in a common language such as UML. We believe that this technique can facilitate the work of designers; this method allows clear interpretation of user stories and can help software analyst to reducing time of drawing use cases diagrams and improving workflow. Also, use cases diagrams are useful because they represent the user needs and describe sequences of actions performed by the system.

References

1. K. Schwaber, Scrum Development Process. In OOPSLA Business Object Design and Implementation Workshop, Eds. London: Springer, pp. 117-134. (1997) . DOI: 10.1007/978-1-4471-0947-1_11.
2. L. Cao, B. Ramesh, Agile requirements engineering practices: an empirical study, IEEE Software, vol: 25, Issue: 1. (2008) . DOI: 10.1109/MS.2008.1
3. M. Kassab, The changing landscape of requirements engineering practices over the past decade. In: Proceedings of the IEEE international workshop on empirical requirements engineering (EmpiRE). IEEE, pp 18. (2015). DOI: 10.1109/EmpiRE.2015.7431299.
4. X.Wang, L. Zhao, Y. Wang, J. Sun. The role of requirements engineering practices in agile development: an empirical study. In: Proceedings of the Asia Pacific requirements engineering symposium (APRES), CCIS, vol 432, pp.195209. (2014) . DOI: 10.1007/978-3-662-43610-3_15.
5. OMG, MDA. http://www.omg.org/mda/, Accessed January (2016).
6. A. Kriouile, N. Addamssiri and T. Gadi, An MDA Method for Automatic Transformation of Models from CIM to PIM. American Journal of Software Engineering and Applications, 4, pp.1-14. (2015) . https://doi.org/10.11648/j.ajsea.20150401.11
7. S. Kherraf, E. Lefebvre and W.Suryn,"Transformation from CIM to PIM Using Patterns and Archetypes," in 19th Australian Conference on Software Engineering, (2008) . DOI:10.1109/ASWEC.2008.4483222.
8. A. Rodrguez, E. Fernndez-Medina and M. Piattini, "Towards obtaining analysis-level class and use case diagrams from business process models," Advances in Conceptual ModelingChallenges and Opportunities. Springer Berlin Heidelberg, pp. 103-112, (2008). DOI: 10.1007/978-

3-540-87991-6_15
9. A. Rodrguez, I. G.-R.de Guzmn, E. Fernndez-Medina and M. Piattini, "Semi-formal transformation of secure business processes into analysis class and use case models: An mda approach," in Information and Software Technology, 52(9):945 971, (2010). DOI: 10.1016/j.infsof.2010.03.015
10. A. Rodrguez, E. Fernndez-Medina, J. Trujillo and M. Piattini, "Secure business process model specification through a UML 2.0 activity diagram profile," Decision Support Systems, vol. 51, no. 3, pp. 446-465, (2011). DOI: 10.1016/j.dss.2011.01.018
11. N.J. Kamarudin, N.F.M. Sani and R.Atan. Automated transformation approach from user requirement to behavior design, Journal of Theoretical and Applied Information Technology, Vol. 81, No. 1, pp. 73-83. (2015). http://www.jatit.org/volumes/Vol81No1/9Vol81No1.pdf
12. D. Bell, UML basics: An introduction to the unified modeling language, IBM Developer Works. (2003). http://www.ibm.com/developerworks/rational/library/769
13. A. Copestake,Natural Language Processing, Lecture Synopsis. (2004). https://www.cl.cam.ac.uk/teaching/2002/NatLangProc/revised.pdf
14. A. Lash, M. Kevin, and M. Gregory. Natural language processing applications in requirements engineering. ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers. (2012). DOI: 10.1115/DETC2012-71084
15. D.D. Kumar, D.D., M.A. Babar, An Automated Tool for Generating UML Models from Natural Language Requirements. Automated Software Engineering, pp. 680-682. (2009). DOI: 10.1109/ASE.2009.48
16. D.D. Kumar, D.D., R.Sanyal, Static UML Model Generator from Analysis of Requirements (SUGAR). In Advanced software engineering and its applications, (ASEA 2008), pp. 7784. (2008). DOI: 10.1109/ASEA.2008.25
17. P. Kruchten, The Rational Unified Process: An Introduction. Addison Wesley, Second Edition, USA. (2000). ISBN: 0201707101
18. P. More, R. Phalnikar, Generating UML Diagrams from Natural Language Specifications. International Journal of Applied Information Systems, Foundation of Computer Science, vol.1, no.8, pp. 19-23. (2012). DOI: 10.5120/ijais12-450222
19. H. Herchi, W.B. Abdessalem, From user requirements to uml class diagram, CoRR, abs/1211.0713. (2012). https://arxiv.org/ftp/arxiv/papers/1211/1211.0713.pdf
20. N. Zhou, X. Zhou, Automatic Acquisition of Linguistic Patterns for Conceptual Modeling. INFO629: Concepts in Artificial Intelligence. (2004). https://pdfs.semanticscholar.org/b809/d4947a9d544cded70446da3c6bfdf4e0256a.pdf
21. L. Mich, NL-OOPS: From Natural Language to Object Oriented Requirements Using the Natural Language Processing System LOLITA. Natural Language Engineering, (1996), vol. 2, no. 2, pp. 161-187. DOI: 10.1017/S1351324996001337
22. A. Vrushali, A. Darshana, J. Aarti and L. Dipali, Class Diagram Extraction from Textual Requirements Using NLP Techniques. IOSR Journal of Computer Engineering (IOSR-JCE), volume 17, issue 2, Ver. III, pp. 27-29. (2015). DOI: 10.9790/0661-17232729.
23. D. K. Deeptimahanti, R. Sanyal, Semi-automatic generation of UML models from natural language requirements. In Proceedings of the 4th India Software Engineering Conference. ACM, 165174. (2011). DOI:10.1145/1953355.1953378
24. R. Madanayake, G.K.A. Dias, N.D. Kodikara, User Stories vs UML Use Cases in Modular Transformation, International Journal of Scientific Engineering and Applied Science (IJSEAS) Volume- 3, Issue-1. ISSN (Online): 2409-4285. (2017).
25. M. Robeer, G. Lucassen, J.M.E.M.Van der Werf, F.Dalpiaz, S.Brinkkemper, Automated Extraction of Conceptual Models from User Stories via NLP. In: Proc. of RE (2016). (2016). DOI: 10.1109/RE.2016.40.
26. K. Letsholo, L. Zhao, E.-V. Chioasca, TRAM: A tool for transforming textual requirements into analysis models. in: Proc. ASE 2013, pp.738741, tool demonstration. (2013). DOI:10.1109/ASE.2013.6693146
27. L. Prasad, S. Patidar, Transformation of analysis model to design model. 2010 International Conference on E-business, Management and Economics IPEDR vol.3, IACSIT Press, Hong Kong. (2011). http://www.ipedr.com/vol3/44-M10006.pdf
28. V.B.R.Vidya Sagar, S.Abirami, Conceptual modeling of natural language functional requirements. Journal of Systems and Software. 88, 2541. (2014). DOI: 10.1016/j.jss.2013.08.036.
29. T. Yue, L. Briand, Y. Labiche, A systematic review of transformation approaches between user requirements and analysis models. Springer: Requirements Engineering, pp. 7599. (2011). DOI: 10.1007/s00766-010-0111-y.
30. 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. 14431458. (2016). DOI: 10.1002/spe.2384.
31. M. Abdouli, W.B.A. Karaa, H.B.Ghezala, Survey of works that transform Requirements into UML Diagrams, 2016 IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA), pp. 117-123. (2016). DOI:10.1109/SERA.2016.7516136.
32. C.C. Osman, P.G. Zalhan, From natural language text to visual models: A survey of issues and approaches. Informatica Economica, vol.20, no.4/2016, 4461. (2016). DOI: 10.12948/issn14531305/20.4.2016.01.
33. H. R. Sharifi and M. Mohsenzadeh, "A New Method for Generating CIM Using Business and Requirement Models.," World of Computer Science and Information Technology Journal (WCSIT), vol. 2, no. 1, pp. 8-12, (2012).
34. W. Zhang, H. Mei, H. Zhao and J. and Yang, "Transformation from CIM to PIM: A Feature-Oriented Component-Based Approach," in Model Driven Engineering Languages and Systems volume 3713 of Lecture Notes in Computer Science, pages 248263, Springer Berlin / Heidelberg, (2005). DOI: 10.1007/11557432_18
35. G. Schmid, TreeTaggerA language-independent part-of-speech tagger. (1994). http://www.ims.uni-stuttgart.de/Tools/DecisionTreeTagger
36. Y. Wautelet, S. Heng, M. Kolp, and I. Mirbel, Unifying and Extending User Story Models, in Advanced Information Systems Engineering, ser. LNCS, vol. 8484. Springer, pp. 211225. (2014). DOI: 10.1007/978-3-319-07881-6_15.
37. L. Garm, Improving Effectiveness of User Stories, Utrecht University, (2016). http://www.staff.science.uu.nl/˜lucas001/conc_modeling_user_stories.zip, (Accessed on June 01, 2017)
38. M. Elallaoui K. Nafil, R. Touahni, Automatic generation of UML sequence diagrams from user stories in Scrum process, in: 10th International Conference on Intelligent Systems: Theories and Applications, (2015). DOI: http://dx.doi.org/10.1109/SITA.2015.7358415
39. M.Elallaoui, K.Nafil, R.Touahni, "Automatic generation of TestNG tests cases from UML sequence diagrams in Scrum process". 2016 4th IEEE International Colloquium on Information Science and Technology (CiSt),(2016), pp.65-70, DOI : 10.1109/CIST.2016.7804972

View publication stats