UML Generator – Use Case and Class Diagram Generation from Text Requirements

Chamitha Ramal Narawita, Kaneeka Vidanage

***Abstract*— This research mainly is focused on automation of Unified Modeling Language (UML) diagrams from the analyzed requirement text using Natural Language Processing (NLP). The looks and styles of software engineering have been completely changed in the recent times. Presently, software engineering follows the rules of Object Oriented design patterns. All phases of software engineering are deviating from the conventions and new paradigms are more popular these days. Same is the case with Software analysis process which uses Unified Modeling Language to map and model the user requirements. Analysis is the key process of building modern information system applications and base for the robust and vigorous software application's design and development. Nowadays everybody needs a quick and reliable service. It was necessary to have some sort of quick, accurate and intelligent software for generating UML based documentations to save time and budget of both the user and system analyst [1].**

***Keywords*— UML Diagrams, Use Case Diagram, Class Diagram, Natural Language Processing, Artificial Intelligence, Machine Learning, Software Design Phase**

I. Introduction

The automation of UML Diagrams using natural language processing is a highly challenging task due to the following reasons [2].

- Natural language is arguable. Thus, detailed and error-free analysis is very difficult.
- There could be different ways of representing the same semantic.
- Concepts that were not explicitly expressed in a written source are often very difficult to model. Usually, expert domain knowledge is needed to identify the hidden classes.

Design phase is the most important among the other phases because blueprint of a system helps developers to avoid all the misunderstanding regarding the software by involving the users. Requirement engineers analyze requirements manually to come out with highly precise UML diagrams for their systems. By modeling a system, most important aspect is to capture the dynamic behaviors. Static behaviors are not sufficient to build models for a system rather using dynamic behaviors. Use case diagram shows dynamic aspects of a system with both internal and external interactions. They describe the events of a system and their flows. However, the use cases never describe how they are implemented. When it comes to class diagrams its other way around. The class diagrams can be defined as static diagrams and they represent the static view of a system. A class diagram contains classes of the system, their attributes, operations and constraints imposed on the system. They can be directly mapped with the object-oriented languages as well. Therefore, to provide a dynamic and a static view of the scenario, UML Generator generates use case and class diagrams by analyzing the input text. This research will be contributing towards filling the gap between gathered user requirements and the phase of the implementation by sorting out the problems mentioned above[3].

According to the norms and conventions, before drawing the UML diagrams the system analyst has to do a lot of work by analyzing the business logics and figuring out the user requirements. Out of the tools, the author considered the absence of a software which provides services by manually drawing UML diagrams more efficiently except Rational Rose and Smart Draw and there is no doubt that these are reasonably good software but with many disadvantages. First, analysis is needed to investigate the requirements and then draw the models separately. Hence, there is wastage of so much time when using current available tools to create UML models for the required scenario. Nowadays, everybody needs a quick and reliable service. Moreover, the time spent on analyzing systems and poor quality of human analysis shows the need of automated support.

Manuscript received on 12th November, 2016. Recommended by Dr. T.M.H.A.Usoof on 4th May, 2017. This paper is an extended version of the paper, "UML Generator An Automated System for Model Driven Development" presented on ICTer2016 Conference.

Chamitha Narawita is a BEng(Hons) student at University of Westminster and following Master of Computer Science (MSC) at University of Colombo. Currently working as a Software Engineer at Calcey Technologies. (e-mail: chamithanara@gmail.com)
Kaneeka Vidanage is an M Phil candidate in semantic web from University of Colombo and a lecturer at Informatics Institute of Technologies.. (e-mail: shehan@cse.mrt.ac.lk)

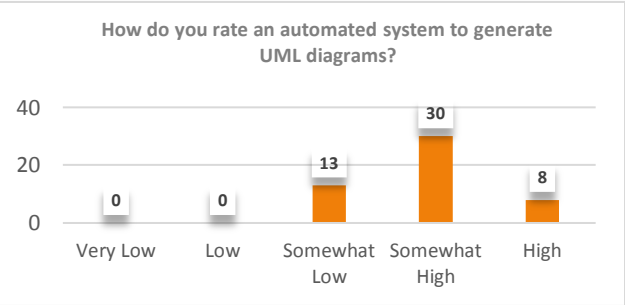


Fig. 1 Importance of an automated system

As shown in figure 1 according to the survey carried out to gather requirements, most of the users have agreed to the researcher's opinion, which 75% (38 out of 51) of overall responses have positive thoughts regarding this system.

II. Literature Review

A. Natural Language Processing (NLP)

Once the user enters the requirements, there should be a mechanism to extract the information and to understand the text from currently available techniques. To analyze a large amount of text data, currently NLP is the only available technique for the developers. NLP is an area of research and application that explores how computers can be used to understand and manipulate natural language text or speech to do useful things [7].

User requirement analysis is an information extraction application of NLP. It is the identification of specific semantic elements within the user's requirements entered in textual form [3]. Therefore, proposed system comes under information extraction over the Information Retrieval and Question Answering Tasks in NLP [5].

To extract the information from a given text, several approaches were used in natural language processing.

1) Sentence Splitting

Using this approach, the proposed system is expected to split the all text into sentences after user entering the requirements.

2) Lexical Analysis

Lexical analysis approach will get the split sentences and it will tokenize the sentences into words.

3) Syntax Analysis

From this, the module receives the lexical tokens as the input and applies a rule-based approach. It generates outputs in form of parts of speech in a given text.

4) Word Chunking

By using a chunking approach to the proposed system, the main expectation is to derive the use cases from the input text. It identifies noun phrases (NP), verb phrases (VP), and prepositional phrases (PP) using tokenized text and POS tags.

UML Generator is a web application, which was developed using C#.Net framework. SharpNLP has been used as the NLP library, which is the C# open port of the Java OpenNLP tools, plus additional code to facilitate NLP.

B. Machine Learning

By using machine, learning with the UML Generator it understands the key features of the use case and class diagrams as follows.

- Identify meaningful and not meaningful use cases (rate use cases).
- Identify relationship types in use case diagram (associations between use cases and actors, include, extends and generalizations).
- Identify relationship types in class diagram (associations between classes, aggregations, compositions and generalizations)
- Identify multiplicities in a relationship in class diagram (Zero to One, One to One, Zero to Many, One to Many, Many to Many).

Table I shows the accuracy levels of the algorithms for the rate use cases model built in Weka. The author has considered both efficiency and accuracy levels of the algorithms when training the Weka models.

TABLE I
Accuracy Levels of the Classifiers

Classifier	Description
Multi-layer Perceptron	Multi-layer Perceptron has 100% of accuracy with the use case rating model. But it has spent 270.28 seconds to build the model. Time taken to build model: 270.28 seconds Correctly Classified Instances 241 100% Incorrectly Classified Instances 0 0% However, there could be more than one use case in a scenario to rate when running it real time. Therefore, using MLP can raise various performance issues with the system.
Logistics	Time taken to build model: 0.35 seconds Correctly Classified Instances 196 81.3278 % Incorrectly Classified Instances 45 18.6722 % Logistics has a decent amount of good accuracy level with the model and it has spent only 0.35 seconds to build the model
Sequential Minimal Optimization (SMO)	SMO has a lower accuracy level than the Logistics. Time taken to build model: 0.24 seconds Correctly Classified Instances 189 78.4232 % Incorrectly Classified Instances 52 21.5768 %

Weka has been used as the machine leaning tool to develop the UML Generator. Taking into consideration about the

accuracy levels and performances, Logistics classifier has been used to train the use case model.

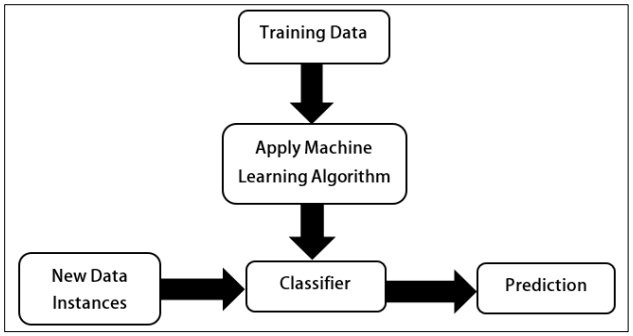


Fig. 2 Text classification flow

Fig. 2 shows the text classification flow of the UML generator.

The relationships between use cases and classes were recognized using an algorithm written in C#. Figure 3 shows an example of the use case relationship generation using Weka vote classifier. Same algorithm has been used to extract the relationships between classes as well.

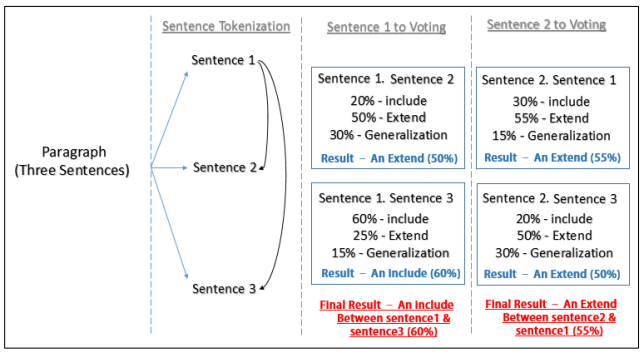


Fig. 3 Relationship recognition algorithm

Let us assume there is a paragraph that consists of three sentences. Using Natural Language Processing sentence tokenization happens and then C# algorithm will combine the sentence 1 and sentence 2 as shown in the figure. Then it will send to the Weka relationship recognition model and will output the voted values for each relationship type. Same process will happen between sentence 1 and sentence 3 as well. Now there are two maximum values have been selected and the highest voted value will be the relationship type between two sentences.

As earlier mentioned Weka models have been used to recognize the relationships between use cases and classes. In the Weka models, the author has defined some specific words to identify the relationship types.

- Must, should, required to, etc. indicates the Include relationships
- Maybe, sometimes, either, etc. indicates the Extend relationships. [8]
- Divided to, either by, is a type of etc. indicates the use case and class Generalizations. [9]
- In, to, by, etc. indicated the class Associations.
- Must, have, first, has to, etc. indicates the Aggregation. [8]
- As an option, depends on, do not have, etc. indicates Composition.

C. Rule Based Approach

A rule-based system is used in the field of computer science, as it is a series of if-then statements using assertions. These assertions dictate how each rule will react, given the assertions already in place [4]. In the proposed system, rule-based approach will assist with the following functionalities

- Remove unwanted terms in the user input text.
- Identify specific terms in the input text.
- Define Weka ARFF file names to read the files.

The best approach to implement the rule based approach is using XML. By using XML rules, we can define an own structure for the XML file and develop our own algorithm to execute the rules. 0

```
<?xml version="1.0" encoding="utf-8" ?>
<Rules>
  <Rule Name="Check_consecutive_Nouns" Type="Class">
    <Conditions LogicalOperation="OR">
      <Condition FieldName="word" Operation="Equals" Value="number"/>
      <Condition FieldName="word" Operation="Equals" Value="no"/>
    </Conditions>
  </Rule>
</Rules>
```

Fig. 4 Structure of the rules XML file

Fig. 4 shows the structure of the XML file that has been used to define the rules.

Root element of the XML file is defined as *Rules*. Within the root element, can define any number of Rule elements with the name and the type of the rule whether it belongs to class diagram (*Class*) or use case (*UseCase*) diagram. Using *Condition* elements, it defines specific terms inside the *Value* attribute.

D. Diagram Generation

The output of this system is a visual studio modeling diagram project that consists of generated use case and class diagram using selected actors, classes, use cases, attributes, use case relationships and class relationships.

There are many CASE tools to draw the UML diagrams. StarUML and Rational Rose are some of famous tools currently using in the industry. UML Generator is an integrated solution for the Visual Studio (VS). To integrate

a software to an existing Microsoft product, all the technologies and tools that have been used to develop the software should have Microsoft license. Therefore, visual studio modeling can be used to generate the final outputs of UML Generator. The all three tools (Rational Rose, StarUML and VS Modeling) are currently not providing any documentation or an API to override their product. However, the base of VS Modeling with XML is simple to understand rather using StarUML or Rational Rose.

E. Related Work

Few researchers have done UML diagram generation using natural language processing. After understanding the importance of automating the UML diagrams, most of the researchers have used natural language processing and domain ontology to get expected outcomes.

Using rule-based module specifies subject nouns as objects, verbs as methods of the objects, and adjectives as attributes of the object. Object nouns are sometimes specified as objects and sometimes as attributes. Then associations and relationships among extracted classes are performed. Finally, a logical model of the class diagrams generates based on previously extracted Information. The drawing module converts the logical model into the class diagrams by connecting small pieces of images already stored in the database [3].

Most of the researchers have used only NLP, rule-based approach to achieve the task and only generates the class diagram or the use case diagram. By creating an image as the final output, users will not be able to add their own modifications, which limits the user to the system.

A research done by Bajwa and Hyder using LESSA approach was all about the automatic generation of the Use Case diagrams after reading and analyzing the given scenario in English language provided by the user. Their designed system could find out the classes and objects together with their attributes and operations using an artificial intelligence technique such as natural language processing. Then the Use Case diagrams are drawn [6]. NLP is the core technology that has used to extract the information from the text.

F. Gathered Information

To gather the requirements the author has used online questionnaires and interviews.

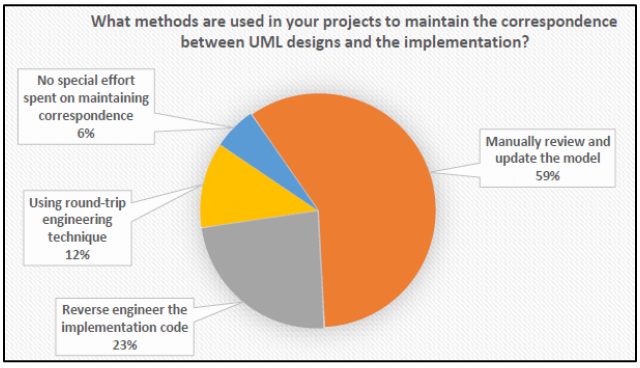


Fig. 5 Current methods to darw UML diagrams

Considering the current methods used to draw the UML diagrams, as shown in figure 5, 59% of the respondents still manually review and update the model, which is more time consuming and is the traditional way of drawing UML diagrams.

The proposed system delivers both static and behavioral features of a system. It generates a use case diagram to express the behavioral behaviors and static behaviors are expressed by the class diagram. However, in UML, there are many diagrams, which deliver static features and behavioral features of a system.

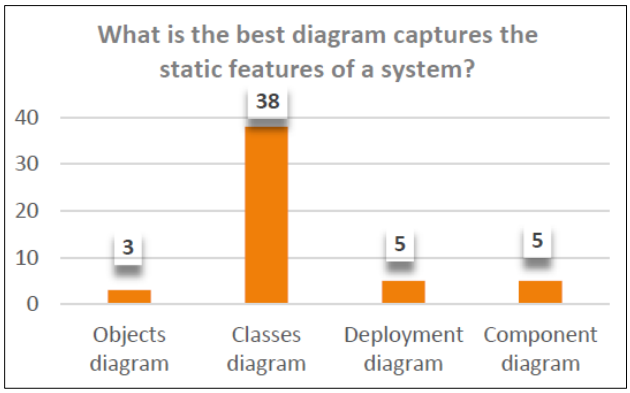


Fig. 6 Static features of a system

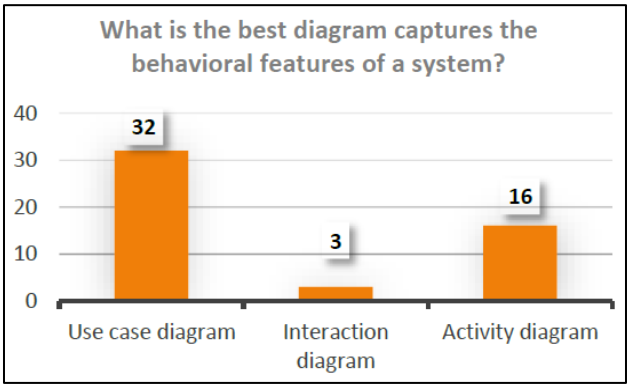


Fig. 7 Behavioral features of a system

As shown in figure 6 and 7 respondents have responded to the questions that asked regarding best diagrams to capture the static and behavioral behaviors of a system (researcher does not mention that the proposed system is going to generate use case and class diagrams). According to the

results, a class diagram is the most desired diagram among the respondents that captures static features. Considering about behavioral features, use case has obtained higher number of responses. However, activity diagram has 16 responses as well. Considering the limitations in automation of activity diagram needs higher level of intelligence, which describes sequences between activities of the system. Hence, use case diagram could be a good solution to deal with the scope of this research.

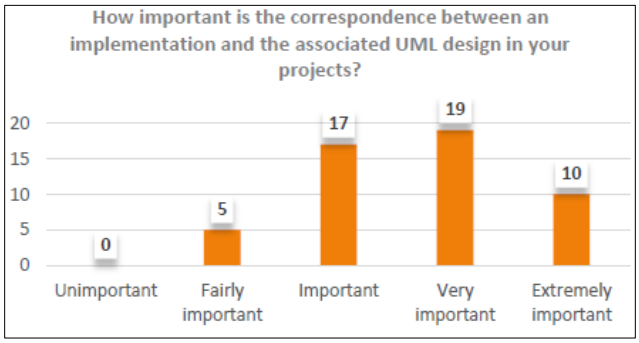


Fig. 8 Importance of correspondent between UML diagrams and implementations

Figure 8 shows how important it is to maintain UML designs correspondent to the implementations. 90% of the respondents have mentioned correspondence between implementation and the associated UML designs are important.

For this questionnaire, the targeted users were software developers, project managers and business analysists in the industry because they use UML diagrams in their day-to-day life.

G. *Functional Requirements*

TABLE III
FUNCTIONAL REQUIREMENTS

No.	Functional Requirement	Description
FR01	Recognize use cases	Using NLP and Weka models recognize the use cases from user input text
FR02	Extract relationships between actors and use cases	From the user input text recognize the different types of relationships in use case diagram
FR03	User select actors for use case	System suggested actors for the use case to be drawn. User can remove unwanted actors suggested by the system.
FR04	Recognize classes for class diagram	Using NLP recognize the classes from user input text
FR05	Extract relationships between classes	From the user input text recognize the different types of relationships in class diagram
FR06	User select classes for class diagram	System suggested classes for the class diagram to be drawn. User can remove unwanted classes suggested by the system.
FR07	User select relationship types for class diagram	System suggested relationship types between classes. If there are any wrong suggestions by the system, user can change to the desired one.
FR08	Extract multiplicities between classes	From the user input text, recognize the multiplicities in class diagram
FR09	Generate use case and class diagram XML files	Using all the data gathered from user input text system generates use case and class diagram XML files for visual studio
FR10	Real time validates the user input text	User has to enter the requirements according to pre-defined rules. If there is any mismatch in the text, system draws a red line under the sentence.

H. *Non Functional Requirements*

Non-functional requirements define the qualities and system attributes that can be used to evaluate the system. Following are the non-functional requirements of the proposed system.

1) Performance

a. Efficiency: System should generate the use case and class diagrams quickly (fast respond).

b. Accuracy: should provide high level of accuracy when extracting the elements in use case and class diagram.

2) Usability

The user interface of the system should clear, simple, attractive, consistent, and easy to understand.

3) Maintainability

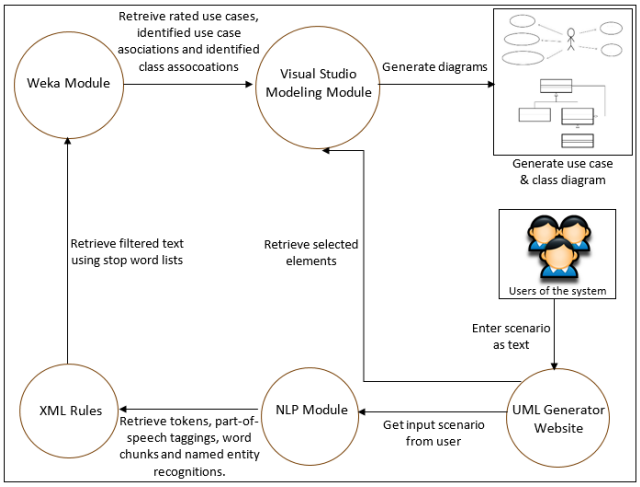
Application should easily maintainable. Implementations should follow the coding standards, which improves readability and understandability.

4) Reliability

System should always behave consistently by acquiring user needs. The use case and class diagrams,

which generates by the system should have higher level of accuracy.

III. High Level Architecture



There are five main modules in the UML Generator with the website. Users of the system interact only with the website of the UML Generator. User can upload a text file that contains a scenario or just copy and paste the text in the text area. The system extracts the user cases, actors, classes, attributes using both NLP module and XML Rules. After identifying use cases, Weka module rates the use cases as "use case" and "not a use case". Associations were extracted using Weka modules according to the C# algorithm that author has implemented.

Visual studio modeling draws the use case and class diagram. The system suggests the findings to the user and they can filter the use cases, actors, classes, attributes and all the relationship types. Therefore, the output is highly customizable and user can get the desired output according to the requirements.

IV. Proposed Approach

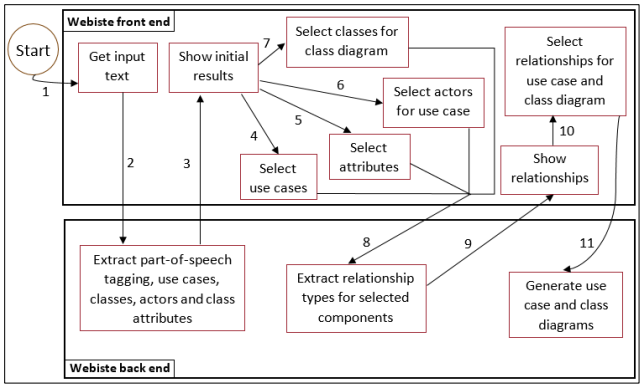


Fig. 10 Workflow diagram of the website

Fig. 10 shows the workflow diagram for the UML Generator. The user has to go through maximum of eleven (11) flows to generate the use case and class diagram from the input text scenario.

1. The user starts the web application by entering the URL of the website. After finish loading, the UML Generator web application user has to enter the requirement text inside the text area.
2. Entered text scenario will send to the NLP module and then text tokenization will happen. Then extracts part-of-speech (POS) tagging and UML Generator identifies actors for the use case diagram and classes for the class diagram using nouns of the POS tag vales. XML rule based approach removes the unwanted words from the identified nouns list and will output further accurate results to the user.

   Use cases were identified using the word chunking technique in NLP. Usually to be an usecase there should be a verb and a noun. Word chunking recognizes verb phrases, noun phrases and prepositional phrases. Therefore, use cases were created using consecutive verb phrases with a noun phrases.

   E.g.: [**VP** inserts/VBZ] [**NP** ATM/NNP card/NN]
   VP – Verb Phrase, NP – Noun Phrase
   Use case: <u>inserts ATM card</u>

   Moreover, the XML rule based approach has been used to filter the unwanted words from the recognized use cases. A trained Weka model has been used to rate the use cases that identified using NLP. Each use case will send to the Weka model and rated as "Use case" or "Not a Use Case".

   Attributes were identified using XML rule based approach. In the XML file, words have defined to recognize as the attributes. Moreover, if there are two consecutive nouns and if the second noun is *number, no, type, code, date, volume, id, name, address, year, record, etc.* it has identified as an attribute.
3. Shows the Initial findings to the user on the website.
4, 5, 6, 7: User can select and filter identified use cases, attributes, actors for use case diagram and classes for the class diagram. The filtering steps are optional to generate the diagrams. User can proceed with the initial findings as it is.
8. Using initial findings, relationships were extracted between use cases and classes. With the relationships recognition algorithm mentioned earlier, the Weka vote algorithm has been used with the Logistics and SMO classifiers.
9. The extracted relationships will be shown using tables in the UML Generator website.
10. User can make further changes by changing the relationship types between use case and classes which identified by the system.
11. By clicking generate diagrams button, system will generate both use case and class diagrams as a visual studio modeling solution only for the selected elements by the user.

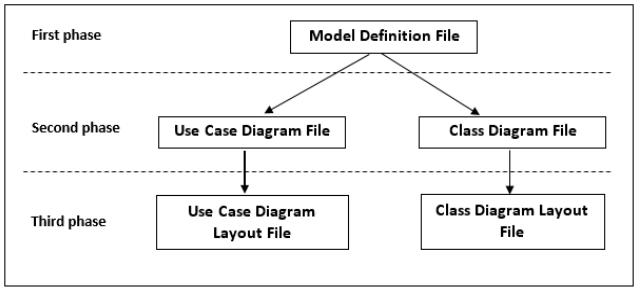


Fig. 11 The phases of diagram generation

The phases of diagram generation using VS has shown in Fig. 11.

V. Implementation

Based on the requirements of the UML Generator, a web application has been developed using Visual Studio .Net environment.

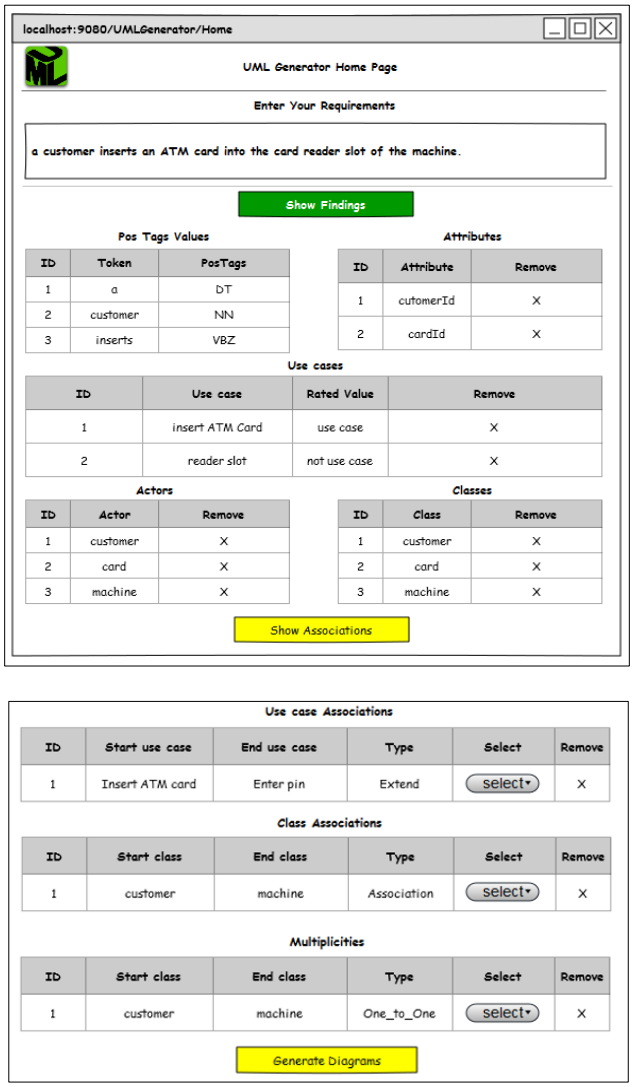


Fig. 12 Mockups of the website

The mockups of the website for initial findings and relationship recognition is shown in Fig. 12.

Phases of the implementation can be given as follows.

1. Natural language processing module to identify initial findings to generate diagrams – part-of-speech tagging, tokenizing, sentence splitting and word chunking.

2. Weka machine learning to identify association types of use case and class diagram – classification and voting.

3. XML rule based approach to identify attributes and filter the words.

4. Visual Studio modeling to generate use case and class diagram.

TABLE IIII
Technologies and Tools Used

Technologies	Tools
Microsoft .NET, XML, SharpNLP, C#, HTML CSS JS	Enterprise 2015 Visual Studio Microsoft, WEKA

The above table shows the technologies and tools used to develop the UML Generator.

VI. Testing

Currently author has tested the system with more than twenty (20) scenarios and it has an accuracy level of around 70%. This value has been calculated according to the passed test case results for all the scenarios.

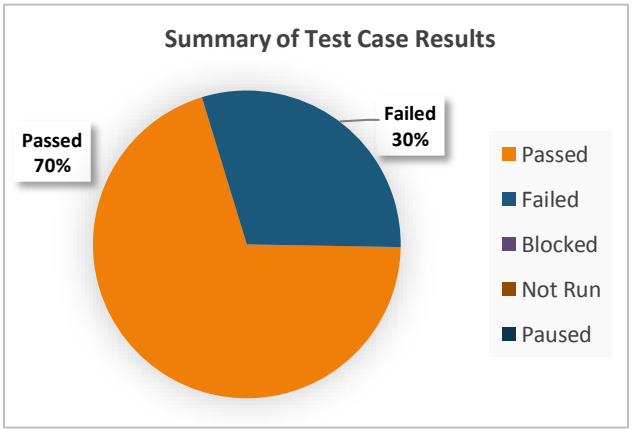


Fig. 13 Summary of the test case results

TABLE IV
BROWSER COMPATIBILITY OF THE SYSTEM

Browser Name	Status	Comments
Internet Explorer	Pass	Performed well and all the styling shown as is supposed.
Google Chrome	Pass	Performed well and all the styling shown as is supposed. Website was lagging when scrolling. However, after 5-10 seconds it became normal.
Mozilla Firefox	Pass	Performed well and all the styling shown as is supposed.
Microsoft Edge	Pass	Performed well and all the styling shown as is supposed. Better than internet explorer/Chrome/Firefox did.
Safari	Fail	Browser was not supported the file uploader and styling shown not as it supposed. All other functionalities are working and performance wise same as the other browsers.

Table IV shows the browser compatibility of the website.

VII. EVALUATION

Detailed discussions of the evaluation results are:

- Users will doubt the findings of the system. By adding a testing module to the UML Generator will enhance the accuracy level of the system.
- Few evaluators have mentioned that Visual Studio is not a suitable tool to generate the diagrams. The diagram generation is a separate module from other functionalities in UML Generator. Therefore, in future it will be easy to replace the diagram generation module with any diagramming tool. However, author's enduring idea is to introduce this application as a plugin to the Microsoft Visual Studio.
- Can extend the system by adding several other diagram types as plugins to the UML Generator. Then users can select the desirable diagram types, which are needed most.
- User experience needs to be enhanced by improving user interfaces, accuracy and performances of the system.
- Need to reduce the user involvement with the generation of UML diagrams. Currently before producing the diagrams, user can remove attributes, actors, classes, use case relationships and class relationships. By identifying highly accurate results, the system does not need to get user inputs, can directly generate the diagrams.
- Developers will forget UML by using the UML Generator. However, they will get used to do designing before the implementations because UML Generator gives a quick understanding about the both static and dynamic features of the scenario.

Moreover, evaluations were made by comparing automated diagrams with actually drawn diagrams.

Scenario of making an appointment: **"*A patient calls the clinic to make an appointment for a yearly checkup. The receptionist finds the nearest empty time slot in the appointment book and schedules the appointment for that time slot.*"**

Fig. 14 shows the actual drawn use case and class diagram for the above-mentioned scenario. According to evaluator's knowledge, he has identified four use cases and two actors for use case diagram and two classes for the class diagram. Fig. 15 shows automated diagrams using the UML Generator.

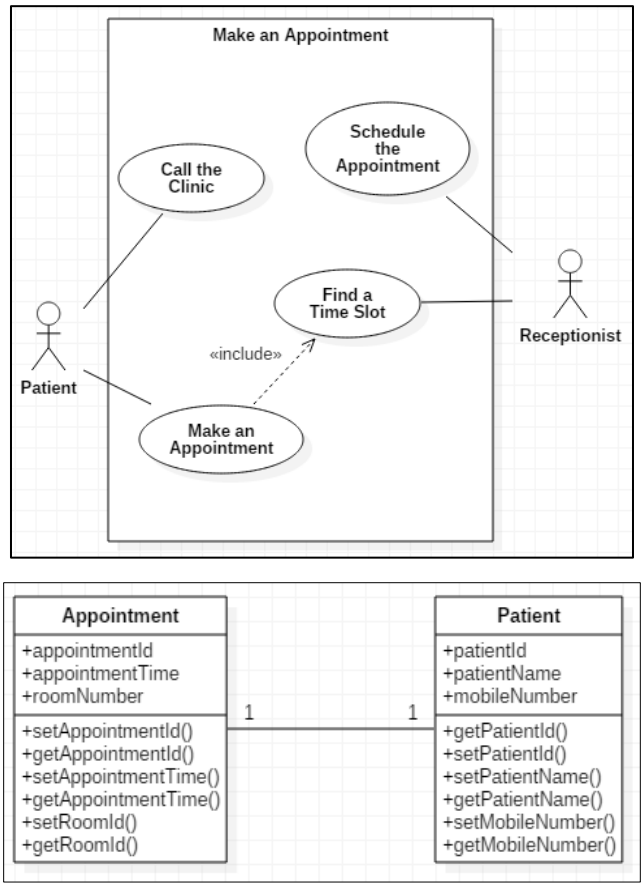


Fig. 14 Drawn diagrams by an evaluator

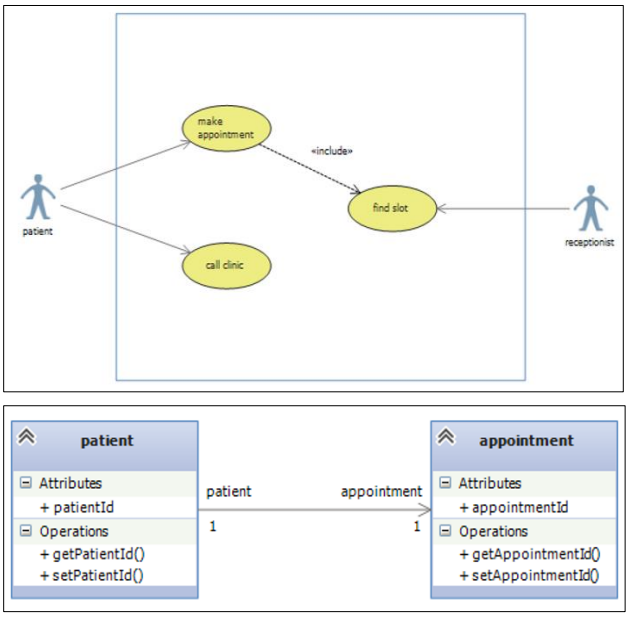


Fig. 15 Generated diagrams by UML Generator

As a conclusion, UML Generator has extracted most of the elements identified by the evaluator. "Schedule the Appointment" use case is the only unused element according to the results. Actually drawn class diagram has identified additional attributes and they are out of the scope (not included in the scenario). However, if this system is domain specific those attributes could be extracted by the system. In general, UML Generator has generated the use case and class diagram up to a considerable extent, which definitely gives a good overall idea about the scenario.

Following are some Evaluations made by evaluators.

1. Mr. Alexander Boltov (*University of Westminster, Department of Computer Science*) - "This is a very useful system when it comes to designing phase of software. This will definitely help for our academic work as well. I am very happy with the approach of the system because system gives chance to the user to play with the results. You need to check the accuracy of the generated diagrams against the hand drawn diagrams for a same scenario. It is good to add another module to test the generated diagrams whether it has identified all the elements that needs to include to the use case and class diagrams."

2. Mr. Antonis Michalas (*Head of the Cyber Security Group at University of Westminster)* – "UI is very user friendly and I am impressed about the customizations that can be done after identifying elements of the diagrams. Layout of the generated diagram is not a problem when it comes to an automated system, even when user can edit the diagrams after generating it. Overall, it is a good system."

3. Mr. Praneeth Wickramasinghe (*Technical Lead at Navantis Sri Lanka)* – "Fairly straight forward since it has a nice flow within a single page. Not complicated. Clear view of each section. This will help Rapid modeling using natural language. However, it can be improved to incorporate more UML diagrams. Support for more diagrams and accuracy in natural language processing. Good if user involvement can be reduced unless necessary."

4. Mr. Hasitha Dananjaya (*Senior Software Engineer at Zone24x7 Sri Lanka)* – **"**System does target and solve the problem in a hassle free and accurate way. Drawing UML Diagrams manually is always skipped by most of the developers and considered as a troublesome task. Therefore, they overlook and underestimate the value of UML diagrams. System like this will encourage developers to interact and play with such diagrams. Intuitiveness of the system is commendable. Having able to manually filter and fine tune identified entities and properties is crucial this system has catered it nicely. As far as I am concerned, the approach you have taken is sufficient in this stage. However, when it comes to reading long SOW documents you might need to fine tune. I would like to see this implemented in a way, which is not depended on visual studio. Having said that it might not make any sense in future since VS will be freely available. Extending and adding several other diagram types would be awesome. But keep in mind that when extending it, do it as plug-in, so that users can get only what they want and it would make maintainability easy. The biggest drawback is developers will forget UML.**"**

Taking about the limitations of the solution

1. The biggest limitation of the solution is all the nouns in the scenario getting identify as actors and classes. Using rule based approach it limits the output nouns to some extend but it is difficult to build perfect system that identifies only nouns related to actors and classes. Therefore, UML Generator gives opportunity to remove unwanted actors/classes and select only appropriate actors and classes for their system.
2. The users of the system have to enter each sentence according to subject-object-predicate structure to get better results.
3. Because of a bug in Weka "stringtowordvector" filter, cannot use "class" word with the input text scenario.

VIII. Conclusion

Initially, the UML Generator encountered many challenges in research and development, and had to grow throughout the process. The author has done many modifications and constructed it into what it is now. Currently the UML Generator possesses its own intelligence to extract the elements of UML and is capable of creating use case and class diagram depending on the input scenario with the customizations.

The main expectation of this project was to generate use case and class diagram from the user input text scenario, reduce the time, and cost factors of both users and system analyst. Moreover, this system gives both static and

dynamic view of a scenario by generating class diagram and use case diagram. It will help the users of the system to get a quick overview regarding the system that going to develop.

When writing the literature review it has found that earlier researchers were not used intelligence with this type of research. Therefore, author has selected a different approach with some intelligence to identify associations for the use case and class diagrams and to rate the use cases that identified using natural language processing. Moreover, author has developed his own algorithms to extract the use cases using natural language processing word chunking.

The system has completed most of the functional requirements mentioned in the system requirement specification. Recognizing multiplicities functionality for the classes is a highly important task when considering generating a class diagram. However, due to time limitation with the undergraduate research, the author has kept it as a future enhancement because it needs higher level of intelligence and higher level of accuracy with the Weka models.

With the concern of evaluation and testing carried out, system shows capability of generating use case and class diagram according to the input scenario. With the use of algorithms, which was produced by the author, the system in question will be producing adequate results in a reasonable time. However, ultimate results can prove that the algorithm is a success and could adapt many different business scenarios. Then it is straightforward to attest that system is almost a good product. By writing business decisions in words, the system has conspicuous ability to identify those by analyzing style of writing of the user.

In conclusion, in the area of automation of UML diagrams author has done a deeper analysis and improvements. With more fine-tuning this research could ultimately lead to a commercial product that could be widely used.

IX. Future Work

This is an undergraduate research and the scope was selected according to the given time period. Technology changes every day and therefore opens up the window of opportunity for continuous improvement (including new features) for this application in the future to make it even better.

A) Weka Module

- This system has identified use case and class relationships using classification. Using regression also could be a good approach.
- A Weka model to test the generated use case and class diagram.
- Increase the accuracy of identifying use case and class associations using a different algorithm.

B) XML Rules Module

- Use of Prolog or R tool could be helpful in enhancing the accuracy of the system.

C) Natural Language Processing Module

- A new algorithm to identify the actors and classes directly using Natural Language Processing without giving chance to select. This will reduce the user involvement in the system.

D) Web Application

- A help menu in website to guide the user throughout the whole process.
- Improve the usability and user experience.
- Add more diagram types as plugins to the UML Generator.

Acknowledgment

Authors of this paper acknowledge the support of all the people who provided their research papers, and those who made these papers available on the web.

References

[1] I. S. Bajwa, M. I. Siddique and M. A. Choudhary, "Rule based Production Systems for Automatic Code Generation in Java," *2006 1st International Conference on Digital Information Management*, Bangalore, 2007, pp. 300-305.

[2] Joshi, s. (2012), *Textual Requirement Analysis for UML Diagram Extraction*, 3rd ed. [ebook] Prof. Dr. S. D. Joshi, p.1 Available at: http://www.ijctee.org/files/VOLUME2ISSUE3/IJCTEE_0612_12.pdf.

[3] Bhagat, S., Kapadni, P., Kapadnis, N., Patil, D. and Baheti, M. (2012), *Class Diagram Extraction Using NLP*, 1st ed. [ebook] International Journal of electronics, Communication & Soft Computing Science & Engineering, p.1 Available at: http://www.ijecscse.org/papers/SpecialIssue/comp2/190.pdf .

[4] Webopedia.com, (2015), *What is rule-based system? A Webopedia Definition*, [online] Available at: http://www.webopedia.com/TERM/R/rule_based_system.html.

[5] Moldovan, D. and Surdeanu, M. (2003), *On the Role of Information Retrieval andInformation Extraction in Question AnsweringSystems*, 1st ed. [ebook] Dallas: pringer-Verlag Berlin.

[6] Bajwa, I. and Hyder, I. (2007), UCD-generator - a LESSA application for use case design, *2007 International Conference on Information and Emerging Technologies*, [online] 1(1), pp.1-3, Available at: http://www.researchgate.net/publication/4290733_UCD-generator_-_a_LESSA_application_for_use_case_design.

[7] Harmain H.M and Gaizauskas R., (2012), *Natural Language Processing and ObjectOriented Analysis*, 1st ed. [ebook] UK: Department of Computer Science University of Sheeld, p.1. Available at: http://www.dcs.shef.ac.uk/intranet/research/public/resmes/CS9808.pdf.

[8] P. More, *Generating UML Diagrams from Natural Language Specifications*, 1st ed. Pune: International Journal of Applied Information Systems, 2012, p. 1.

[9] D. Deeptimahanti and R. Sanyal, *Semi-automatic Generation of UML Models from Natural Language Requirements*, 1st ed. Association for Computing Machinery, 2011.