Nabil Arman* and Sari Jabbarin

Generating Use Case Models from Arabic User Requirements in a Semiautomated Approach Using a Natural Language Processing Tool

Abstract: Automated software engineering has attracted a large amount of research efforts. The use of object-oriented methods for software systems development has made it necessary to develop approaches that automate the construction of different Unified Modeling Language (UML) models in a semiautomated approach from textual user requirements. UML use case models represent an essential artifact that provides a perspective of the system under analysis or development. The development of such use case models is very crucial in an object-oriented development method. The main principles used in obtaining these models are described. A natural language processing tool is used to parse different statements of the user requirements written in Arabic to obtain lists of nouns, noun phrases, verbs, verb phrases, etc., that aid in finding potential actors and use cases. A set of steps that represent our approach for constructing a use case model are presented. Finally, the proposed approach is validated using an experiment involving a group of graduate students who are familiar with use case modeling.

Keywords: Arabic user requirements, use case model, natural language processing (NLP) tool.

DOI 10.1515/jisys-2014-0092
Received February 24, 2014

1 Introduction

Object-oriented methods are used for software systems development for the many benefits they provide, such as software reuse, reducing the software development costs, among others. Therefore, there is a need for the development of automated tools that can help in constructing different components of an object-oriented software system.

A use case diagram shows a set of use cases and actors, and their relationships. Use case diagrams address the static use case view of a system. These diagrams are especially important in organizing and modeling the behaviors of a system. This article addresses the problem of generating a use case model from user requirements, written in Arabic, in a semiautomated approach. An Arabic natural language processing tool/software, namely Stanford Parser, is used to parse different statements of the user requirements, written in Arabic, to obtain lists of nouns, noun phrases, verbs, verb phrases, etc., that aid in finding potential actors and use cases. A set of steps that represent our approach for constructing a use case model are presented.

The rest of the article is organized as follows: Section 2 presents related works; Section 3 describes the process of constructing use case models from Arabic user requirements; Section 4 presents the validation of our proposed approach; and, finally, the conclusion is presented in Section 5.

***Corresponding author: Nabil Arman,** Department of Computer Science and Engineering, Palestine Polytechnic University, P.O. Box 198, Hebron, West Bank, State of Palestine, e-mail: narman@ppu.edu
Sari Jabbarin: Department of Computer Science and Engineering, Palestine Polytechnic University, Hebron, West Bank, State of Palestine

2 Related Works

Recently, there has been great interest in automating software engineering activities. Many tools were developed to automate different activities of software systems development, such as normalizing relational database schemas, reverse engineering of relational database, and generating the corresponding entity-relationship data model [1, 2]. In addition, many CASE tools were developed to aid in drawing different diagrams of Unified Modeling Language (UML). For example, Rational Rose is an object-oriented UML software design tool intended for the visual modeling and component construction of enterprise-level software applications [12].

Rational Unified Process (RUP) is an object-oriented Web-enabled program development method. According to Rational (developers of Rational Rose and the UML), RUP is like an online mentor that provides guidelines, templates, and examples for all aspects and stages of program development. RUP and similar products, such as Object-Oriented Software Process and OPEN Process, are comprehensive software engineering tools that combine the procedural aspects of development (such as defined stages, techniques, and practices) with other components of development (such as documents, models, manuals, and code) within a unifying framework [11].

RUP establishes four phases of development, each of which is organized into a number of separate iterations that must satisfy defined criteria before the next phase is undertaken: in the inception phase, developers define the scope of the project and its business case; in the elaboration phase, developers analyze the project's needs in greater detail and define its architectural foundation; in the construction phase, developers create the application design and source code; and in the transition phase, developers deliver the system to users. RUP provides a prototype at the completion of each iteration. The product also includes process support for Java 2 Enterprise Edition (J2EE) and BEA (Web Logic) development, and supplies an HTML-based description of the unified process that an organization can customize for its own use [11].

More advanced tools were developed to automate software engineering activities that are more complicated than just aiding in drawing a UML diagram or checking its overall structure. Arman and Daghameen [3] proposed a systematic approach that generates class diagrams from textual software requirements. They presented some steps to build a matrix that was used to obtain classes and their associations to generate class diagrams. The same authors later developed a CASE tool, called SDLCCASE tool, that implemented their approach [5].

Kothari [8] proposed an approach that can extract the basic elements for generating a class diagram from user requirements written in a clear way. The NLPC (natural language processing for class) can extract classes, data members, and member functions from the given user requirements. This approach was implemented as a software tool to generate class diagrams.

Seresht and Ormandjieva [13] proposed an approach to generate use case diagrams from software requirements; however, this approach depends on other models to obtain the use case by combining two technologies: Recursive Object Model (ROM) and Expert Comparable Contextual (ECC) model. The ROM provides a formal graphical model of the text and the knowledge it carries, and the ECC can extract the stakeholder role. This approach can generate the Context Use Case Model (CUCM) by applying the knowledge included in the ECC model to identify the actors and devising rules for extracting CUCM elements.

Cayaba et al. [4] proposed an approach called computer automated use case diagram generator (CAUse), which can generate the use case diagrams from a text described by using a special language called ADD. However, this approach depends on the ADD language to generate the use case diagrams.

Mala and Uma [9] proposed an approach to extract the object-oriented elements of system requirements. This approach started by assigning parts-of-speech tags to each word in the given requirements. Thus, this approach resolved the ambiguity posed by the pronouns, the pronoun resolutions, by normalizing the text at the beginning. Thereafter, the elements of the object-oriented system, such as names of the classes, the attributes, methods and relationships between the classes, sequence of actions, and the use cases and actors, are identified by mapping the "parts-of-speech-tagged" words onto the Object Oriented Modeling Language elements using mapping rules.

An automated approach that helps a software engineer in developing formal specifications in VDM is presented in Ref. [10]. In this approach, the detection of ambiguous sentences and inconsistences in the informal specifications was a major concern. A sentence is ambiguous if more than one meaning representation is assigned to it based on syntactic and semantic processing. If this occurs, user intervention is needed to decide on the intended meaning. After this phase, the entities are determined from the nouns and noun phrases in the text sentences. Relationships are determined from the verbs in the sentences. Entities and relationships are then used to develop an entity-relationship model from which VDM data types are obtained. Another major research endeavor in automated software engineering was the work of using natural language processing to aid in object-oriented analysis [7]. The natural language processing capabilities to build a UML class diagram was used. The research approach involved two major stages. The first stage is a linguistic analysis of the text to build a semantic net. The second stage uses the semantic net to obtain the class model (and its classes, associations, attributes, etc.).

3 Constructing Use Case Models

This section describes how the actors and use cases are extracted from the user requirements written in Arabic. There is a need for an Arabic natural language processing tool such as Stanford Parser, which is used in this research to help in splitting and tokenizing the Arabic user requirements text. Once this is performed, a set of heuristics are used to construct the use case model as presented in Sections 3.1 through 3.5.

3.1 Stanford Parser

Stanford Parser is a piece of software that reads text in some language, in our case Arabic, and assigns the part of speech to each word, such as noun, verb, or adjective. This software is implemented in Java programming language. Stanford Parser was originally written by Kristina Toutanova. Since that time, Dan Klein, Christopher Manning, William Morgan, Anna Rafferty, Michel Galley, and John Bauer have improved its speed, performance, usability, and support for other languages [14].

All user requirements are processed using Stanford Parser by writing the requirements in the text area provided for that purpose, as shown in Figure 1. A set of user requirements for a system implementing ridesharing are used. The requirements are written in Arabic, and some of these requirements are used in our examples. The ridesharing system includes many requirements. Two examples are presented below:

- يقوم السائق بتسجيل الدخول الى النظام ومن ثم يستطيع الاعلان عن الرحله التي سيقوم بها ويقوم في هذه المرحله بتحديد ومتطلباتها وتشمل: وقت الرحله (الانطلاق) و المسار الذي سيسلكه اضافة الى عدد المقاعد الفارغه. كما ويستطيع حذف رحلة بعد انتهاءها او الغائها.

A translation of the example: “The driver shall be able to sign-in to the system and then he shall be able to make an advertisement about the trip he is going to make. At this stage, he provides all information related to the trip, including the time and the number of seats available. He shall also be able to delete the trip afterwards.”

- يقوم السائق بقبول الركاب او رفضهم , يستطيع ايضا تتبع المسافرين باستخدام ال GPS ان توفرت هذه الخاصيه عند الركاب في النهايه يقوم بتسجيل الخروج.

A translation of the example: “The driver shall be able to accept or reject passengers. He shall also be able to follow the passengers using a GPS if available. At the end, he shall also be able to sign out.”

In addition, Stanford Parser uses a set of tags to describe different components of a statement. Stanford Parser tokenizes the statements and uses a large number of tags. Table 1 shows the ones used in our approach.

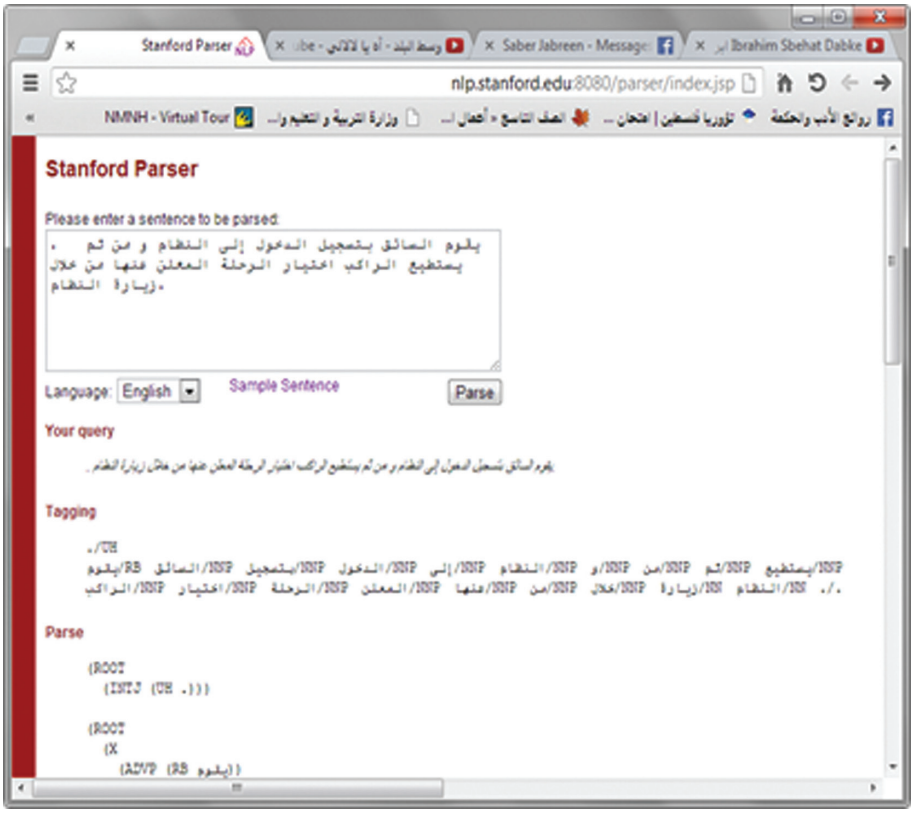

Figure 1. Stanford Parser Screen Snapshot.

Table 1. Tags' Descriptions.

Tag	Description
CC	Conjunction, coordinating
IN	Preposition or conjunction, subordinating
JJ	Adjective or numeral, ordinal
NN	Noun, common, singular or mass
NNP	Noun, proper, singular
VB	Verb, base form
VBD	Verb, past tense
VBN	Verb, past participle
VBP	Verb, present tense, not third person singular
VBZ	Verb, present tense, third person singular

3.2 Actors Identification

To identify the actors from the user requirements written in Arabic, a set of heuristics are presented. These heuristics are used to extract the actors from the tagging of the user requirements generated from Stanford Parser. These heuristics are presented as follows:

– If the statement is simple (i.e., it contains only a verb, a subject, and an object), then the actor is the main subject in the statement.
  For example, يقوم السائق بتسجيل الدخول
  A translation of the example: "The driver shall be able to sign-in to the system."
  Here, the main subject السائق and it is the actor.

Generalization: If the statement is in the form of <VBZ> <NNP> <NNP> <NNP> when using Stanford Parser, then the first NNP is the actor. To simplify referencing, the form can be written as <VBZ> <$NNP_{(1)}$> <$NNP_{(2)}$> <$NNP_{(3)}$>, where the subscripts determine the order of the NNPs.

- When there are two statements combined with a connection, then there are three cases:

  (a) The subject is the actor.

  For example, يقوم السائق بتسجيل الدخول إلى النظام و من ثم يستطيع الإعلان عن الرحلة

  A translation of the example: "The driver shall be able to sign-in to the system and then he shall be able to make an advertisement about the trip he is going to make."

  The actor is السائق.

  (b) If the subject is redundant in the second statement, then the actor does not change.

  For example, يقوم السائق بتسجيل الدخول إلى النظام و من ثم يستطيع السائق الإعلان عن الرحلة

  A translation of the example: "The driver shall be able to sign-in to the system and then the driver shall be able to make an advertisement about the trip he is going to make."

  The actor is السائق.

  (c) If the subject changes in the second statement, then this is another actor.

  For example, يقوم السائق بتسجيل الدخول إلى النظام ومن ثم يستطيع الراكب اختيار الرحلة المعلن عنها من خلال زيارة النظام.

  A translation of the example: "The driver shall be able to sign-in to the system and then the passenger shall be able to make a selection of the advertised trip."

  The actors are السائق and الراكب.

  Generalization: If the statement is in the form of <VBP> <DTNN> <NN> <DTNN> <IN> <DTNN> <DTJJ> <CC> <VBP> <DTNN> <IN> <DTNN> <WP> <VBD> <NNP> <CC> <NNP> <IN> <DT> <DTNN> <NNP> <NNP> <NNP> <PUNC> <NN> <JJ> <DTNN> <DTJJ> <CC> <DTNN> <WP> <VBP> <NN> <IN> <NN> <DTNN> <DTJJ> <PUNC> <CC> <VBD> <NN> <NN> <NN> <NN> <CC> <NN> when using Stanford Parser, then the first NN is the actor, or if there is an CC found in the statement, then we check the first NN after the CC. If this NNP is redundant after the CC, then the actor does not change; however, if the NNP has changed after the CC, then this is another actor.

- As mentioned previously, to simplify referencing, subscripts are used. Thus, the form is <$VBP_{(1)}$> <DT $NN_{(1)}$> <$NN_{(1)}$> <$DTNN_{(2)}$> <$IN_{(1)}$> <$DTNN_{(3)}$> <$DTJJ_{(1)}$> <$CC_{(1)}$> <$VBP_{(2)}$> <$DTNN_{(4)}$> <$IN_{(2)}$> <$DTNN_{(5)}$> <$WP_{(1)}$> <$VBD_{(1)}$> <$NNP_{(1)}$> <$CC_{(2)}$> <$NNP_{(2)}$> <$IN_{(3)}$> <$DT_{(1)}$> <$DTNN_{(6)}$>< <$NNP_{(3)}$> <$NNP_{(4)}$> <$NNP_{(5)}$> <PUNC> <$NN_{(2)}$> <$JJ_{(1)}$> <$DTNN_{(7)}$> <$DTJJ_{(1)}$> <$CC_{(3)}$> <$DTNN_{(8)}$> <$WP_{(2)}$> <$VBP_{(3)}$> <$N_{(3)}$> <$IN_{(4)}$> <$NN_{(4)}$> <$DTNN_{(9)}$> <$DTJJ_{(2)}$> <PUNC> <$CC_{(4)}$> <$VBD_{(2)}$> <$NN_{(5)}$> <$NN_{(6)}$> <$NN_{(7)}$> <$NN_{(8)}$> <$CC_{(5)}$> <$NN_{(9)}$>.

3.3 Use Cases Identification

To identify the use cases from the user requirements, more heuristics that can be used to extract the use cases from the user requirements are presented.

- If the statement is simple (i.e., it contains only a verb, a subject, and an object), then the use case is the main object in the statement.

  For example, يقوم السائق بتسجيل الدخول

  A translation of the example: "The driver shall be able to sign-in to the system."

  The main object is تسجيل الدخول and it is the use case.

  Generalization: If the statement is in the form of <VBZ> <NNP> <NNP> <NNP> or in the form <$VBZ_{(1)}$> <$NNP_{(1)}$> <$NNP_{(2)}$> <$NNP_{(3)}$> after using subscripts when using Stanford Parser, then the first VB is the use case.

- If the statement contains the connector (و) without any verb or actor in the second statement, then the second statement is the use case.

  For example, يستطيع الراكب الانضمام إلى الرحلة و الحجز فيها

  A translation of the example: "The passenger shall be able to join a trip and book a trip."

  In the above example, the statement contains a connector (و); thus, there are two use cases: 1- يستطيع الانضمام 2- يستطيع الحجز.

Generalization: If the statement is in the form of <VBZ> <NNP> <NNP> <IN> <NNP> <CC><NNP> <NNP> when using Stanford Parser, then

(a) The use case is the first VB with the second NNP.

(b) The use case is the first VB with the first NNP after the CC.

Again, the form can be rewritten using subscripts to tags, such as $<VBZ_{(1)}>$ $<NNP_{(1)}>$ $<NNP_{(2)}>$ $<IN_{(1)}>$ $<NNP_{(3)}>$ $<CC_{(1)}>$ $<NNP_{(4)}>$ $<NNP_{(5)}>$.

– If the statements contain the connector (أو) without any verb or actor in the statement, then the first verb in the statement with the first noun after each connector is a use case.

For example يستطيع الراكب الانضمام إلى الرحلة و الحجز فيها أو الانسحاب منها.

A translation of the example: "The passenger shall be able to join a trip and book a trip or withdraw from a trip."

In the above example, the statement contains a connector (e.g., أو); thus, there are three use cases: 1- يستطيع الانضمام 2- يستطيع الحجز 3-يستطيع الانسحاب.

Generalization: If the statement is in the form of <VBZ> <NNP> <NNP> <IN> <NNP> <CC> <NNP> <NNP> <CC> <NNP> <NNP> when using Stanford Parser, then

(a) The use case is the first VB with the second NNP.

(b) The use case is the first VB with the first NNP after the CC. The form after adding the subscript to each tag is $<VBZ_{(1)}>$ $<NNP_{(1)}>$ $<NNP_{(2)}>$ $<IN_{(1)}>$ $<NNP_{(3)}>$ $<CC_{(1)}>$ $<NNP_{(4)}>$ $<NNP_{(5)}>$ $<CC_{(2)}>$ $<NNP_{(6)}>$ $<NNP_{(7)}>$.

3.4 Use Case Model Generation

To complete the generation of the use case model, a structure that depicts the relationships among the different tokens is needed. A matrix consisting of columns with headings, which contain the potential use cases, and rows with labels, which contain the potential actors, is used. These are obtained from the heuristics explained previously. The matrix is filled by arrow symbols. An arrow means that an actor is associated with one or more particular use cases, as shown in Table 2. For example, if an arrow is shown in the cell that corresponds to the row labeled with "Use case i" and the column labeled with "Actor j," as shown in Table 2, it is concluded that Actor j is associated with Use case i.

Once the matrix is constructed, the use case model is obtained by taking an actor with all its associated use cases to generate a use case diagram. The set of all use case diagrams represent the use case model. According to the above description, this approach can be implemented easily to generate a use case model.

The proposed approach in terms of the previously presented heuristics can be presented as a high-level algorithm, as shown in Figure 2.

Applying the proposed approach described thus far to the set of user requirements mentioned previously generates the matrix presented in Table 3.

Table 2. Matrix of Potential Actors and Their Use Cases.

Potential use cases	Potential actors					
	Actor 1	Actor 2	...	Actor j	...	Actor n
Use case 1						
Use case 2						
...						
Use case i				←		
...						
Use case n						

```
GenerateUseCaseModel(Input: Arabic User Requirements; Output: Use Case Model)
Begin
  // Determining the actors and populating them in the matrix
 for each requirement in the Arabic user requirements and using Stanford Parser
  begin
    if the statement is simple then
      the actor is the main subject in the statement
   else
     if there are two statements combined with a connection then
        the subject is the actor
     else
       if the subject is redundant in the second statement then
          the actor doesn't change
       else
         if the subject changes in the second statement then
           this subject is another actor
   insert the actors in the matrix
  end for

  // Determining the use cases and populating them in the matrix
 for each requirement in the Arabic user requirements and using Stanford Parser
  begin
    if the statement is simple then
      the use case is the main object in the statement
   else
     if the statement contains the connector without any verb or actor in
     the second statement then
       the second statement is the use case
    else
      if the statements  contain the connector without any verb or actor in the statement
      then
        the first verb in the statement with the first noun after each connector  is a use
        case
   insert these use cases in the matrix
  end for

 // Constructing use case diagrams
 for each actor in the matrix
    begin
      for each use case under the current actor
          begin
            connect the actor to the use case
         end for
         add the use case diagram to the use case model
    end for
End.
```

Figure 2. Use Case Model Generation Algorithm.

3.5 Further Refinement to Use Case Model

In UML, there are three associations between use cases, namely "include," "extend," and "generalization." From the matrix, we can obtain the "include" relationship, by asking if a certain use case depends on another use case based on the meaning or semantics. Thus, we can ask if use case 1 is required by use case 2; then, if the answer is "yes," we can suggest that there is an "include" relationship, and if the answer is "no" then there is no "include" relationship between these use cases. The "extend" and "generalization" relationships can be handled similarly. These steps need human intervention; thus, our approach cannot be fully automated but is semiautomated.

4 Proposed Approach Validation

In this section, an evaluation of the proposed approach is presented. An experiment similar to the approach in Ref. [6] was designed. In the experiment, the same ridesharing case study/user requirements description

Table 3. Matrix of Potential Actors and Their Use Cases.

Potential use case	Potential actors		
	الراكب	السائق	المدير
تسجيل الدخول	←	←	←
يستطيع الاعلان		←	←
تحديد متطلبات		←	
قبول الركاب		←	
ننبع المسافرين		←	
تسجيل الخروج	←	←	←
يستطيع الانضمام	←		
يستطيع الحجز	←		
يستطيع الانسحاب	←		
يوفر تغذيه	←		
يستطيع اضافة			←
يستطيع حذف			←
تصنيف مستخدمين			←
يستعرض الرحلات	←		←

was used. The user requirements were given to five graduate students in the master of informatics program at our university. The graduate students have a good knowledge of software engineering and use case modeling. They developed some use case diagrams based on the Arabic user requirements presented. The graduate students' use case models were studied/analyzed carefully and summarized based on the average number of correct and incorrect choices of use case diagram components, namely, actors, use cases, and their communications.

The ridesharing system was implemented as a graduation project at the computer science and engineering department at our university, and the use case model presented in the graduation project report served as a benchmark for validation purposes.

Then, the students' use case diagrams and the use case diagrams developed using our proposed approach were compared with the benchmark use case model. Actors, use cases, and communications were considered equal if they were playing the same function, achieving the same aim, and presenting the same relationship between the same actors and use cases. They were considered acceptable if the actors, use cases, and communications were equal. However, some synonyms were used for some names; thus, they were also considered equal. They were called different if the actors, use cases, and communications were incorrect or add some information (but valid).

The results obtained from the group of graduate students included the number of actors determined by each student in the group. In addition, the number of use cases was also determined as shown in Table 4.

As shown in Table 4, the number of actors generated by graduate students is the same when all graduate students were presented with the same Arabic user requirements for the ridesharing system; however, the number of use cases is different among all graduate students. This means that each graduate student may understand the user requirements differently. Some of them may lack an in-depth understanding of use case

Table 4. Number of Actors and Use Cases Generated by Graduate Students.

Student no.	No. of actors	No. of incorrect actors	No. of use cases	No. of incorrect use cases	No. of correct use cases
Student 1	3	0	12	3	9
Student 2	3	0	16	0	16
Student 3	3	0	14	0	14
Student 4	3	0	18	4	14
Student 5	3	0	5	3	2

Table 5. Average Number of Use Cases and Actors Generated by Our Approach and by Graduate Students.

	Benchmark	Our proposed approach	Graduate students (average)
No. of actors	3	3	3
No. of correct use cases	16	14	11

modeling in software engineering. In rare cases, some graduate students involved in the experiment did not take their participation seriously.

When our proposed approach was applied to the same set of user requirements, the number of actors determined was 3. The number of uses cases determined was 14, as shown in Table 3. Some use cases were missing in our proposed approach due to Stanford Parser's error in parsing some Arabic user requirements. In addition, some requirements may not be written properly. The average number of use cases determined correctly by all graduate students was (9 + 16 + 14 + 14 + 2) / 5 = 11. These results are summarized in Table 4.

As presented in Table 5, our proposed approach correctly determined all actors, and 14 / 16 = 87.5% of the use cases were determined in the benchmark. The group of graduate students determined (on average) 11 / 16 = 68.75% of the use cases in the benchmark.

From this experiment, it is concluded that interaction with a human is needed in order to take into account the potential associations among use cases, namely "include," "extend," and "generalization," as explained previously. Our approach is better in identifying use cases as it does not generate incorrect use cases. In contrast, the graduate students, as humans, tend to extract incorrect use cases. As shown in Table 4, the number of incorrect use cases generated by the graduate students was 3 + 4 + 3 = 10, with an average of 10 / 5 = 2 incorrect use cases per graduate student.

5 Conclusions

The proposed approach of developing use case models is very essential in the practice of object-oriented software engineering. This approach can be implemented and incorporated into any integrated CASE (Computer Aided Software engineering) tool to aid in the process of obtaining the use case models from user requirements written in Arabic. The approach has the main advantage of dealing with the Arabic language. In addition, a set of heuristics were presented to obtain the use cases. These heuristics used the tokens produced by a natural language processing tool, namely Stanford Parser. These tokens were then used as the main components of the use case diagram, namely, the actors and the use cases. Finally, the proposed approach was validated using an experiment involving a group of graduate students who are familiar with use case modeling and a benchmark use case model obtained from a graduation project report documenting a ride-sharing system.

Acknowledgments: The authors would like to thank the Software Engineering Research Group (SERG) at Palestine Polytechnic University (PPU) for their feedback to the original idea, and Dr. Diya Abu Zeina for his help with Stanford Parser.

Bibliography

[1] N. Arman, Normalizer: a case tool to normalize relational database schemas, *Inform. Technol. J.* **5** (2006), 329–331, ISSN: 1812-5638.

[2] N. Arman, Towards E-CASE tools for software engineering, *Int. J. Adv. Corp. Learn.* **6** (2013), 16–19.

[3] N. Arman and K. Daghameen, A systematic approach for constructing static class diagrams from software requirements, in: *International Arab Conference on Information Technology (ACIT2007)*, Amman, Jordan, November 26–28, 2007.
[4] C. Cayaba, J. Rodil and N. Lim, *CAUse: Computer Automated Use Case Diagram Generator*, 2006.
[5] K. Daghameen and N. Arman, Requirements based Static class diagram constructor (SCDC) case tool, *J. Theor. Appl. Inform. Technol.* **15** (2010), 108–114.
[6] I. Diaz, L. Moreno and O. Pastor, Integrating natural language techniques in OO-method, in: *Proceedings of the 6th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing'05)*, LNCS, Springer Verlag, 2005.
[7] H. Harmain, *Building object-oriented conceptual models using natural language techniques*, PhD thesis, University of Sheffield, 2000.
[8] P. Kothari, Processing natural language requirement to extract basic elements of a class, *Int. J. Appl. Inform. Syst. (IJAIS)* **3** (2012), 39–40, ISSN: 2249-0868.
[9] G. Mala and G. Uma, *Automatic Construction of Object Oriented Design Models [UML Diagrams] from Natural Language Requirements Specification*, 2006.
[10] F. Meziane, *From English to formal specification*, PhD thesis, Department of Maths and Computer Science, University of Salford, UK, 1994.
[11] *Rational Unified Process (RUP)*, ch1, Prentice Hall, 1990, ISBN 0-13-629841-9.
[12] SearchCIO Web site, http://searchciomidmarket.techtarget.com/home/0,289692,sid183,00.html, Accessed 15 October, 2013.
[13] S. Seresht and O. Ormandjieva, Automated assistance for use cases elicitation from user requirements text, in: *11th Workshop on Requirement Engineering*, 2009.
[14] Stanford Parser, http://nlp.stanford.edu:8080/parser/index.jsp, Accessed 1 October, 2013.