An Interactive System for Arabic Software Requirements Elicitation

Hanan Elazhary

Abstract— Eliciting user requirements is one of the most critical stages in requirements engineering. Due to human factors, this is an error-prone task that affects later stages. This calls for developing automated tools to help in the elicitation process. Most research studies focus on developing automated tools for eliciting English software requirements. The problem of eliciting requirements in other languages has been greatly overlooked in the literature. This is particularly a critical problem since translating elicited requirements to English introduces additional imprecision. Thus, this paper proposes an interactive Arabic software Requirements Elicitation Assistance System (AREAS) in an attempt to tackle this problem. The system would be of a great help to billion of stakeholders in the middle east.

Index Terms— Arabic, interactive system, natural language, requirements elicitation, software engineering, software requirements engineering, user requirements.

1 Introduction

THE goal of software requirements engineering is specifying the services and constraints of software systems. To achieve this goal, it involves two main tasks: software requirements elicitation and specification [1], [2]. Software requirements are elicited from stakeholders resulting in natural-language user requirements statements describing high-level goals of software systems [3]. This is one of the most critical stages in software engineering since imprecision in the elicited user requirements causes errors in later stages. Such imprecision is at least an order of magnitude more expensive to correct when undetected until late software engineering stages [4]. Thus, focusing on improving the precision of the elicited user requirements in the first stage is one of the ambitious aims of software requirements engineering [5]. This imprecision is typically due to the lack of accuracy and/or completeness in the elicited user requirements, and most importantly due to the ambiguity of the natural languages used to express these requirements [6]. This problem becomes worse when users speak languages other than English. This is because they express their user requirements in their own languages, while programmers typically expect English requirements to be easily coded using English-like programming languages. Translation of user requirements to English introduces further ambiguities. In spite of the criticality of this problem, it has been overlooked in the literature. Thus, one of the main concerns of this paper is to tackle the problem of eliciting Arabic user requirements as a prototype. This should be greatly beneficial to billions of stakeholders in the middle east.

Analyzing natural language user requirements is mainly a manual task carried out by the software requirements engineers [7], [8]. Since user requirements usually occupy hundreds of pages that need weeks or even months to be examined, manual analysis takes a very long time and the probability of human error is very high. Consequently, using automated tools for this task would be of a great help.

Some systems have been developed to automatically detect potential imprecision in already written natural language user requirements documents through indicators such as weak verbs [9], [10], [11]. But, these systems don't assist in correcting any imprecision. Another approach in the literature attempts to avoid the introduction of imprecision while the user requirements are being written by imposing the use of natural language patterns. Most research studies in the literature have focused on developing such natural language patterns for specific domains such as database systems [12], scenarios [13], [14], and embedded systems [6].

Jain et al. [15] developed the general-purpose RAT system that imposes the use of specific natural language patterns that help users adhere to best practices in writing software requirements in different situations [7], [8], [9]. Elazhary [16] proposed a similar general-purpose system for Arabic software requirements elicitation. But, unfortunately, it is sometimes very hard for nontechnical stakeholders to stick to the suitable pattern in each situation. Besides, for the sake of consistency, these systems require predefining all the used terms in a glossary, which is cumbersome. The REAS system [17] is a semi-automated system that has been proposed for integrating these two approaches intelligently to exploit their advantages and avoid their disadvantages while eliciting software requirements. This is achieved by imposing the use of a good writing style to avoid the introduction of many types of imprecision and by interactively emulating a conversation between the requirements engineer and the user to help correct introduced imprecision. Besides, it builds a glossary of terms incrementally to help ensure consistency in the used terminology. Finally, explanations produced by the REAS system can be easily inspected by the requirements engineer to correct any missed imprecision.

This paper proposes an Arabic version of the REAS system, namely the Arabic software Requirements Elicitation Assistance System (AREAS) for eliciting Arabic software requirements. It should be noted however that Arabic is a Semitic

• *Hanan Elazhary is an associate professor at the Faculty of Computing and Information Technology, King Abdulaziz Univerity, Jeddah, KSA. She is also affiliated with the Electronics Research Institute, Cairo, Egypt. E-mail: hananelazhary@hotmail.com; helazhary@kau.edu.sa*

language which differs from Indo-European languages including English with respect to morphology (form of words), syntax (grammar) and semantics (meaning). This makes the task of processing Arabic statements challanging. Besides one-to-one translation of the REAS system to the AREAS system is impossible except in the block diagram and the modules. In other words, the AREAS system has a different set of rules and attempt to enforce a different writing style. It should be noted that the proposed version of the AREAS system is concerned with disambiguating the role of subjects, verbs, and objects in the input statements rather than *parsing* the statements. So, input statements are requested to be shortened iteratively if needed. This will be avoided in future improved versions of AREAS where more complicated statements are allowed and processed.

The paper is organized as follows: Section 2 describes the operation of the AREAS system. Section 3 provides the results and discussion. Finally, Section 4 provides the conclusions and directions for future research.

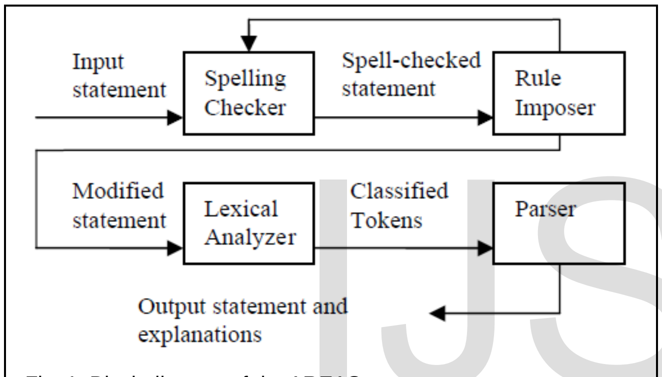

Fig. 1. Block diagram of the AREAS system.

2 The AREAS System

The purpose of the AREAS system is to help avoid the introduction of many types of imprecision while the software requirements are being written and help correct others after the software requirements are written. This is achieved by helping users adhere to some of the best practices in writing Arabic software requirements rather than imposing the use of specific natural language patterns. It enforces a set of rules that are easy enough to be followed by non-technical users. By emulating a conversation between the requirements engineer and the user, the AREAS system interactively detects and helps correct many causes of imprecision quickly and easily. Besides, it builds a glossary of used terms incrementally to help ensure consistency in the used terminology. Finally, the AREAS systems produces explanations that can be easily inspected by the software engineer for detecting and correcting missed imprecision. The current version of the AREAS system is mainly concerned with disambiguating the role of different terms as subjects, objects, or verbs. The block diagram of the AREAS system is shown in Figure 1. It is composed of four main modules: the spelling checker, the rule imposer, the lexical analyzer, and the parser. These modules are described in details in the following sub-sections. It should be noted that while the user inputs a given statement, he is prompted to use terms from the glossary (if any) to help maintain consistency in the used terminology.

2.1 The Spelling Checker

As the name implies, the purpose of the spelling checker is to spell-check the words in the input statement by referring to a lexicon. If a word in not in the lexicon, the user is prompted to correct it or add it to the lexicon. In future versions of the AREAS system, the spelling checker will be extended to be able to produce a list of suggestions for the user to select from. The spell-checked statement then enters the rule imposer.

2.2 The Rule Imposer

The rule imposer enforces a set of simple rules to help users adhere to the required writing style *while* writing the software requirements. This style is intended to help preserve the clarity of the input statements and reduce ambiguity.

Rule 1. *Write short statements.*

This rule increases the probability of having a simple statement that can be quickly and easily processed. Generally, a short statement should include no more than 25 words. In case the rule is violated, the user is prompted to shorten the statement or decompose it into two or more shorter statements.

Rule 2. *Write statements of the form: subject-verb-object or verb-subject-object only.*

Unlike English, in Arabic, statements are read from the right to the left. The Arabic language has a relatively free word order [18]. There are generally four different forms of a given declarative statement:

1. subject-verb-object; example: أحمد أكل السمكة or *Ahmed ate the fish*
2. object-verb-subject; example: السمكة أكلها أحمد or *The fish ate it Ahmed*
3. verb-subject-object; example: أكل أحمد السمكة or *Ate Ahmed the fish*
4. verb-object-subject; example: أكل السمكة أحمد or *Ate the fish Ahmed*

The two forms object-verb-subject and verb-object-subject are confusing in English and also in Arabic and so Rule 2 forbids using them. The form object-verb-subject can be detected through the accompanying pronoun (such as ها or *it* in the corresponding example above). The form verb-object-subject cannot be easily detected especially if both the subject and object are of the same gender such as in case of the statement أكلت السمكة منى or Ate the fish Mona. In this statement, it is not clear whether Mona ate the fish or whether the fish ate Mona. If a violation of this rule is detected, the user is prompted to rewrite the input statement.

Rule 3. *Two consecutive nouns are interpreted as a noun (subject or object) and a corresponding modifier.*

Though the current version of the AREAS system is concerned with disambiguating the role of different terms as subjects, objects, or verbs, modifiers are inevitable and so have to be considered.

Rule 4. *Write only active statements.*

In Arabic, a passive statement can typically take one of the following two forms:

1. verb-object; example: أكلت السمكة or *was eaten the fish*
2. object-verb; example: السمكة أكلت or *The fish was eaten*

Rule 4 imposes writing active statements only since in case of passive statements, it is not clear who or what is doing the action. Violation of this rule can be detected when an object is missing in the verb-object form (assuming verb-subject-object) or in the object-verb form (assuming subject-verb-object). If a violation of this rule is detected, the user is prompted to re-write the input statement.

Rule 5. *Write only declarative statements.*

Other possible types of statements in the Arabic language include:

1. imperative (command or request); example: كل السمكة or *eat the fish*
2. interrogative (question); example: أكل السمكة؟ or *Did he eat the fish?*
3. exclamatory (surprise or strong feelings); example: أكل السمكة! or *He ate the fish!*

These statements are not allowed since they are unnecessary in software requirements documents and would complicate the operation of the system. Violations of this rule can be easily detected when an assumed object or subject is missing. Some keywords also aid in detecting many of these statements such as the keyword هل in the statement هل أكل السمكة or *Did he eat the fish*? or the keyword ما in ما أجمل السمكة! or *What a beautiful fish!* If any of these statements is detected, the user is prompted to remove it.

Rule 6. *Do not use pronouns.*

There are many forms of declarative statements in Arabic using pronouns [19]. These include:

1. personal pronouns; example: هو أكل السمكة or *He ate the fish*
2. relative pronouns; example: السمكة التي أكلها أحمد or *The fish which Ahmed ate*
3. demonstrative pronouns; example: هذا أكل السمكة or *This ate the fish*

This rule tries to avoid referential ambiguity [7], [8] and the implicitly problem [9] since it is not clear what these words refer to. Whenever a pronoun is detected, the user is prompted to replace it.

Generally, if any of the above rules is violated and the violation is detected by the rule imposer, it prompts the user to modify it, remove it, or keep it for subsequent inspection by the requirements engineer. A modified statement reenters the spelling checker and the rule imposer for inspection. This continues until both modules detect no further problems.

The statement then enters the lexical analyzer. The lexical analyzer and the parser try to emulate a conversation between the requirements engineer and the user in an attempt to detect and help correct many causes of imprecision in each inspected input statement. The lexical analyzer also helps in building a glossary of used terms incrementally in order to help maintain consistency in the used terminology.

2.3 The Lexical Analyzer

The goal of the lexical analyzer is to consult a lexicon to generate a set of classified tokens such as nouns, verbs, modifiers, etc. The problem is that there exist many sources of ambiguity in the Arabic language [19]:

1. category ambiguity where a given word has several interpretations; example: the word كتب can mean the noun *books* or the verb *wrote*
2. homographs or words with two or more meanings; example: the word صبر can mean *patience* or *cactus plant*
3. syntactic ambiguity when there are several interpretations for the syntax of a given statement; example: رأى أحمد رجل بنظارة can mean *Ahmed saw a man wearing eye glasses* or *Ahmed saw a man using eye glasses*

In an attempt to address some of the causes of such ambiguities, the following rules apply:

Rule 7. *A word at the start of a given statement suffering from category ambiguity is a verb if followed by a noun*

Rule 8. *A word at the start of a given statement suffering from category ambiguity is a noun if followed by a verb*

In fact, Rule 2 dictates Rules 7 and 8. This is because statements are allowed to take the forms subject-verb-object or verb-subject-object only.

Rule 9. *In case of a homograph, suggestions are provided to the user for selection*

Rule 10. *Each item is given a code and added to the glossary unless already exists.*

The purpose of Rule 10 is to build a glossary of terms incrementally to help ensure consistency in the used terminology. Giving one code to two different items or different codes to the same item signals inconsistency in the used terminology and so possible ambiguity.

The statement and the classified tokens generated by the lexical analyzer are passed to the parser for specifying the role of each item (subject, object, etc). It also produces explanations that can be later inspected by the requirements engineer to detect missed ambiguities.

2.4 The Parser

The goal of the parser is to specify the role of each item in the input statement as subject, object, verb, or modifier. In case of a syntactic ambiguity, the parser provides the user with a list of possiblities to make a selection. The following rules apply:

Rule 11. *A verb at the start of a given statement is followed by a subject then an object*

Rule 12. *A noun at the start of a given statement is followed by a verb then an object*

In fact, Rule 2 dictates Rules 11 and 12. This is because statements are allowed to take the forms subject-verb-object or verb-subject-object only.

Rule 13. *Two consecutive nouns are a noun and a corresponding modifier*

In case of consulting the user, the user selections are produced as explanations that can be easily inspected by the software engineer for detecting and correcting any missed imprecision.

It should be noted that in case the user selects items from the glossary, verbs are allowed to play the role of objects as shown in the example in Section 3.

3 Results and Discussion

The AREAS system was tested against a user-description of some simple programs. An example of these programs ia as follows:

البرناميج يطلب من المستخدم إدخال عشر أرقام. هو يجمع الأرقام ويخرج المجموع

In English, these Arabic statements can be expressed as follows (with an error in the word program):

The progiram prompts the user to input 10 numbers. It adds the numbers and produces the sum.

These two statements enter the system subsequently. When the statement البرناميج يطلب من المستخدم إدخال عشر أرقام enters the sytem, since the glossary is initially empty, the user is not prompted to select items from it. The spelling checker detects an error in the spelling of the first word البرناميج so the user is prompted to correct it. The rule imposer counts the number of words forming the statement and recognizes no violation of Rule 1. However, it detects violation of Rule 2. The statement does not take the form subject-verb-object or verb-subject-object. The user is prompted to rewrite it. It is rewritten as follows:

البرنامج يطالب المستخدم. البرنامج يطلب إدخال عشر أرقام.

In English, this Arabic statement can be expressed as follows:

The program prompts the user. The program requests inputting 10 numbers.

When the statement البرنامج يطالب المستخدم enters the sytem, the spelling checker does not detect any problems. The rule imposer detects no problems. The statement enters the lexical analyzer. It classifies the word البرنامج as a noun with code 001, the word يطالب as a verb with code 002, and the word المستخدم as a noun with code 003. The statement and the classified tokens are passed to the parser for specifying the role of each item. It classifies the word البرنامج with code 001 as a subject, the word يطالب with code 002 as a verb, and the word المستخدم with code 003 as an object.

When the statement البرنامج يطلب إدخال عشر أرقام enters the sytem, the spelling checker does not detect any problems. The rule imposer detects a violation of Rule 2. The statement does not take the form subject-verb-object or verb-subject-object. The user is prompted to rewrite it. It is rewritten as follows:

البرنامج يطلب إدخال. المستخدم يدخل عشر أرقام.

In English, this Arabic statement can be expressed as follows:

The program requests inputting. The user inputs 10 numbers.

When the statement البرنامج يطلب إدخال enters the sytem, the user is prompted to use items from the glossary so the statement is input in the form of *يطلب إدخال 001* or *001 requests inputting*. The spelling checker and the rule imposer do not detect any error. The lexical analyzer classifies the word يطلب as a verb with code 004 and the word إدخال as a noun with code 005. The statement and the classified tokens are passed to the parser. It classifies the word البرنامج with code 001 as a subject, the word يطلب with code 004 as a verb, and the word إدخال with code 005 as an object.

When the statement المستخدم يدخل عشر أرقام enters the sytem, the user is prompted to use items from the glossary so the statament is input in the form of *عشر أرقام 005 003* or *003 005 ten numbers*. The spelling checker and the rule imposer do not detect any error. The lexical analyzer detects two consecutive nouns so consults the user to specify the modifier. Accordingly, it classifies the word عشر as an identifier with code 006 and the word أرقام as a noun with code 007. The statement and the classified tokens are passed to the parser. It classifies the word عشر with code 006 as an identifier to the word أرقام with code 007, which is in turn classified as an object.

When the statement هو يجمع الأرقام ويخرج المجموع enters the sytem, the user is prompted to use items from the glossary. Suppose that the user does not select any glossary item. The statement enters the spelling checker, which does not detect any error. The rule imposer detects a pronoun so the user is prompted to rewrite the statement. It is rewritten as follows:

البرنامج يجمع الأرقام ويخرج المجموع

In English, this Arabic statement can be expressed as follows:

The program adds the numbers and produces the sum.

Again, the user is prompted to use items from the glossary. The statement using glossary term 001 enters the system in the form of *يجمع الأرقام ويخرج المجموع 001* or *001 adds the numbers and produces the sum*. The spelling checker does not detect any error. The rule imposer detects a violation to Rule 2 so the user is prompted to rewrite the statement. It is rewritten as follows:

001 يجمع الأرقام. 001 يخرج المجموع

In English, this Arabic statement can be expressed as follows:

001 adds the numbers. 001 produces the sum.

When the statement 001 يجمع الأرقام enters the system, the user is prompted to use items from the glossary and so the statement enters the system in the form of *007 يجمع 001* or *001 adds 007*. The spelling checker and the rule imposer do not detect violation of any type. The lexical analyzer classifies the word يجمع as a verb with code 008. With respect to the parser, 001 acts as a subject and 007 acts as an object.

When the statement 001 يخرج المجموع enters the system, the spelling checker and the rule imposer do not detect violation of any type. The lexical analyzer classifies the word يخرج as a verb with code 009 and the word المجموع as a noun with code 010. Finally, the parser classifies the word يخرج with code 009 as averb and the word المجموع with code 010 as an object.

Accordingly, the following explanations are output by the system to be inspected by the requirements engineer:

1. البرنامج 001 يطالب 002 المستخدم 003
2. البرنامج 001 يطلب 004 إدخال 005
3. المستخدم 003 يدخل 005 عشر 006 أرقام 007
4. البرنامج 001 يجمع 008 الأرقام 007
5. البرنامج 001 يخرج 009 المجموع 010

In English, these explanations can be interpreted as follows:

1. The program 001 asks 002 the user 003
2. The program 001 requests 004 inputting 005
3. The user 003 inputs 005 ten 006 numbers 007
4. The program 001 adds 008 the numbers 007
5. The program 001 produces 009 the sum 010

Further inspection by the requirements engineer reveals that the verb 002 and 004 should take the same code.

4 Conclusion

This paper presented the proposed interactive Arabic software Requirements Elicitation Assistance System (AREAS) that is a compromise between detection and correction of imprecision while and after the natural language software requirements are written. The system is designed for Arabic software requirements to help reduce the ambiguity of the Arabic language statements and the further imprecision introduced due to the translation of these statements to English. This is especially important in the middle east where stakeholders typically speak Arabic while software engineers expect English user requirements statements.

It should be noted that the system is not intended as a translator, so not all forms of senetences are allowed. Alternatively, the system is intended to help reduce introduced ambiguities in requirements documents by inforcing a set of rules and building a glossary of terms to help maintain consistency in the used terminology. It emulates a conversation between the requirements engineer and the user to resolve and reduce several ambiguities. It produces explanations that can be easily inspected by the requirements engineer to detect any missed ambiguity.

It sould be noted also that this is the first version of the AREAS system that is designed to deal mainly with disambigating the role of verbs, subjects, and objects in a given statement. The system is being modified to deal with more complicated sources of ambiguity. Results will be reported in subsequent papers.

References

[1] I. Sommerville, *Software Engineering*. Addison Wesley, 9th edition, 2010.

[2] K. Wiegers and J. Beatty, *Software Requirements*. Microsoft Press, 3rd edition, 2013.

[3] A. Lamsweerde, R. Darimont, and E. Letier, "Managing Conflicts in Goal-Driven Requirements Engineering," *IEEE Trans. Software Engineering*, vol. 24, no. 11, pp. 908-926, 1998.

[4] S. Schach, *Object-Oriented and Classical Software Engineering*. McGraw-Hill, 8th edition, 2010.

[5] C. Rupp, "Requirements and Psychology," *IEEE Software*, vol. 19, no. 3, pp. 16-18, 2002.

[6] C. Denger, D. Berry, and E. Kamsties, "Higher Quality Requirements Specifications through Natural Language Patterns," *Proc. IEEE Conf. Software: Science, Technology and Engineering*, 2003.

[7] E. Kamsties and B. Paech, "Taming Ambiguity in Natural Language Requirements," *Proc. the 13th Int. Conf. Software & Systems Engineering and Their Applications*, 2000.

[8] E. Kamsties, D. Berry, and B. Paech, "Detecting Ambiguities in Requirements Documents Using Inspections," *Proc. the 1st Workshop on Inspection in Software Engineering*, 2001.

[9] G. Lami, "QuARS: A Tool for Analyzing Requirements," Technical Report CMU/SEI-2005-TR-014, Carnegie Mellon Software Engineering Institute, PA, USA, 2005.

[10] I. Hussain, O. Ormandjieva, and L. Kosseim, "Automatic Quality Assessment of SRS Text by Means of a Decision-Tree-Based Text Classifier," *Proc. the 7th Int. Conference on Quality Software*, 2007.

[11] B. Lal and C. Chavan, "An Optimization Approach to Analysis of Requirement Pre-Processing in Software Engineering," *Int. Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 2, pp. 311-316, 2013.

[12] A. Ohnishi, "Software Requirements Specification Database Based on Requirements Frame Model," *Proc. the 2nd International Conference on Requirements Engineering*, 1996.

[13] Y. Wang, L. Zhao, X. Wang, X. Yang, and S. Supakkul, "PLANT a Pattern Language for Transforming Scenarios into Requirements Models," *Int. J. Human-Computer Studies*, vol. 71, pp. 1026-1043, 2013.

[14] C. Ben Achour, "Guiding Scenario Authoring," *Proc. the 8th European-Japanese Conference on Information Modeling and Knowledge Bases*, 1998.

[15] P. Jain, K. Vema, A. Kass, and R. Vasquez, "Automated Review of Natural Language Requirements Documents: Generating Useful Warnings with User-extensible Glossaries Driving a Simple State Machine," *Proc. the 2nd India Software Engineering Conference*, 2009.

[16] H. Elazhary, "Translation of Software Requirements," International Journal of Scientific and Engineering Research, vol. 2, no. 5, 2011.

[17] H. Elazhary, "REAS: An Interactive Semi-Automated System for Software Requirements Elicitation Assistance," *International Journal of Engineering Science and Technology*, vol. 2, no. 5, pp. 957-961, 2010.

[18] A. Ramsay and H. Mansour, "Local Constraints on Arabic Word Order," Proc. the 5th International Conference on NLP, pp. 447–457, 2006.

[19] Y. Salem, "A Generic Framework for Arabic to English Machine Translation of Simplex Sentences Using the Role and Reference Grammar Linguistic Model," M.Sc. thesis, ITB, 2009.