ISSN 2229-5518

Translation of Software Requirements

Hanan Elazhary

Abstract—Stakeholders typically speak and express software requirements in their native languages. On the other hand, software engineers typically express software requirements in English to programmers who program using English-like programming languages. Translation of software requirements between the native languages of the stakeholders and English introduces further ambiguities. This calls for a system that simplifies translation of software requirements while minimizing ambiguities. Unfortunately, this problem has been overlooked in the literature. This paper introduces a system designed to facilitate translation of requirements between English and Arabic. The system can also facilitate the analysis of software requirements written in Arabic. This is achieved through enforcing writing software requirements statements using templates. Templates are selected such that they enforce following best practices in writing requirements documents.

Index Terms— Requirements, Software Engineering, Translation

1 INTRODUCTION

SOFTWARE requirements engineering is concerned with understanding and specifying the services and constraints of a given software system. It involves software requirements elicitation and specification [1]. Elicitation of software requirements from stakeholders typically results in user requirements, which are natural language statements that describe the high-level goals of a given software system [2]. Analysis of natural language user requirements is an important activity since imprecision in this stage causes errors in later stages. Requirements imprecision is at least an order of magnitude more expensive to correct when undetected until late software engineering stages [3]. Thus, focusing on improving the precision of the elicited user requirements in the first cycle is one of the ambitious aims of software engineering [4]. One of the main causes of imprecision is the ambiguity of natural languages used to express the user requirements [5]. To minimize ambiguity, a number of best practices in writing requirements documents have been proposed by experts [6-9]. These practices include:

- Maintain terminological consistency and clarity by restricting action and actor descriptions to terms that are clearly defined in a glossary.
- Do not use different phrases to refer to the same entity (For example, do not use Order Processing System and Order Entry System to refer to the same system).
- Avoid using phrases, such as “easy to use”, whose meaning is subjective and leads to ambiguity.
- Write each requirement as a single separate sentence.
- Write complete sentences rather than bulleted buzz phrases.
- Write complete active-voice sentences which clearly specify the actor / agent and the action.
- Write requirements sentences in a consistent fashion using a standard set of syntaxes with each syntax-type corresponding to and signaling different kinds of requirements.
- Associate a unique identifier with each requirement.

While these practices are relatively easy to state and understand, it seems fairly difficult for requirements engineers to consistently apply them throughout requirements documents with thousands of requirements. Thus, some tools in the literature have been developed to help users adhere to best practices in writing requirements documentation. This also simplifies the automatic analysis of requirements documents written in natural language and allows generating warning messages when the requirements do not conform to best practices [9].

One of the problems that have been overlooked in the literature is the problem of software requirements translation. Stakeholders typically speak their native languages, while requirements documents are typically written in English and software programs are typically developed in English-like programming languages. The problem is that the translation of software requirements from the native language of stakeholders to English introduces further ambiguities. Whenever errors are discovered in later stages of the software engineering process, suggested modifications of the software requirements result. To negotiate these modifications with the stakeholders, modified requirements need to be translated between English and the native language of the stakeholders back and forth. This can introduce more ambiguities that complicate the problem even more.

To address this problem, we suggest implementing systems that helps users adhere to best practices in writing requirements documents using different natural languages. This simplifies analyzing requirements documents in the natural language of the stakeholders. By specifying the mappings between the different developed systems, we allow translation of software requirements between different natural languages while minimizing ambiguities. In this paper, we introduce the Arabic Requirements Analysis Tool (ARAT) system that has been designed to handle software requirements in Arabic. The reason for selecting the Arabic language is that it is the official language of hundreds of millions of people in the Middle East and North Africa. It is expected that a large number of these targeted users would benefit from ARAT. We also specify mappings between our system and a similar system called Requirements Analysis Tool (RAT) [9]. RAT handles requirements written in English. Mappings simplify translation of natural language requirements between English and Arabic, while minimizing ambiguities.

The paper is organized as follows: Section 2 describes related research in the literature. Section 3 describes the RAT system and Section 4 describes the ARAT system and how the Arabic requirements are analyzed using it. Section 5 provides examples that illustrate the mappings between the English syntaxes in the RAT system and the Arabic syntaxes in the ARAT system. The examples also illustrate how translation is performed between English and Arabic requirements accordingly while minimizing ambiguities. Finally, Section 6 provides conclusions and directions for future research.

TABLE 1
A SNAPSHOT OF AN ENTITY GLOSSARY

Entity Descriptor	Explanation	Is Agent
order processing system	System for processing orders	Yes
Web server	HTTP Web Server	Yes
finance department user	User from finance department	Yes
chemical containers	Containers that store acids	No
customer standing	The status of the customer	No

TABLE 2
A SNAPSHOT OF AN ACTION GLOSSARY

Action Descriptor	Explanation
process payroll	Action for processing of payroll.
inform administrator	Action for sending e-mail notification to the administrator.
send contracts data	Action for transfer of contract data.
Display	Rendering an item on screen.

2 REALTED WORK

Many tools in the literature have been developed to automatically analyze natural language software requirements. Lami [10] and Hussain et al. [11] developed systems that can automatically detect potential imprecision in natural language software requirements through indicators such as weak verbs. But, these systems don't assist in correcting any imprecision.

Another approach in the literature attempts to avoid the introduction of imprecision while the software requirements are being written by imposing the use of natural language patterns. Some of these have focused on developing natural language patterns for specific domains such as database systems [12], scenarios [13], and embedded systems [5]. General purpose systems include Raven [14], which can analyze use cases. Jain et al. [9] 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 and can provide useful advices.

The REAS system [15] attempts to integrate these two approaches intelligently to exploit their advantages and avoid their disadvantages, but cannot help in the translation of requirements. Thus, the prposed system emulates the RAT system and uses the analogy ti simplify the translation of requirements between Arabic and English while minimizing ambiguities.

3 THE RAT SYSTEM

In this section, we describe the structure of an English requirements document according to the RAT system and discuss how the requirements document is analyzed accordingly.

3.1 User-Defined Glossaries

User-defined glossaries should be created to define valid entities and actions in the requirements document. This helps requirements engineers adhere to one of the best practices in writing requirements documents; that is using entities and actions consistently. As will be explained later in Section 3.3, the terms in the glossaries will also be used as placeholders that help in the analysis of the requirements.

The entity glossary defines all entities in the requirements document. It also specifies whether each entity is an agent that can perform actions or not. Agents and non-agents will be referred to as agents and entities respectively throughout the paper. Snapshots of an entity glossary and an action glossary are shown in Tables 1 and 2 respectively.

3.2 Best Practices Syntaxes

A set of syntaxes has been defined in the RAT system to help requirements engineers adhere to best practices in writing requirements documents as explained in Section 1. These syntaxes are as follows:

(a) The syntax *<Agent Phrase> <"shall" | "must" | "will"> <Action Phrase>* is used to express a requirement that an agent is responsible for carrying out some action. For example: The Web server must inform administrator of failed login attempts.

(b) The syntax *<Agent Phrase> <"shall" | "must" | "will"> "be able to" <Action Phrase>* is used to express a requirement that an agent should have the ability to perform an action. For example: The payroll system shall be able to deduct loan amounts from paychecks.

(c) The syntax *<Agent Phrase> <"shall" | "must" | "will"> <"allow" | "permit"> <Agent Phrase> "to" <Action Phrase>* is used to express a requirement that an agent should provide another agent with the capability to perform an action. For example: The order processing system must permit administrator to view daily transactions.

(d) The syntax *<Agent Phrase> <"shall" | "will" | "may"> <"only" | "not"> <Action Phrase> <"when" | "if"> <condition>* is used to express imposed conditions or constraints on actions performed by agents. For example: The account management system shall only close an account if the current balance is zero.

(e) The syntax *"Only" <Agent Phrase> <"may" | "maybe"> <Action Phrase>* is used to express imposed conditions on agents who may perform an action or to whom an action may be performed. For example: Only the payroll employees may access the payroll database.

(f) The syntax <*Entity Phrase* | *Agent Phrase*> *"must"* <*"always"* | *"never"* | *"not"*> <*"be"* | *"have"*> <*Value Phrase*> is used to express imposed constraints on attributes or values of attributes of entities or agents. For example: The customer standing must always be gold, silver, or bronze.

(g) The syntax <*Entity Phrase* | *Agent Phrase*> *"is"* <*"defined as"* | *"classified as"*> < *Entity Phrase*> is used to express a definition of an entity or an agent. For example: The total sales value is defined as total item value plus sales tax.

(h) The syntax <*Entity Phrase* | *Agent Phrase*> <*"is"* | *"is not"*> <*Action Phrase*> is used to express policies that should be adhered to. For example: The sales tax is computed on instate shipments.

It is clear that each category of requirements is expressed by a specific syntax type with different keywords. This simplifies understanding the intent of each requirement and its analysis as explained in Section 3.3. It should be noted that these syntaxes can be written in the format of the conditional syntax (d).

3.3 Analyzing Requirements

RAT uses a two phased approach for the analysis of the requirements document: 1) lexical analysis and 2) syntactic analysis.

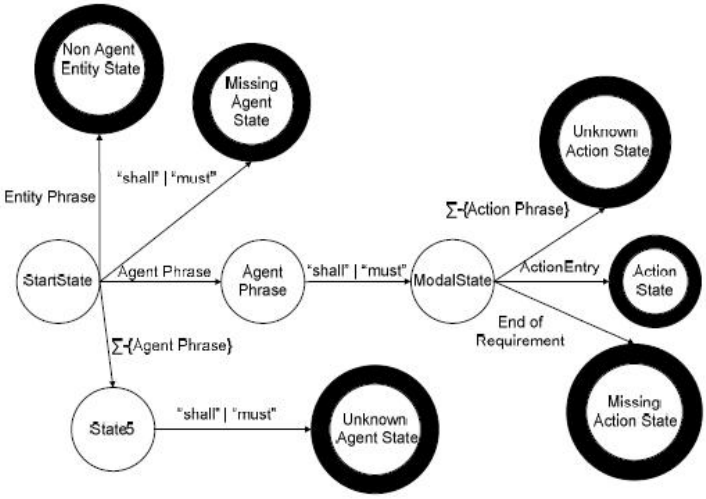

Fig. 1. A state machine for syntax type (a) of the RAT system [9].

3.3.1 Lexical analysis

A lexical analyzer breaks down a given requirement into a set of tokens: agent phrases, entity phrases, action phrases, or modal phrases formed of keywords such as "shall", "will" and "shall be able to". For example, the following statement "The SAP system shall send the vendor data to the order processing system" is tokenized into: the agent phrase "The SAP system", the modal phrase "shall", the action phrase "send", the entity phrase "the vendor data", the unknown phrase "to" and the agent phrase "the order processing system". After tokenization, the requirement is classified into one of the syntax types based on the modal phrase(s). Thus, according to the above modal phrase "shall", the requirement follows syntax type (a).

3.3.2 Syntactic analysis

The syntactic analyzer has a different sate machine to validate each syntax type. The tokenized requirement is run through the corresponding state machine. A requirement is treated as syntactically correct when the state machine successfully transitions from the start state to a valid final state. In the above example, the state machine shown in Figure 1 is used. The token stream for the requirement will end up in "Action State" and so will be treated as a valid requirement.

For every error state, there is a predefined warning message that is displayed to the user. The statement of each warning message explains why the requirement deviates from best practices in writing requirements documents. Table 3 shows the warning messages corresponding to different error states in the above state machine. For example, the statement "shall display error messages in new window" halts in "Missing Agent State" since it lacks an agent phrase in the beginning. The generated warning message is "This requirement lacks an agent before 'shall'. It can be confusing to leave the agent implicit."

TABLE 3
ERROR STATES AND THE CORRESPONDING WARNING MESSAGES

Error State	Warning Message
Missing Agent State	*This requirement lacks an agent before '<variable at which error occurs>'. It can be confusing to leave the agent implicit.*
Unknown Action State	*This requirement contains '<variable at which error occurs>' where an action is expected, but '<variable at which fault occurs>' is not in the action glossary.*
Unknown Agent State	*This requirement contains '<variable at which error occurs>' where an agent is expected, but '<variable at which fault occurs>' is not in the entity glossary.*
Non Agent Entity State	*This requirement contains '<variable at which error occurs>' where an agent is expected. '<variable at which error occurs>' is in the entity glossary but is not designated as an agent.*
Missing Action State	*This requirement lacks an action before '<variable at which error occurs>'. It can be confusing to leave the action implicit.*

3.3.3 Early Deployment Results

Eleven real industrial software projects have been used to assess the tool. They used RAT to make changes to the requirements based on the warning messages generated by RAT. This resulted in:

- 10-30% reduction in time required to transform notes taken in interview sessions to well formed requirements.
- 30-50% reduction in time needed to review requirements.
- 5% estimated reduction in overall budget, due to expected reduction in requirements defects and associated reduction in rework.

4 THE ARAT SYSTEM

The ARAT system is an Arabic version of the RAT system. The advantage of this system is twofold. First, it has similar advantages as the RAT system but with respect to Arabic requirement documents. In other words, it helps requirements engineer adhere to best practices in writing Arabic requirements documents and simplifies the analysis of these documents. Second, due to the analogy between both systems, it allows the translation of software requirements between Arabic and English while resolving many ambiguities. Thus, an Arabic syntax (and a corresponding state machine) in the ARAT system has been designed corresponding to each English syntax (and its corresponding state machine) in the RAT system. While reading the Arabic syntaxes, it should be noted that:

- Arabic statements and thus Arabic syntaxes are written from the right to the left.
- Some syntaxes in RAT are decomposed into two syntaxes in ARAT due to the nature of the Arabic language.
- Some model phrases in RAT are represented by two modal phrases in the corresponding syntax in ARAT due to the nature of the Arabic language. For example "be able to" is represented by "يمكنه أن" and"يمكنها أن" for masculine and feminine respectively.
- The meanings of the Arabic modal phrases in these syntaxes are provided in the modal phrases dictionary shown in Table 4.
- The arrangements of the components of a given syntax in ARAT may be different from that of the corresponding syntax in RAT due to the nature of the Arabic language.

The ARAT Arabic syntaxes (corresponding to the RAT English syntaxes described in Section 3.2) are as follows:

TABLE 4
THE MODAL PHRASES DICTIONARY

Modal phrases in RAT	Corresponding modal phrases in ARAT
shall | will	سوف
shall | will not	لن
must	يجب أن
must not	لا يجب أن
may | maybe	يمكن أن
may not | may not be	لا يمكن أن
is	
is not	لا
be able to	يمكنه أن / يمكنها أن
allow | permit	يمكن / تمكن
to	أن
only	فقط
when | if	لو / عندما / حين
always	دائما
never	مطلقا لا
be	يكون / تكون
have	يكون له / يكون لها
defined as	تعريفه / تعريفها
classified as	تصنيفه / تصنيفها

(a) <Action Phrase> <"يجب أن" / "سوف"> <Agent Phrase>
(b) <Action Phrase> <"يمكنها أن"> <Action Phrase> <"يمكنه أن" / "يجب أن" / "سوف"> <Agent Phrase>
(c) <Action Phrase> <"أن"> <Agent Phrase> <"تمكن" / "يمكن" / "يجب أن" / "سوف"> <Agent Phrase>.
(d) <Action Phrase> <"لو" / "عندما" / "حين"> <condition> <Action Phrase> <"لن" / "لا يمكن أن"> <Agent Phrase> <"فقط"> <Agent Phrase> <"يمكن أن" / "سوف">
(e) <Action Phrase> <"يمكن أن"> <Agent Phrase> <"فقط">
(f) <Action Phrase> <"لو" / "عندما" / "حين"> <condition> <Action Phrase> <"لا" / "دائما"> <Entity Phrase | Agent Phrase> <"مطلقا لا">
(g) <Value Phrase> <"يكون لها" / "يكون له" / "تكون" / "يكون"> <"يجب أن"> <Entity Phrase | Agent Phrase> <"تعريفها" / "تعريفه"> <Entity Phrase> <"تصنيفها" / "تصنيفه">
(h) <Action Phrase> <Entity Phrase | Agent Phrase> <Action Phrase> <"لا"> <Entity Phrase | Agent Phrase>

As an example of analyzing Arabic requirements in ARAT, consider the following Arabic statement "نظام ساب سوف يرسل بيانات التاجر إلى نظام تدوير الطلبيات". This statement is tokenized into: the agent phrase "نظام ساب", the modal phrase "سوف", the action phrase "يرسل", the entity phrase "بيانات التاجر", the unknown phrase "إلى", and the agent phrase "نظام تدوير الطلبيات". According to the above modal phrase "سوف", the requirement follows syntax type (a). When this token stream is run through the corresponding state machine, it ends up in "Action State" and so is treated as a valid requirement.

In the next section, we provide examples that illustrate the mappings between the English syntaxes in the RAT system and the Arabic syntaxes in the ARAT system. The examples also illustrate how translation is performed between English and Arabic requirements accordingly while resolving many ambiguities.

5 TRANSLATION BETWEEN ARABIC AND ENGLISH REQUIREMENTS

Translating a given statement between RAT and ARAT systems is done by following the following steps:

- Break down the statement into a set of tokens.
- Specify the corresponding syntax.
- Translate each modal phrase according to the modal phrases dictionary shown in table 4.
- Interpret the unknown phrases and rearrange the statement according to the corresponding goal syntax (RAT syntax if translating from Arabic to English and ARAT syntax if translating from English to Arabic).
- The rest is left to the software engineer to resolve.

This is illustrated by few examples. The first example involves translating Arabic statements to English. Because the reader expects to read mainly English, the rest of the examples will involve translating English statements to Arabic.

5.1 Example 1

Consider the following Arabic statement: "نظام ساب سوف يرسل بيانات التاجر إلى نظام تدوير الطلبيات". This statement is tokenized into: the agent phrase "نظام ساب", the modal phrase

ISSN 2229-5518

"سوف", the action phrase "يرسل", the entity phrase "بيانات التاجر", the unknown phrase "إلى", and the agent phrase "نظام تدوير الطلبيات" (remember that the Arabic statements are written from the right to the left). According to the above modal phrase, the requirement follows syntax type (a). The unknown phrases are interpreted as extensions of the action phrase. According to the modal phrases dictionary, this modal phrase is translated into "shall | will". The statement is then rearranged as follows:

- *"نظام ساب" "shall | will" "يرسل" "بيانات التاجر إلى نظام تدوير الطلبيات".*

The software engineer can then translate "نظام ساب" to "The SAP system", "يرسل" to "send", and "بيانات التاجر إلى نظام تدوير الطلبيات" to "the vendor data to the order processing system" without/with minimal ambiguity. Note that the words "shall" and "will" have the same meaning. The translated statement is:

- *"The SAP system shall | will send the vendor data to the order processing system".*

5.2 Example 2

Consider the following English statement: "The Web server must inform administrator of failed login attempts". This statement is tokenized into: the agent phrase "The Web server", the modal phrase "must", the action phrase "inform administrator", the unknown phrase "of", and the entity phrase "failed login attempts". According to the above modal phrase, the requirement follows syntax type (a). The unknown phrases are interpreted as extensions of the action phrase. According to the modal phrases dictionary, the modal phrase "must" is translated into "يجب أن". The statement is then rearranged as follows (remember that the Arabic statements are written from the right to the left):

- *"يجب أن" "The Web server" "inform administrator of failed login attempts"*

The software engineer can then translate "The Web server" to "خادم الويب", "inform administrator of failed login attempts" to "يخبر المسؤول عن محاولات فاشلة لتسجيل الدخول" without/with minimal ambiguity. The translated statement is:

- *"خادم الويب يجب أن يخبر المسؤول عن محاولات فاشلة لتسجيل الدخول"*

5.3 Example 3

Consider the following English statement: "The payroll system shall be able to deduct loan amounts from paychecks". This statement is tokenized into: the agent phrase "The payroll system", the modal phrase "shall", the modal phrase "be able to", the action phrase "deduct loan amounts", the unknown phrase "from", and the entity phrase "paychecks". According to the above modal phrases, the requirement follows syntax type (b). The unknown phrases are interpreted as extensions of the action phrase. According to the modal phrases dictionary, the modal phrase "shall" is translated into "سوف" and the modal phrase "be able to" is translated into "يمكنه أن | يمكنها". The statement is then rearranged as follows:

- *"سوف" "يمكنه أن / يمكنها أن" "The payroll system" "deduct loan amounts from paychecks"*

The software engineer can then translate "The payroll system" to "نظام جداول الرواتب" and "deduct loan amounts from paychecks" to "يقتطع مبالغ القروض من شيكات المرتبات" without/with minimal ambiguity. Since "نظام الرواتب" is masculine in Arabic, the software engineer selects "يمكنه أن" rather than "يمكنها أن". The translated statement is:

- *"نظام جداول الرواتب سوف يمكنه أن يقتطع مبالغ القروض من شيكات المرتبات"*

5.4 Example 4

Consider the following English statement: "The order processing system must permit administrator to view daily transactions". This statement is tokenized into: the agent phrase "The order processing system", the modal phrase "must", the modal phrase "permit", and agent phrase "administrator", the modal phrase "to", and the action phrase "view daily transactions". According to the above modal phrases, the requirement follows syntax type (c). According to the modal phrases dictionary, the modal phrase "must" is translated into "يجب أن", the modal phrase "permit" is translated into "يمكن | تمكن", and the modal phrase "to" is translated into "أن". The statement is then rearranged as follows:

- *"يجب أن" "يمكن / تمكن" "The order processing system" "administrator" "أن" "view daily transactions"*

The software engineer can then translate "The order processing system" to "نظام معالجة الطلبات", "administrator" to "المسؤول", and "view daily transactions" to "يرى المعاملات اليومية" without/with minimal ambiguity. Since "المسؤول" is masculine in Arabic, the software engineer selects "يمكن" rather than "تمكن". The translated statement is:

- *"نظام معالجة الطلبات يجب أن يمكن المسؤول أن يرى المعاملات اليومية"*

5.5 Example 5

Consider the following English statement: "The account management system shall only close an account if the current balance is zero". This statement is tokenized into: the agent phrase "The account management system", the modal phrase "shall", the modal phrase "only", the action phrase "close an account", the modal phrase "if", and the unknown phrase "the current balance is zero". According to the above modal phrases, the requirement follows syntax type (d). The unknown phrase is interpreted as a condition. According to the modal phrases dictionary, the modal phrase "shall" is translated into "سوف", the modal phrase "only" is translated into "فقط ", and the modal phrase "if" is translated into "لو | عندما | حين". The statement is then rearranged as follows:

- *"سوف" "فقط" "The account management system" "close an account" "لو / عندما / حين" "the current balance is zero"*

The software engineer can then translate "The account management system" to "نظام إدارة الحسابات", "close an account" to "يغلق حساب", and "the current balance is zero" to "الرصيد الحالي صفر" without/with minimal ambiguity. Note that the words "لو", "عندما", and "حين" have the same meaning. The translated statement is:

- *"نظام إدارة الحسابات فقط سوف يغلق حساب لو / عندما / حين الرصيد الحالي صفر"*

5.6 Example 6

Consider the following English statement: "Only the payroll

ISSN 2229-5518

employees may access the payroll database". This statement is tokenized into: the modal phrase "only", the agent phrase "the payroll employees", the modal phrase "may", and the action phrase "access the payroll database". According to the above modal phrases, the requirement follows syntax type (e). According to the modal phrases dictionary, the modal phrase "only" is translated into "فقط" and the modal phrase "may" is translated into "يمكن أن". The statement is then rearranged as follows:

- *"فقط" "the payroll employees" "يمكن أن" "access the payroll database"*

The software engineer can then translate "the payroll employees" to "موظفي جداول الرواتب" and "access the payroll database" to "يصلوا إلى قاعدة بيانات جداول الرواتب" without/with minimal ambiguity. The translated statement is:

- *"فقط موظفي جداول الرواتب يمكن أن يصلوا إلى قاعدة بيانات جداول الرواتب"*

5.7 Example 7

Consider the following English statement: "The customer standing must always be gold, silver, or bronze". This statement is tokenized into: the entity phrase "The customer standing", the modal phrase "must", the modal phrase "always", the modal phrase "be", and the unknown phrase "gold, silver, or bronze". According to the above modal phrases, the requirement follows syntax type (f). The unknown phrase is interpreted as a value phrase. According to the modal phrases dictionary, the modal phrase "must" is translated into "يجب أن", the modal phrase "always" is translated into "دائما", and the modal phrase "be" is translated into "يكون | تكون". The statement is then rearranged as follows:

- *"The customer standing" "دائما" "يجب أن" "يكون | تكون" "gold, silver, or bronze"*

The software engineer can then translate "The customer standing" to "موقف العميل" and "gold, silver, or bronze" to "ذهب، فضة، أو برونز" without/with minimal ambiguity. Since "موقف العميل" is masculine in Arabic, the software engineer selects "يكون" rather than "تكون". The translated statement is:

- "موقف العميل دائما يجب أن يكون ذهب، فضة، أو برونز"

5.8 Example 8

Consider the following English statement: "The total sales value is defined as total item value plus sales tax". This statement is tokenized into: the entity phrase "The total sales value", the modal phrase "is", the modal phrase "defined as", and the entity phrase "total item value plus sales tax". According to the above modal phrases, the requirement follows syntax type (g). According to the modal phrases dictionary, the modal phrase "is" is translated into "", and the modal phrase "defined as" is translated into "تعريفه | تعريفها". The statement is then rearranged as follows:

- "تعريفه | تعريفها" "The total sales value" "total item value plus sales tax"

The software engineer can then translate "The total sales value" to "إجمالي قيمة المبيعات" and "total item value plus sales tax" to "إجمالي قيمة البنود وضريبة المبيعات" without/with minimal ambiguity. Since "إجمالي قيمة المبيعات" is masculine in Arabic, the software engineer selects "تعريفه" rather than "تعريفها". The translated statement is:

- "إجمالي قيمة المبيعات تعريفه إجمالي قيمة البنود وضريبة المبيعات"

5.9 Example 9

Consider the following English statement: "The sales tax is computed on instate shipments". This statement is tokenized into: the entity phrase "The sales tax", the modal phrase "is", the action phrase "computed on instate shipments". According to the above modal phrase, the requirement follows syntax type (h). According to the modal phrases dictionary, the modal phrase "is" is translated into "". The statement is then rearranged as follows:

- "computed on instate shipments" "The sales tax"

The software engineer can then translate "The sales tax" to "ضريبة المبيعات" and "computed on instate shipments" to "تحسب على الشحنات المحلية" without/with minimal ambiguity. The translated statement is:

- "ضريبة المبيعات تحسب على الشحنات المحلية"

5.10 Results

From the above examples, it is clear that many ambiguities have been resolved while translating software requirements between English and Arabic. These include specifying the agents, the entities, the actions, who performs each action, and to whom an action is performed. Besides, the intent of each statement is well understood. Few ambiguities are left to the software engineer to resolve, which dramatically simplifies the work of the software engineer.

6 CONCLUSION

Stakeholders typically speak and express software requirements in their native language. On the other hand, software engineers typically express software requirements in English to programmers who write program using English-like programming languages. This requires translating requirements between the native language of the stakeholders and English back and forth. This introduces further ambiguities, which affects the length of the whole software process. Unfortunately, this problem has been overlooked in the literature. To tackle this problem, this paper introduces the ARAT system. The ARAT system is analogous to the RAT system [9] that uses templates to express English software requirements. As illustrated by many examples, specifying mappings between both systems facilitate translation of requirements between English and Arabic while resolving many ambiguities such as specifying agents and their actions. Since the RAT system uses templates that enforce following best practices in writing requirements documents in English, the ARAT system achieves the same while writing requirements documents in Arabic. Similarly, since the RAT system has been designed to facilitate the analysis of English software requirements, the ARAT system facilitates the analysis of Arabic software requirements. As future work, the tool should be tested against more requirements documents. Observings and suggestions should be taken into consideration in future versions of the system. We encourage

researchers to develop similar systems in their native languages with the aim of facilitating communication between stakeholders and software engineers using different languages.

REFERENCES

[1] I. Sommerville, *Software Engineering*. Addison Wesley, 8th edition, 2006.
[2] A. Lamsweerde, R. Darimont, and E. Letier, "Managing Conflicts in Goal-Driven Requirements Engineering," *IEEE Transactions on Software Engineering*, vol. 24, no. 11, pp. 908-926, 1998.
[3] S. Schach, *Object-Oriented and Classical Software Engineering*. McGraw-Hill, 7th edition, 2006.
[4] C. Rupp, "Requirements and Psychology," *IEEE Software*, vol. 19, no. 3, pp.16-18, 2002.
[5] C. Denger, D. Berry, and E. Kamsties, "Higher Quality Requirements Specifications through Natural Language Patterns," *Proc. IEEE SwSTE'03*, 2003.
[6] K. Wiegers, *Software Requirements*. Microsoft Press, 2003.
[7] R. Young, *Effective Requirements Practices*. Addison-Wesley Longman Publishing Co., 2000.
[8] "IEEE Recommended Practice for Software Requirements Specifications," IEEE/ANSI Standard 830-1998, Institute of Electrical and Electronics Engineers, 1998.
[9] 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. Second India Software Engineering Conference*, 2009.
[10] G. Lami, "QuARS: A Tool for Analyzing Requirements," Technical Report CMU/SEI-2005-TR-014, Carnegie Mellon Software, Engineering Institute, PA, USA, 2005.
[11] 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 International Conference on Quality Software*, 2007.
[12] A. Ohnishi, "Software Requirements Specification Database Based on Requirements Frame Model," *Proc. the 2nd International Conference on Requirements Engineering*, 1996.
[13] C. Ben Achour, "Guiding Scenario Authoring, *Proc. the 8th European-Japanese Conference on Information Modeling and Knowledge Bases*, 1998.
[14] Raven Software, www.ravensoft.com.
[15] H. Elazhary, "REAS: An Interactive Semi-Automated System for Software Requirements Elicitation Assistance," Int. Journal of Engineering Science and Technology, vol. 2, no. 5, pp. 958-962, 2010.