PROMIRAR: Tool for Identifying and Managing Implicit Requirements in SRS Documents

Onyeka Emebo, Daramola Olawande, Ayo Charles

***Abstract*— Implicit requirements (IMRs) in software requirements specifications (SRS) are subtle and need to be identified as users may not provide all information upfront. It is found that successful functioning of a software crucially depends on addressing its IMRs. This work presents a novel system called PROMIRAR with an integrated framework of Natural Language Processing, Ontology and Analogy based Reasoning for managing Implicit Requirements. It automates early identification and management of IMRs and is found helpful in targeted application domain. We present the PROMIRAR system with its architecture, demo and evaluation.**

***Index Terms*— analogy-based reasoning, implicit requirement, natural language processing, ontology, requirement engineering**

I. Introduction

State-of-the-art: IMR management deals with identification and handling of implicit requirements. Studies such as [4, 5, 12] made use of ontology-based approaches and analogy-based reasoning for IMR identification. [7, 13] managed implicit requirements by addressing implicit knowledge. These systems lack the simulation of human reasoning, e.g., a human software engineer can identify IMRs from a software requirements specifications (SRS) document, distinguish them from explicit requirements and manage them further.

PROMIRAR's Novelty: We propose a system that embodies Analogy based reasoning (ABR) for IMR management in SRS document. ABR facilitates the reuse of previously documented requirements specifications in the detection of new IMRs.to simulate human reasoning. We find ontology imperative here as facilitates formalized semantic description of relevant domain knowledge for IMRs. Natural Language Processing entails analyzing text to extract useful information [8] and is thus significant in SRS analysis to understand similarities, identify a basis for analogy and discover knowledge for IMRs. Our proposed system known as PROMIRAR (PROduct for Managing Implicit Requirements using Analogy-based Reasoning) shows significant improvement over the state-of-the-art as evaluation by software engineers, shows that it enhances software development by augmenting implementation time and reducing software bugs.

Layout of the Paper: The rest of this paper is organized as follows. Section 2 describes related works while Section 3 discusses the system architecture of PROMIRAR. Section 4 provides the system demo. Evaluation details appear in Section 5. Section 6 gives conclusions and ongoing work.

Manuscript received June 04, 2018; revised June 15, 2018. This work was supported in part by the U.S. Department of State under Fulbright Grant.

Onyeka Emebo is with Covenant University, Department of Computer and Information Sciences, Ota, Nigeria (phone: +234 803-687-8407; e-mail: onye.emebo@covenantuniversity.edu.ng).

Daramola Olawande is with Cape Peninsula University of Technology, Department of Information Technology, Cape Town, South Africa. (e-mail: daramolaj@cput.ac.za).

Ayo Charles is with the Department of Computer and Information Sciences, Covenant University, Ota, Nigeria (e-mail: charles.ayo@covenantuniversity.edu.ng).

II. Related Work

A number of researchers have suggested numerous ways for IMR identification. While several have developed tools, others have given conceptual and theoretical frameworks and other such as software engineers, requirement engineers have taken on an investigative approach to get real life views on the reality of the specified theories, ideologies and concepts.

A two part research is conducted by [3] targeted at ascertaining the impacts of explicit and tacit knowledge conveyed throughout the software development process. A method to authenticate the spectrum of tacit knowledge in software development is used in the first phase and a conceptual framework of a model for tacit to explicit knowledge transfers is part of the second phase.

In MaTREx [14], a literature review on the usefulness of implicit knowledge for requirement engineering is given. Systems such as NAI, SR-elicitor and ARUgen were reviewed. Their focus on is on presenting such developing techniques and tools that enhances requirements information management via non-provenance requirements, determining the existence of tacit knowledge from tracing of presuppositions, automatic trace recovery, etc.

A few studies has covered Requirements reuse for the detection and management of IMRs.

In [12], a system that uses semantic case-based reasoning for managing IMRs is proposed. A tool was modelled, which aids in the management of IMRs by making use of analogy-based requirements reuse of earlier known IMRs is further presented. This approach guarantees the detection, organized documentation, right prioritization, and development of IMRs, which overall improves the attainment of software development.

Authors in [17] presented a model that computes matches between requirements specifications in order to support their analogical reuse. A model based on the concept of semantic modeling abstractions with generalization, attribution and classification was formulated.

Based on this study of the literature, our PROMIRAR system is unique as a result of the fact that it brings together ABR with natural language processing and ontology for early identification and management of IMRs. It also outperforms existing systems as evident from the experiments conducted.

III. PROMIRAR SYSTEM ARCHITECTURE

In this section we briefly outline the architecture of the PROMIRAR system. Figure 1 depicts the PROMIRAR pipeline with its core modules.

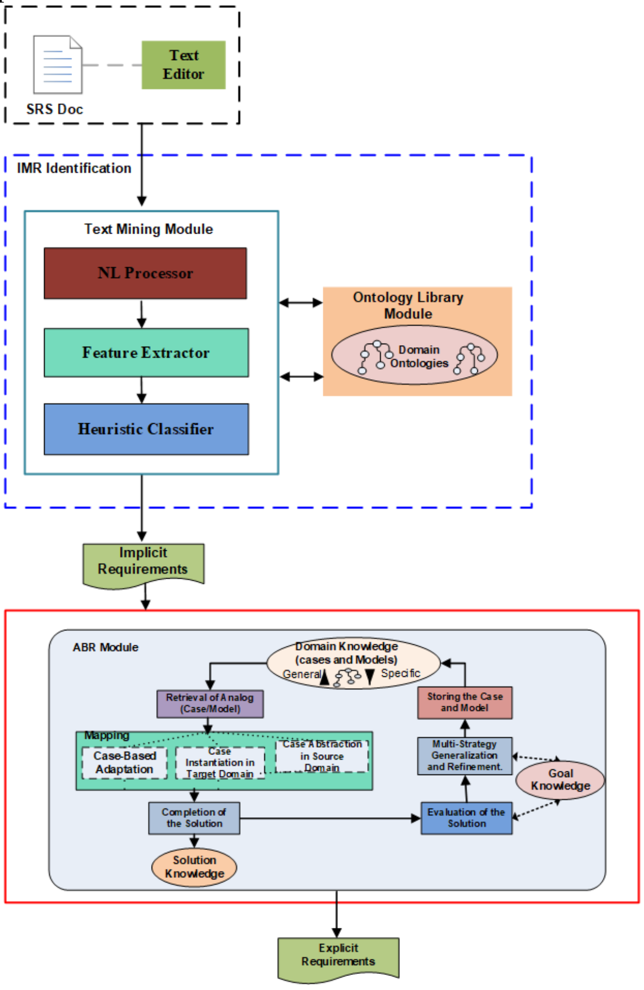


Fig 1: The PROMIRAR Pipeline

A. *Data Input*

The input into PROMIRAR is a preprocessed Software Requirements Specification (SRS) document. Preprocessing entails a manual method of extracting boundary sentences from the requirements document and additionally replacing tables, images and figures in the correspondent written format.

B. *NL Processor*

The NL processor module enables the handling of natural language requirements for the process that enables the feature extractor. The essential natural language processing tasks fulfilled in this architecture are as follows: i) selection of sentence, ii) Word Tokenization iii) tagging of Parts of speech (POS) iv) detection of entity v) and Parsing. The various NLP operations were implemented using Apache OpenNLP library. The text processing functionality of PROMIRAR that is a part of its NL Processor is illustrated in Figure 2. Raw text is input to this module from requirements documents. It conducts sentence segmentation to output strings, subject to tokenization. The tokenized sentences undergo POS (part of speech) tagging. These POS tagged sentences are then subject to entity detection. This gives chunked sentences as a list of trees which undergo relation detection. The ontology library module plays a very important role in identifying these entities and relations, using the ontology structure **O** defined as **O**=$\{C, R, A^{o}\}$ of concepts, relations and axioms respectively.

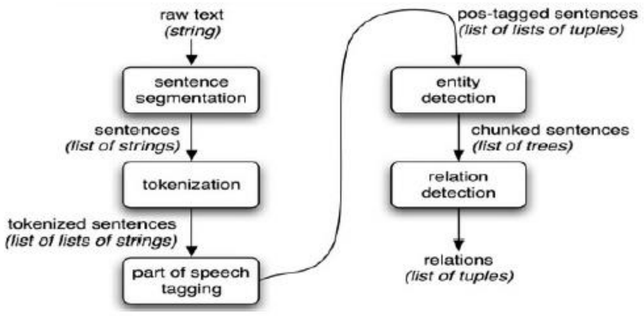


Fig 2: Text processing in NL Processor

C. *Ontology Library*

The Ontology Library (OL) module form the PROMIRAR backbone, serving as the knowledge representation for domain ontologies (for specific purposes / general business rules). Java Protégé 4.1 ontology API was used to build the ontology library. A part of a Course Management System (CMS) domain ontology imported is shown in Figure 3. This constitutes the ontograph of the steps required for conducting registration.

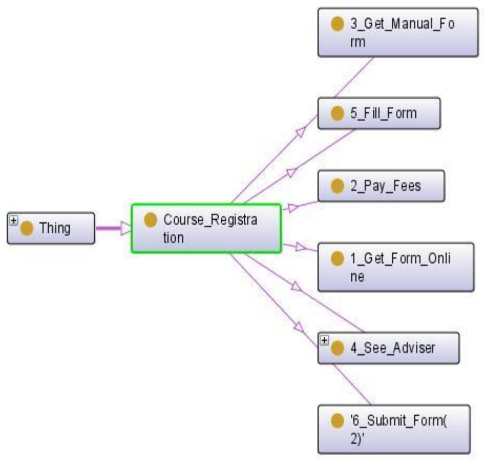


Fig 3: Ontograph of Steps Needed for Registration

D. *Feature Extractor*

This provides the essential rules for classifying possible sources of IMR in a requirement document. Some of the characteristic features that could possibly make a natural language text implicit are outlined below as documented in literature [6, 9, 10, 11, 20]. Ambiguity for instance lexical and structural ambiguity have the following features i) Existence of vague phrases and words such as "in excessive magnitude", ii) imprecise verbs such as "administered", or "excluded", iii) Occurrence of weak phrases for example "typically", "commonly" and iv) Incomplete knowledge.

E. *Heuristic Classifier*

This is responsible for classifying the actual requirements based on the intermediate outputs of the previous modules, thus helping to identify the IMRs which are the ultimate outputs of the PROMIRAR tool. It follows the pipeline of the previous modules and is the final module to help conduct the classification.

F. *Analogy-based Reasoner*

The knowledge reuse capability of the framework is facilitated by the ABR component according to maiden [19]. The component comprise of three type of knowledge (domain, solution and goal), which have been reflected in the creation of the Implicit Requirements Model (IRMM).

In order to manage IMR, a reuse-based IRMM is outlined below. This formal representation is an extension of the formalisation presented in [11].

IRMM = < D, S, G, O, R_{id}, RQ_i, IMR_{id}, IMR_i > where D is the software project domain description; S depicts the solution approach the software project implemented; G depicts the system's goal under development; O depicts the Ontology domain of Requirement R; R_{id} is a description of the distinct id of the requirement; and RQ_i is a description of the requirement statement symbolized by R_{id}; IMR_{id} describes the distinct id of the implicit requirements related with R_{id}; IMR_i depicts the implicit aspects related with the requirement RQ_i symbolized as R_{id}.

The objective of the IRMM is to offer a uniform structure for describing requirements such that it will be possible to establish a basis for analogy reasoning. A case-based representation of requirements will classify the known parts of IRMM as problem specification of a case at hand, while the unknown part will constitute the solution part. From our IRMM, the set {D, S, G} represent the domain, solution and goal parts of both the source and target project.

An example of a network representing the structural isomorphism of an analogical match that exist between a University Smart City Parking System and a Course Management System is shown in Figure 4. These two domains are case projects used in this study (domain objects are denoted in oval shapes, domain terms are represented using rectangles and lines). The potential reuse that can be done from this analogy is at the functional and structural parts for example the processes (e.g., "course placement" and "sensor car park"), the data stores (e.g., "course place" and "sensored parking space") and finally the external agents ("student" and "driver"). Even though the two systems are in dissimilar domains, the two of them share substantial features (e.g., reservations, waiting lists, places) that aids analogical understanding and recognition.

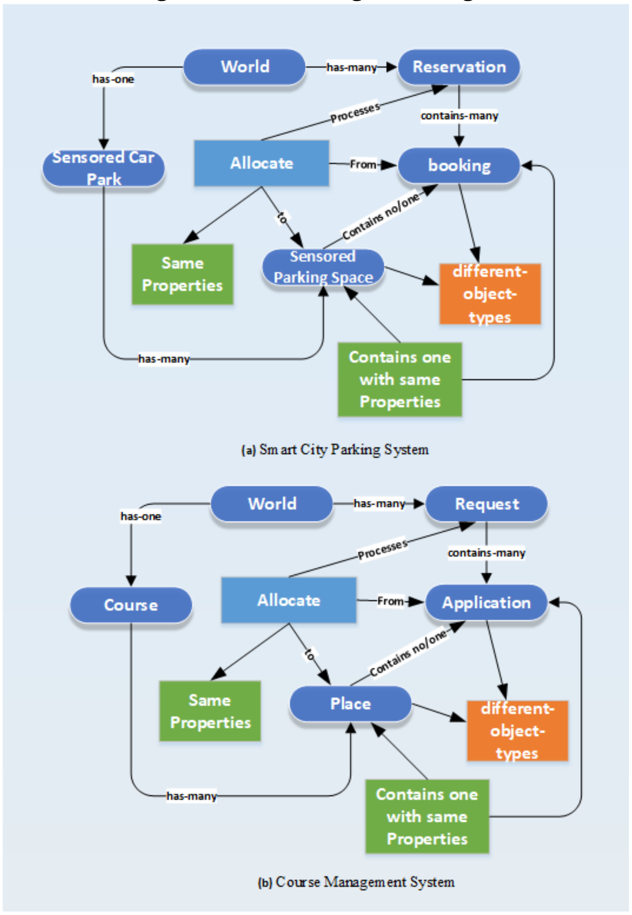
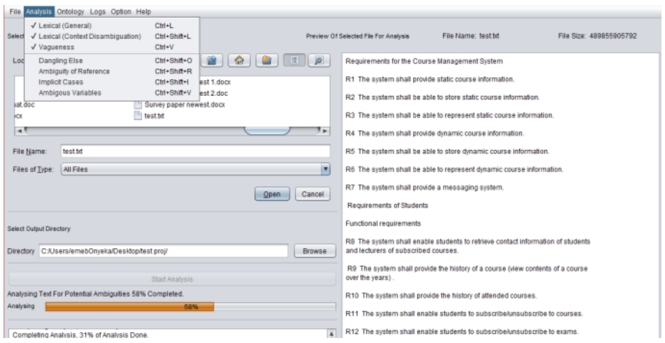


Fig 4: An Example of a Structural Isomorphism Network between Two Domains (a) (b).

IV. SYSTEM DEMONSTRATION

We provide a demo of our PROMIRAR system with various snapshots. A few of these are shown below while more will be available in a live demo. Figure 5 is a snapshot of the screen for PROMIRAR Input and Analysis. User interaction and I/O occur as explained next.


File
Analysis
Ontology
Logs
Option
Help
✓ Lexical (General) Ctrl+L
✓ Lexical (Context Disambiguation) Ctrl+Shift+L
✓ Vagueness Ctrl+V
Dangling Else Ctrl+Shift+O
Ambiguity of Reference Ctrl+Shift+R
Implicit Cases Ctrl+Shift+I
Ambigous Variables Ctrl+Shift+V
Open
Cancel
Browse
Start Analysis

Select Output Directory
Directory: C:/Users/emebOnyeka/Desktop/test proj/
File Name: test.txt
Files of Type: All Files
Analysing Text For Potential Ambiguities 58% Completed.
Analysing 58%
Completing Analysis. 31% of Analysis Done.

Preview Of Selected File For Analysis File Name: test.txt File Size: 489855905792

Requirements for the Course Management System
R1 The system shall provide static course information.
R2 The system shall be able to store static course information.
R3 The system shall be able to represent static course information.
R4 The system shall provide dynamic course information.
R5 The system shall be able to store dynamic course information.
R6 The system shall be able to represent dynamic course information.
R7 The system shall provide a messaging system.
Requirements of Students
Functional requirements
R8 The system shall enable students to retrieve contact information of students and lecturers of subscribed courses.
R9 The system shall provide the history of a course (view contents of a course over the years) .
R10 The system shall provide the history of attended courses.
R11 The system shall enable students to subscribe/unsubscribe to courses.
R12 The system shall enable students to subscribe/unsubscribe to exams.

Fig 5: Demo Snapshot of PROMIRAR Input / Analysis Screen

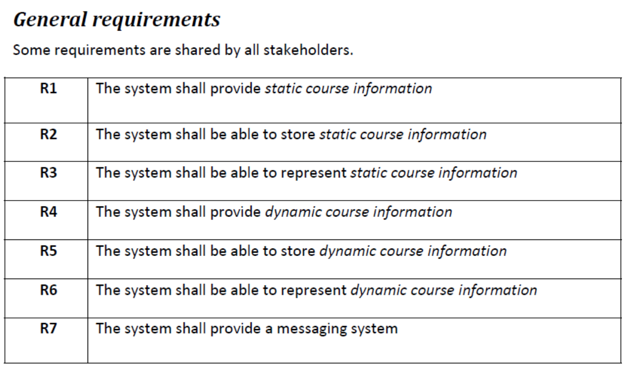
A. *User Interaction with the PROMIRAR Tool*

The process of using the PROMIRAR tool is as follows.
Preprocess: Source documents are converted to obtain requirements in textual format (without graphics, images, and tables).

Import: Requirement documents and domain ontology are transferred to the PROMIRAR environment.
Analyze: Possible sources of IMR are outlined by the feature extractor.
Identify: Potential IMRs are detected along with suitable recommendations.
Manage: The recommendations are used to handle IMRs, this could include expert opinion and then serves as the output. Each IMR that is approved as well as its explicated part are then stored in the case base of PROMIRAR.

B. Input/Process/Output of PROMIRAR

The text source for our demo shown here makes use of the Course Management System (CMS) requirements specification document [1]. This document was created for adoption at the University of Twente and is potentially useful in AI applications such as intelligent tutoring systems. The requirements describes some basic functionality such as student course enrollment, course notes and timetable upload, grades of student and e-mails communication. An excerpt from a sample requirements specification is as shown in Figure 6.

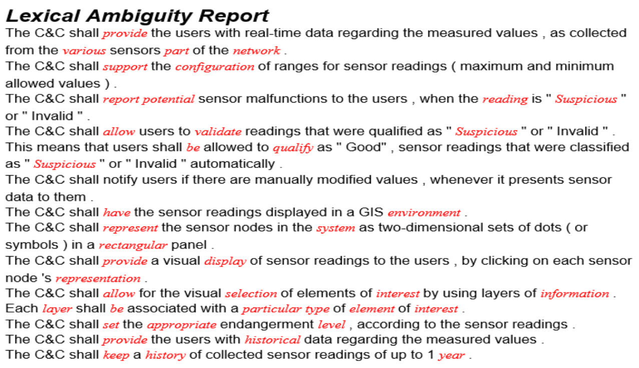

General requirements

Some requirements are shared by all stakeholders.

R1	The system shall provide *static course information*
R2	The system shall be able to store *static course information*
R3	The system shall be able to represent *static course information*
R4	The system shall provide *dynamic course information*
R5	The system shall be able to store *dynamic course information*
R6	The system shall be able to represent *dynamic course information*
R7	The system shall provide a messaging system

Fig 6: Excerpt from CMS Requirements Specification

The process of analysis for identification of implicit requirements uses the feature extraction module and the heuristic classifier based on some characteristic features that could potentially make a natural language text implicit. A partial snapshot of the output after the analysis, to identify potential implicit requirements as contained in the document, appears in Figure 7. This refers to a lexical ambiguity report pertaining to the IMRs.


Lexical Ambiguity Report

The C&C shall *provide* the users with real-time data regarding the measured values , as collected from the *various* sensors *part* of the *network* .
The C&C shall *support* the *configuration* of ranges for sensor readings (maximum and minimum allowed values) .
The C&C shall *report potential* sensor malfunctions to the users , when the *reading* is " *Suspicious* " or " Invalid " .
The C&C shall *allow* users to *validate* readings that were qualified as " *Suspicious* " or " Invalid " .
This means that users shall *be* allowed to *qualify* as " Good" , sensor readings that were classified as " *Suspicious* " or " Invalid " automatically .
The C&C shall notify users if there are manually modified values , whenever it presents sensor data to them .
The C&C shall *have* the sensor readings displayed in a GIS *environment* .
The C&C shall *represent* the sensor nodes in the *system* as two-dimensional sets of dots (or symbols) in a *rectangular* panel .
The C&C shall *provide* a visual *display* of sensor readings to the users , by clicking on each sensor node 's *representation* .
The C&C shall *allow* for the visual *selection* of elements of *interest* by using layers of *information* .
Each *layer* shall *be* associated with a *particular type* of *element* of *interest* .
The C&C shall *set* the *appropriate* endangerment *level* , according to the sensor readings .
The C&C shall *provide* the users with *historical* data regarding the measured values .
The C&C shall *keep* a *history* of collected sensor readings of up to 1 *year* .

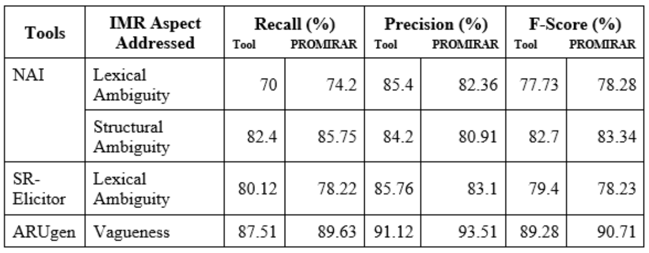
Fig 7: Demo Snapshot of Lexical Ambiguity Output

Likewise, many more examples can be depicted in a live demo to illustrate the detailed functioning of the PROMIRAR system for identification and management of IMRs. We would specifically consider examples useful in AI tools, since implicit requirements are highly critical in such applications.

V. EXPERIMENTAL EVALUATION

PROMIRAR is evaluated with real data for software development in course management, smart cities and tactical control. Ground truth is annotated by experts. Evaluation metrics used are Recall R = TP/(TP + FN), Precision P = TP/(TP + FP), F-score F = 2P * R / (P + R) where TP, TN, FP, FN are true positives, true negatives, false positives, false negatives respectively (TP: requirements judged by expert and PROMIRAR as implicit, TN: both as explicit, FP: requirements judged by PROMIRAR as implicit and expert as explicit, FN: vice versa). A group of experts were requested to manually highlight implicitness in the requirement document as well as make use of the PROMIRAR tool.
The experts are a collection of computing specialists, which encompasses software engineers/developers, academics and research students. Each of this expert were given this set of instructions: 1) for each itemized requirement, highlight the kind of implicit nature of that requirement (bearing in mind that a particular requirement may have more than one kind of implicitness). 2) For each itemized requirement, on a scale of 1 to 5, state the degree of criticality of each requirement's implicitness. (5 = most critical to 1 = least critical). The kinds of implicitness comprises i) Ambiguity (A) ii) Incomplete Knowledge (IK) iii) Vagueness (V) iv) Others (specify).
The result of the evaluation achieved by making use of the three requirements documents, the mean precision, recall and F-score were computed with results **R=83.20%, P=86.16%,** and **F=84.51%** respectively. Since PROMIRAR perform the role of detecting IMR, the outcome of its recall is certainly more significant than its precision. In a best case scenario, recall ought to be 100%, as it would save human analysts from the ecclesiastical job of analyzing the document [18]. PROMIRAR with a mean recall value of 83.20% shows that the tool in reality is adequate for use, as it clearly highlighted a minimum of six out of eight IMR discovered by a human expert and this is at par with best practices. The mean precision of 86.16% shows that the proportion of IMR detected manually by experts were also highlighted by the PROMIRAR too and it is well above average. This is also at par with best practices. The F-score which is 84.51%, clearly shows that PROMIRAR is very efficient. Based on manual examination, IMR highlighted by human evaluators but missed by PROMIRAR, shows that they denote implicit factors where PROMIRAR could not recognize the explicit forms that could help automate the detection of IMR. A further observation at the evaluation experiment, showed that the PROMIRAR tool's performance is highly influenced by the domain ontology's quality (i.e. the richness of vocabulary and coverage of the ontology with respect to a specific domain increases the accuracy of PROMIRAR).
Comparative assessment of PROMIRAR was conducted with related tools NAI, SR-Elicitor and ARUgen [15, 20, 21]. The assessment results are summarized in Table I.

Table I: Comparative Assessment of PROMIRAR with Other Tools



Tools	IMR Aspect Addressed	Recall (%)		Precision (%)		F-Score (%)	
		Tool	PROMIRAR	Tool	PROMIRAR	Tool	PROMIRAR
NAI	Lexical Ambiguity	70	74.2	85.4	82.36	77.73	78.28
	Structural Ambiguity	82.4	85.75	84.2	80.91	82.7	83.34
SR-Elicitor	Lexical Ambiguity	80.12	78.22	85.76	83.1	79.4	78.23
ARUgen	Vagueness	87.51	89.63	91.12	93.51	89.28	90.71

Comparison shows that for Lexical Ambiguity and Structural Ambiguity, PROMIRAR is better than NAI and SR-Elicitor in Recall and F-Score; and is almost at par in Precision. For Vagueness, PROMIRAR does better that ARUgen across all metrics. Hence, we can conclude from our experimental evaluation that PROMIRAR on the whole outperforms the state-of-the-art.

VI. CONCLUSION

This paper presents a novel system called PROMIRAR to automate early identification and management of IMRs in SRS. A significant aspect is that it embodies commonsense with ontology and text mining to manage IMRs. PROMIRAR is evaluated with real data in specific applications. It overshadows other tools for IMRs. Use of PROMIRAR can augment implementation, reduce bugs and enhance software development. As ongoing work, we would consider replacing the heuristics based classifier with a neural classifier having LSTM architecture over text. We would also deploy a softmax layer that classifies requirements as implicit or explicit. Applications of PROMIRAR entail AI tools in various areas, e.g., intelligent tutors, smart cities etc. where implicit requirements are crucial. PROMIRAR would be very interesting to professionals in requirements engineering and knowledge management. It presents interdisciplinary research in these fields, overlapping artificial intelligence and software engineering.

ACKNOWLEDGMENT

A part of this work was conducted when Onyeka Emebo who was a visiting scholar from Covenant University, Nigeria embarked on Fulbright Scholar at Montclair State University, USA. The authors greatly appreciates the source of the Scholarship for this funding.

REFERENCES

[1] Abma, B. J. M., "Evaluation of requirements management tools with support for traceability-based change impact analysis," Master's thesis, University of Twente, Enschede 2009.
[2] Douglas B. Lenat, "CYC: A Large-Scale Investment in Knowledge Infrastructure," Comm. of the ACM 38(11), pp. 32-38, 1995.
[3] Dreyer, H., Wynn, M. G., & Bown, G. R..., "Tacit and Explicit Knowledge in Software Development Projects: Towards a Conceptual Framework for Analysis," In eKnow 7th International Conference on Information, Process and Knowledge Management (No. A, pp. 49-52). ThinkMind, 2015.
[4] Emebo, O.; Olawande, D.; and Charles, A. "An automated tool support for managing implicit requirements using analogy based reasoning" In IEEE RCIS, 1–6, 2016.
[5] Emebo, Onyeka, Olawande Daramola, and Charles Ayo. "A survey on implicit requirements management practices in small and medium-sized enterprises." Tehnički vjesnik 24.Supplement 1, 219-227, 2017.
[6] Fabbrini, F., Fusani, M., Gnesi, S., & Lami, G, "An automatic quality evaluation for natural language requirements," In Proceedings of the Seventh International Workshop on Requirements Engineering: Foundation for Software Quality REFSQ, Vol. 1, pp. 4-5, 2001.
[7] Gervasi, V.; Gacitua, R.; Rouncefield, M.; Sawyer, P.; Kof, L.; Ma, L.; and Nuseibeh, B, "Unpacking tacit knowledge for requirements engineering," Managing requirements knowledge 23–47, 2013.
[8] Gharehchopogh, F. S., and Khalifelu, Z. A., "Analysis and evaluation of unstructured data: text mining versus natural language processing," In AICT, 1–4, 2011.
[9] Kamsties, E., Berry, D. M., Paech, B., Kamsties, E., Berry, D. M., & Paech, B., "Detecting ambiguities in requirements documents using inspections," In Proceedings of the first workshop on inspection in software engineering WISE'01, pp. 68-80, 2001 .
[10] Lami, G., Gnesi, S., Fabbrini, F., Fusani, M., & Trentanni, G., "An automatic tool for the analysis of natural language requirements," Informe técnico, CNR Information Science and Technology Institute, Pisa, Italia, Setiembre, 2004.
[11] Meyer, B., "On formalism in specifications," IEEE software, 2(1), 6, 1985.
[12] O. Daramola, T. Moser, G. Sindre, and S. Biffl, "Managing Implicit Requirements Using Semantic Case-Based Reasoning," Research Preview. REFSQ 2012, LNCS 7195, pp. 172–178, Springer-Verlag Berlin Heidelberg, 2012.
[13] Olmos, K., and Rodas, J., "Kmos-re: knowledge management on a strategy to requirements engineering," Requirements Engineering 19(4):421–440, 1993.
[14] R. Gacitua, B. Nuseibeh, , P. Piwek, , A.N. de Roeck, , M. Rouncefield, , P. Sawyer, , A. Willis, and H. Yang, "Making Tacit Requirements Explicit", Second International Workshop on Managing Requirements Knowledge (MaRK'09) 2009.
[15] Shah, U., and Jinwala, D., "Resolving ambiguities in natural language software requirements: A comprehensive survey," ACM SIGSOFT Software Engineering Notes 40(5):1–7, 2015.
[16] Singh P, Lin T, Mueller E, Lim G, Perkins T, Zhu W. "Open mind common sense: Knowledge acquisition from the general public," In Proc. Conf. Cooperative Information Systems, pp.1223-1237, 2002.
[17] Spanoudakis, G., "Analogical reuse of requirements specifications: A computational model," Applied Artificial Intelligence, 10(4), 281-305, 1996.
[18] Kiyavitskaya, N., Zeni, N., Mich, L., and Berry, D. M. "Requirements for tools for ambiguity identification and measurement in natural language requirements specifications," Requirements Engineering, 13(3): 207-239, 2008.
[19] Maiden, N. A. M. and Sutcliffe, A. G., "Exploiting Reusable Specifications through Analogy," Communications of the ACM, 34(5): 55-64, 1992.
[20] Umber, A., Bajwa, I. S., and Naeem, M. A., "NL-based automated software requirements elicitation and specification," In International Conference on Advances in Computing and Communications (pp. 30-39). Springer Berlin Heidelberg, 2011.
[21] Wilson, W. M., Rosenberg, L. H., & Hyatt, L. E., "Automated analysis of requirement specifications," In Proceedings of the 19th international conference on Software engineering (pp. 161-171). ACM, 1997.