A SURVEY ON ABSTRACTION IDENTIFICATION TECHNIQUES IN REQUIREMENT ENGINEERING

J. Yesudoss[1], A.V. Ramani[2]

ABSTRACT

Understanding the concept of unfamiliar domain or requirements document is very important for the domain expert and requirements engineer. Abstraction identification is a process to get a quick grasp of the important concepts present in the particular requirement documents. To identify the concept with in unstructured paper and from unconstrained natural language from requirements documents are challenging in software engineering. Many techniques have been proposed for abstraction identification. A New approach Ontology Based Abstraction Identification is proposed to give the precise understanding of the abstraction identification. The comparative study presented in this paper will provide the guidelines to requirement engineers, domain experts and software developers for the quick software building ability.

Key words : *Software requirements engineering, Software Requirement Specification, Abstraction Identification, Information Retrieval, Term extraction, Natural language processing, Ontology Learning.*

[1]Assistant Professor, Department of Computer Science Sri Ramakrishna Mission Vidyalaya College of Arts and Science, Coimbatore – 641 020, India.
E-mail : jydoss@gmail.com

[2]Associate Professor and Head, Department of Computer Science Sri Ramakrishna Mission Vidyalaya College of Arts and Science, Coimbatore – 641 020, India.
E-mail : avvramani@yahoo.com

I. INTRODUCTION

Managing software consistency should be carried out at the very initial stage of software engineering [3] - software requirement analysis (SRA). Abstraction identification has been proposed and evaluated as a useful technique in requirement engineering (RE).

The term Abstract[13] means theoretical, conceptual, intangible, summary, extract, take out, select, precise, synopsis, or short version.

"Abstraction" refers to an entity or concept that has a particular significance in the domain. "Abstraction" literally Means, concept, idea, thought, notion, construct or generalization. This may be formed by reducing the information content of a concept or a noticeable incident, usually to preserve only information that is relevant for a particular purpose.

Computer scientists use "abstraction" and communicate their solutions with the computer in some particular computer language. Abstraction allows software developers to separate categories and concepts from instances of implementation. So they do not depend on concrete details of software or hardware, but on an abstract contract. Abstraction is a process by which concepts are derived from the usage and classification of first principles, literal ideas or other techniques.

II. Abstraction Identification

An early stage in requirements engineering is abstraction identification. Abstraction identification is named as a key problem in Requirements Analysis (RA) [7]. Hence, abstraction identification has been done manually by an RA. The RA scans all the transcripts, trying to note important subjects and objects of sentences, i.e., nouns. The problem is that humans get tired, get bored, fall asleep, and overlook relevant ideas. So it is proposed the tools that do the clerical part of the search without getting tired, falling asleep and overlooking anything. The human RA still has to do all of the thinking with the output of the tools, but he or she will be confident that no piece of information has been overlooked in the process of gathering input to the human process of abstraction identification.

In Software Requirement Specification, Final product "Abstraction Identification" is simply known as A Contract Document, which is used as a communication medium between the supplier and the customer. Once the software requirement specification is finished and accepted by all customers, the end of the requirements engineering phase has been reached. It is not to say, that after the acceptance phase, no requirements can be changed, but the changes must be strongly controlled. The software requirement specification should be edited by both the supplier and the customer, as initially neither has both the knowledge of what is required(the seller) and what is feasible (the customer) [15].

a. Software Requirement Specification

A software requirement specification has a number of purposes and contexts in which it is used. This can vary from a group of companies publishing a software requirement specification to other companies for competitive tendering or the companies writing their own software requirement specification in response to a user requirement documents. Firstly, the author of the document has to write the document in such a way that it is general enough as to allow a number of different suppliers. This leads to propose solutions, and in the same time it has number of constraints which can be used in the requirement documents. Secondly, the software requirement specification is used to capture the user's requirements and if any best part of the requirement's conflicts and inconsistencies define system and acceptance testing activities.

A software requirement specification in its most basic form is a formal document used in communicating the software requirements between the developer and the customer. With this knowledge the minimum amount of information of the software requirement specification should contain is a list of requirements which has been agreed by both parties. The different types of requirements can be categorized as: Functional, Performance, Documentation, Operational, Interface, Resource, Safety, Quality, Reliability, Acceptance testing, Verification and Maintainability requirements.

Here, the part of Requirements document is a reference document, i.e., SRS document which is a contract between the development team and the customer. The SRS document is known as black-box specification. The black box system is considered as an internal detail is not known. Only its visible external (i.e. input/output) behavior is documented. Requirements traceability was defined as "the ability to describe and follow the life of a requirement, in both ahead and back direction".

However the requirements will only give a narrow view of the system and additional information is required to place the system into a context which defines the purpose of the system. This additional information will aid the developer in creating a software system which will be aimed at the user's ability and the clients function [15].

III. Problems in Abstraction Identification

Major problem in software development is to find Abstraction identification which is faced by requirement engineers, domain experts and software developers. Problems in software requirement specifications are divided into the following categories.

- Problems in classification
- Problems in identification, (in any software model construction)

IV. Methods

The following methods are explained to solve the problems in Abstraction Identification in Requirement Engineering.

1. Relevance based abstraction identification.

In Relevance based Abstraction Identification (RAI) method, it compares with the human judgement, and discusses the techniques, how RAI could be used to good effect in Requirements Engineering [10].

The two techniques RAI-0 and RAI-1 are used to identify the concepts in requirement document.

The first version of RAI is RAI-0 which is clearly reported by the following procedure.

1. Every word in the domain text is annotated with a Part of- Speech tag (PoS tag).
2. The set of words is filtered to remove common words unlikely to signify abstractions. Several lists of such stop words exist; RAI uses the ONIX (Onix text retrieval toolkit) list.
3. The remaining words are lemmatized to reduce them to their dictionary form, to collapse inflected forms of words to a base form or lemma. Thus, for example, tag and tags will be recognized as terms referring to the same concept despite having different lexical forms.
4. Each word is assigned a Log-Likelihood (LL) value by applying the corpus-based frequency profiling approach described above, using the 100 million words British National Corpus (BNC) as the normative corpus K.
5. Syntactic patterns are applied to the text to identify multiword terms.
6. A significance score is derived for every term by applying.
7. Identified terms are sorted based on their significance score and the resulting list is returned.

The procedure of RAI-I is as follows:

The major modification to RAI-0 made for RAI-1 addressed a pattern that was noticed in the output from RAI-0. This pattern was an effect of the variable values of k assigned to component words of multiword terms. It is established that assigning variable weights produced slightly better performance than weighting every component word equally. However, as a side-effect, compound terms composed of the same number of words and ending in the same head word tended to cluster

together. This is because the headword is weighted most heavily by being assigned the highest value of k in. If the headword has a relatively high LL value compared to the other list words in the term, it will make a dominating contribution to the term's overall significance value. Thus, terms that share the same high LL-value headword, tend to have similar significance values.

Human judgement is needed to infer abstractions from their signifiers. RAI is designed so that the requirements engineer would work down from the most significant term, stopping when recall dropped below an acceptable level but drawing on other evidence to help form an opinion about whether each term was a genuine domain abstraction.

These are presented a technique for the identification of single- and multiword abstractions which we call Relevance-Based Abstraction Identification (RAI).

2. Automatic Results Identification

A method for Automatic Results Identification in Software Engineering Papers [8] explains an analysis of the main methods for sentence classification in scientific papers and evaluates the feasibility of this technique in unstructured papers in Software Engineering area. A summary of automatic analysis of abstractions and indexing can also be found from the requirements document [5]. In order to automatically find the results or terms, tests are conducted with the existing methods using unstructured Test Software papers [8,9]. These results are far below reported by the authors with input sets composed by structured health papers. Three different approaches followed by three different authors are as follows[8]:

Agarwal [1,8] has described the methods in classifying sentences automatically. A rule-based and machine-learning approach is explained to classify a sentence into the "Introduction, Methods, Results and Discussion" categories. Shortly it is known as IMRAD. [1,4].

The following procedures are the IMRAD categories:

i. A Baseline System

ii. A Rule-based System

iii. Supervised Machine-Learning Systems Trained on Non-Annotated Corpus

iv. Supervised Machine-Learning System Trained on Manually Annotated Full-Text Sentences

v. Machine-Learning Systems

Ibekwe-SanJuan [6,8]has posted the results for "Identifying Strategic Information from Scientific Articles through Sentence Classification" topic and Lexico-syntactic patterns method to classify the sentences is used here. After sentence categorization, users can navigate the result by accessing specific information types and the annotated sentences are clustered. These results can be used for advanced information retrieval purposes.

Teufel [8,11] has explained about classification techniques in his topic "sentence extraction as a classification task". A specialist can be used to give booklovers an idea of what the longer text is about or it can used as input into a process to produce a more coherent abstract. Here, a method used for sentence selection as classification.

3. Automatic extraction of glossary terms

A method for the "Automatic Extraction of Glossary Terms from Natural Language Requirements" is presented in this topic [2]. The glossary terms are identified in two steps:

(a) Compute units (which are candidates for glossary terms)

(b) Precise the result between the mutually exclusive units to identify terms.

This introduces novel linguistic techniques to identify auxiliary verbs [14], process and abstract nouns. The recognition of units also handles adjectival modifiers and co-ordinating conjunctions. This requires solving adjectival modifier ambiguity and co-ordination ambiguity. The identification of terms in the units adapts an in-document statistical metric. Here, an evaluation of a method is presented over a real-life set of software requirements' documents with a base algorithm and compare our results. The problematical linguistic classification and the tackling of ambiguity result in superior performance by the base algorithm as followed:

- Categorize nouns into abstract, concrete and process
- Categorize verbs into auxiliary and concrete
- Explanation of co-ordinating conjunctions and adjectival modifiers of nouns
- Statistical metric for solving coordination ambiguity and adjectival ambiguity.

Automatic glossary term extraction uses the following techniques [2]:

A. Pre-Processor techniques

B. Unithood Determination, includes

1) Handling Co-ordinating Conjunctions

2) Handling Adjectival Modifiers

3) Handling Nominalizations

C. Termhood Determination, includes the step

1) Linguistic Techniques and

2) Statistical Techniques, in Termhood Determination

The pre-processor identifies requirements sentences as those which begin with an explicit label and identifies sentence boundaries. Words in title case, words in capital case and words in quotes are converted into an internal parser token. However, a list of exceptions are maintained containing words such as – 'only', 'always', 'and', etc. where the case of such words are ignored. All acronyms are automatically treated as glossary terms and placed in the entity list. The contents within the brackets and the brackets themselves are pruned from the requirement. Finally, extra white spaces from the requirements are trimmed.

The Unithood Determination architecture identifies all "units" in a requirement sentence. A unit is a word or phrase that conveys meaning. The approach is to look for infinitives of verbs and the algorithm: "Algorithm to identify nominalizations" is given below:

A noun phrase such as 'data synchronization' is split into two units - 'data' and 'synchronization'.

1. Indicator Noun (IN) = the head word in the noun phrase.
2. if IN is one of the predefined process nouns like "event", "process", and "method"
3. the Noun is process
4. else
5. for the IN, Find the infinitive modifier type

6. if the infinitive is a verb then tag based on modifier
7 if the modifier ends in "tion", "ment", etc
8. Noun is a process nouns
9. end if
10. end if
11. end if

In Termhood Determination, the units determined in the previous module are now checked for termhood - i.e. identify the units in two steps which truly carry out domain specific information according to the given requirements' document.

a) The algorithm "Linguistic technique in Term hood Determination" is given below for the determination of the terms from it.

1. input – The object, U, with the units
2. for each row in U (till k rows)
3. for each Unit in
4. Indicator Noun (IN) = the head word in the Unit.
5. if IN is one of the {list of common concrete nouns}
6. tag as Concrete, continue to next Unit
7 else if the infinitive of the IN is an adjective then tag as abstract noun
8. else if the infinitive of the IN is a verb then tag based on modifier
9. if the modifier ends in "er", "or" then tag as concrete noun.
10. else tag as Abstract noun.
11. end if
12. Prune Units that are abstract or contains 'below', 'following', anaphora, etc
13. end for
14. end for

b) Statistical Technique in Termhood Determination, involves Determination of terms which is done in three steps. In the first step, all those units that have no mutually exclusive options (i.e. no ambiguity exists) are chosen as terms. In step two, the resolved terms are used to select among the mutually exclusive units. This step is run recursively till no more resolutions are possible. Finally, in step three, those options that could not be resolved are shown to the user for resolution.

By using these algorithms, this paper explains the Automatic extraction of glossary terms from natural language requirements.

4. AbstFinder

The approach of a tool AbstFinder [7] is for a Prototype Natural Language Text Abstraction Finder for Use in Requirements Elicitation. Typically, the abstractions must be found among the large mass of natural language text collected from the clients and users. To solve the problems of requirements elicitation, this paper motivates and describes an approach, based on traditional signal processing methods, for obtaining abstractions in natural language text and presents a new tool "AbstFinder" as an implementation.

In this paper, the tool AbstFinder describes the phrases of the remainder were analyzed very carefully in order to

see if AbstFinder missed any abstraction. The phrases of the remainder were separated to several categories according to their characteristics as follows:

1. Most of the phrases originate from the strict meta-language of the requirements specification format of the human-made document, such as "activate", "allow", "deactivate", "herein", "include", "integrate", "must", "only", "provide", which are not abstractions and were used only in the human-made document for stating requirements and not in the RFP original transcript.

2. Some concepts were in different grammatical forms such as "transmit" in the Abst-Finder abstraction list, and "transmitting", "transmitters", and "transceiver" in the human-made document. Those words in the leftovers do not carry any new concept, they actually describe the same abstraction. The same is for: "calibrate" and "calibration" "assigned" and "assignment". While AbstFinder is designed to classify all the "transmit..." words as single abstractions, Strainer is designed to remove only whole words and does not remove words that properly contain a recognized root. If Strainer were to remove parts of words, then the remainder of the document will be a mass of unreadable text. For instance, suppose that "inter" were found by AbstFinder as a common part among "interchangeability" and "interfaces". Removing "inter" as part of word would leave in the leftovers "changeability" and "faces". Both of these accidentally generated words are garbage relative to the application.

3. Acronyms such as "NBC" are introduced to replace a longer full phrase such as "Nuclear, Biological, Chemical"; the full phrase appears only once at the introduction of the acronym or in a dictionary of acronyms, and the acronym appears many times throughout the document. The acronyms are used to save the writing of the longer full phrase. AbstFinder did not identify many acronyms. Many acronyms are shorter than the *WordThreshold*, and a full phrase if appears only once it is not going to be caught by any frequency-based scheme. Actually, only the "NBC" was not found, all the others were found since the term of the acronyms were repeated in the text more than once. Given that reducing *WordThreshold* causes generation of too much noise, there are two solutions, both general enough to be made part of a standard scenario for the elicitor.

   a. The synonym dictionary can be used to replace the acronyms by their full phrases for the purpose of abstraction identification.

   b. Recognize all the acronyms as important abstractions, log them as abstractions, and then add them to the *ignored-application-phrases-file*. Only after recognizing the abstractions, the elicitor may switch to using acronyms as abstractions identifiers.

4. Ten concepts appeared in the leftovers because they appear in the RFP only once, and AbstFinder identifies only concepts that appear more than once, at least once for definition and once for use. Of these, five phrases were synonyms in the context of the system that was defined in the RFP,

such as "contour" and "elevation", and "enemy" and "threats" that occurred because the *synonym-file* was not implemented yet.

5. The remaining five phrases were specific examples of some already captured abstractions and appeared in the text with linguistic clues, "i.e.", "e.g.", and "for example". These are not abstractions; they are details that will be put inside the abstraction.

To sum up, after some generally applicable modifications that should be part of a standard scenario for use of AbstFinder, full coverage was achieved. This, by the way, is how each case study led to the refinement of the use scenarios.

5. Ontology based abstraction identification

Ontology is an explicit description of a domain which includes hierarchical relationships, properties and attributes of concepts, properties and attributes of constraints and Individuals. Ontology defines a common vocabulary and a shared understanding. It can be developed to share common understanding of the structure of information among people and among software agents. This concept will be used to enable to reuse of domain knowledge for introducing standards to allow interoperability and to separate domain knowledge from the operational knowledge.

Ontology learning [12] greatly facilitates the construction of ontology by the ontology engineer. The notion of ontology learning that proposes includes a number of complementary disciplines that feed on different types of unstructured and semi-structured data in order to support a semi-automatic and cooperative ontology engineering process. Ontology learning framework proceeds through ontology extraction, ontology import, ontology refinement, and ontology pruning giving the ontology engineer a wealth of coordinated tools for ontology modeling.

Abstraction identification is an essential step in automatic ontology learning or model generation. Though it is mentioned all the proposed methods, an Ontology Based Abstraction Identification for requirement document is also needed for the better solution of abstraction identification.

IV. Discussion

The results of the above methods are discussed in the categorization of sentences and concepts that represent abstraction of requirement documents. The demonstrated feasibility study uses the different methods and techniques in unstructured requirement documents. The precision found in the case studies fell far short of the tests on terms and sentence classification. The authors have given their original studies and analyzed with structured and unstructured requirement documents from different areas. The summary of the methods proposed for different techniques in software requirements. A proposed method "Ontology Based Abstraction Identification" also introduced for further better results for abstraction identification to the use of software engineers and software developers.

V. Conclusion

Many techniques have been proposed to refer abstraction identification. The comparative study presented in this paper will provide the guidelines to requirement engineers, domain experts and software developers for the faster

knowledge of software constructing ability. Comparing other techniques proposed in this literature, the research of Ontology Based Abstraction Identification study will be a good result of the previous effectiveness in abstraction identification.

REFERENCES

[1] Agarwal, S.; and Hong Yu, *Automatically Classifying Sentences in Full-Text Biomedical Articles into Introduction, Methods, Results and Discussion.* In Proceedings of the AMIA Summit on Translational Bioinformatics, 2009.

[2] Anurag Dwarakanath, Roshni R. Ramnani, and Shubhashis Sengupta. *Automatic Extraction of Glossary Terms from Natural Language Requirements,* IEEE, 978-1-4673-5765-4, 314-319, 2013.

[3] Bing Chao, XiaoDong Zhu, Qiang Li, AnCe Huang, *Reliability Management in Software Requirement Analysis, Management of Innovation and Technology,* DOI:10.1109/ICMIT.2006.262394, IEEE, 2006.

[4] Burrough-Boenisch. J, International *Reading Strategies for IMRD Articles,* Written Communication 16 (3): 296–316, 1999.

[5] Edmundson HP and Wyllys RE, *Automatic abstracting and indexing—survey and recommendations,* ACM 4(5):226–234. doi:10.1145/366532.366545, 1961.

[6] Fidelia Ibekwe-SanJuan, Chaomei Chen and Roberto Pinho. *Identifying Strategic Information from Scientific Articles through Sentence Classification. 6th International Conference* on Language Resources and Evaluation Conference (LREC-08), Marrakesh, Morocco, 26 May - 1st June, 2008.

[7] Goldin, Leah and Daniel M. Berry, AbstFinder, a *prototype natural language text abstraction finder for use in requirements elicitation, Automated Software Engineering* 4, no. 4, 375-412, 1997.

[8] José Alberto S. Torres, Daniela S. Cruzes and Laís do Nascimento Salvador, *Automatic Results Identification in Software Engineering Papers. Is it possible?,* 12th International Conference on Computational Science and Its Applications, IEEE, 978-0-7695-4710-7/12, DOI 10.1109/ICCSA.2012.27, 108-112, 2012.

[9] Kageura, Kyo and Bin Umino, *Methods of automatic term recognition: A review, Terminology* 3, no. 2, 259-289, 1996.

[10] Ricardo Gacitua, Pete Sawyer, and Vincenzo Gervasi. *Relevance - based abstraction identification: technique and evaluation. Requirements Engineering* , Springer-London, DOI 10.1007/s00766-011-0122-3, 2011.

[11] Simone Teufel and Marc Moens. *Sentence Extraction as a Classification Task, Workshop 'Intelligent and scalable Text summarization,* ACL/EACL 1997, July 1997.

[12] Wilson Wong, Wei Liu and Mohammed Bennamoun. Ontology Learning from Text : *A Look Back and into the Future, ACM Computing Surveys,* Vol. 44, No. 4, Article 20, Publication date: August 2012.

[13] http://en.wikipedia.org/wiki/Abstract_summary.

[14] http://en.wikipedia.org/wiki/Auxiliary_verb.

[15] http://www4.informatik.tu- muenchen.de/proj/va/SRS.pdf, Chapter 3 - Software Requirements Specification.

Author's Biography

J. Yesudoss, received the M.Phil degree in Computer Science from Madurai Kamaraj University, in 2004. He is a research student of Bharathiar University. Currently, he is an Assistant Professor of Computer Science at Sri Ramakrishna Mission Vidyalaya College of Arts and Science, Coimbatore, Tamilnadu, India. His interests are in Software Engineering and Data mining.

Dr. A.V. Ramani, received the Ph.D. degree in Computer Science from the Bharathiar University, in 2009. Currently, he is an Associate Professor and Head of Computer Science at Sri Ramakrishna Mission Vidyalaya College of Arts and Science, Coimbatore, Tamilnadu, India. His interests are in Software Engineering and Data and information Modeling.