Journal of Theoretical and Applied Information Technology
20th September 2014. Vol. 67 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

SEMANTIC ANALYSIS OF SOFTWARE SPECIFICATIONS WITH LINKED DATA

[1]MARTIN DOSTAL, [2]MICHAL NYKL, [3]KAREL JEŽEK
[1]NTIS, Faculty of Applied Sciences, University of West Bohemia, Pilsen, Czech Republic
[2,3]Department of Computer Science and Engineering, FAS, University of West Bohemia
E-mail: [1]madostal@ntis.zcu.cz, [2]nyklm@kiv.zcu.cz, [3]jezek_ka@kiv.zcu.cz

ABSTRACT

Software development life cycle is the process involved in the design, development and improvement of a software application. Nowadays especially component systems are used due to possibility of implementing reusable independent modules. The individual module, known as a software component, can be implemented in a form of a software package, a web service or a web resource that encapsulates a set of related functions. Software products are derived in a configuration process by composing different components. Moreover a software product line enables stakeholders to derive a different software products based on their needs. This fact and need for validation of the software product and its components requires methods for software specification processing and matching with concrete sw properties. In this article we will propose an approach for semantic analysis of software specifications with Linked Data.

Keywords: *Software Specifications, Linked Data, Semantic Analysis*

1. INTRODUCTION

Essentially software is determined by its functional and non-functional characteristics [1]. Functional characteristics describe software behavior that can be directly implemented and evaluated by common programmer. Furthermore, software specification documents are usually focused on functional characteristics, because it is easy to understand and describe application behavior to the customer. Unfortunately, there has been a lop-sided emphasis in the functionality of the software, even though the functionality is not usable without the necessary non-functional requirements.

These requirements are usually hidden and it occurs at the time when it is least suitable. It includes usability, interoperability, flexibility, performance and software security. Real-world problems are more non-functionally oriented than they are functionally oriented, so even great application could be unusable if it disregards the typical use. These problems include high cost and slow processing, poor productivity, lower profit and it leads to unhappy customer.

In this article we will discuss our approach for semantic analysis of software specification using Linked Data (LD). The Web of Data is often illustrated as a fast growing cloud of interconnected datasets representing information about barely everything [2]. Linked Data refers to the Web of Data in contrast to the Web of Documents. Linked Data extends the current web that consists of documents and the computer-meaningless links between documents. In the case of Linked Data, not just documents but also data elements (things) and the links between these data elements exist. More over the links in LD are expressive, unambiguous and identified by URI. LD is therefore more structured and machine process able than Web of Documents [3].

In Section 2 we will discuss the related work in a field of Linked Data and related web services. In Section 3 we will describe problematic of key phrase extraction with focusing on Named Entity Recognition (NER) with Linked Data. Our approach for SW specification analysis will be introduced in Section 4. Section 5 is devoted to results and conclusions.

2. RELATED WORK

In this section we will introduce Requirements engineering (RE) and Linked Data applications. We will propose our approach to text analysis using Linked Data thus we need to explain the basics about Linked Data and related web services such as DBpedia Spotlight [4]. This web service is used for semantic enrichment of a text and it consists of methods for entity detection, candidate selection

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

and disambiguation based on related concepts extracted from Linked Data.

2.1 Requirements Engineering

Requirements engineering (RE) is the process of developing requirements through an iterative cooperative process of analyzing the problem, documenting the resulting observations in a variety of representation formats, and checking the accuracy of the understanding gained [7]. Requirements engineering identify the purpose of a software system, and documents it in a form that is suitable to analysis, communication and subsequent implementation [8]. If we want to build a software system it has to be described in some way before the design and implementation process will be started. Typically, these descriptions, known as requirement documents, are far from representing the real business logic. Instead, we have a set of statements that is:

- incomplete (forgotten features),
- inconsistent (contains contradictions),
- ambiguous (more possible interpretations).

Elements of the RE are elicitation, specification and validation as shown on Fig. 1.

- Elicitation – the aim is to gain knowledge relevant to a problem in order to produce a requirements model.
- Specification - Software Requirement Specification (SRS) will be described later in this article.
- Validation – the purpose is to check whether the requirements specification complies with stakeholders intentions.

Before software can be implemented, we need to find, analyze and describe customers' needs known as requirements. We distinguish the following kinds of requirements:

- Application requirements
  - Functional requirements
  - Non-functional or Extra-functional requirements
- Domain requirements – requirements that come from the characteristics of the system domain.

Benefits on applying ontologies in RE leads to reduce the negative effects of the previous factors on the RE processes. The potential uses of ontologies in RE include the representation of:

- The requirements model – requirements ontology,
- acquisition structures for domain knowledge – software requirements specification document ontology,
- the knowledge of the application domain – application domain ontology.

2.2 Linked Data

The concept of Linked Data [7] was first introduced by Tim Berners-Lee. He set up four rules for machine readable content on the Web:

- Use URIs as names for things.
- Use HTTP URIs so that people can look up those names.
- When someone looks up a URI, provide useful information using the standards (RDF*, SPARQL[1]).
- Include links to other URIs so that they can discover more things.

More specific is the idea of Linked Open Data[2] (LOD), which is based on the presumption of freely published data without restrictions in usage or additional fees.

The Linked Data initiative has given rise to an increasing number of RDF[3] documents as well as other machine-readable sources, many of which are freely accessible online. These resources are often created as a result of database exports. That is the reason why we have to deal with duplicate information sources. There are two basic problems with duplicates resources: disambiguation and co-reference resolution. These problems were discussed in [8]. DBLP[4] and DBpedia[5] are two of those common Linked Data resources often used for academic research.

Linked Data contains information about a resource and moreover links to other related resources. There are two basic types of links that we can directly use:

- Parent-child relation,
- links to synonyms.

These connections are bidirectional so a child can find his parent and a parent can find his children. Relations are described by ontology predicates. For example: "dbpedia-owl:genre", "skos:broader", "dcterms:subject". The meaning of these predicates differs, but we can use it in the same way.

1 http://www.w3.org/TR/rdf-sparql-query/
2 http://lod-cloud.net
3 http://www.w3.org/RDF/
4 http://dblp.uni-trier.de
5 http://dbpedia.org

Journal of Theoretical and Applied Information Technology
20th September 2014. Vol. 67 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

An example of these relations between resources is shown in Fig. 2.

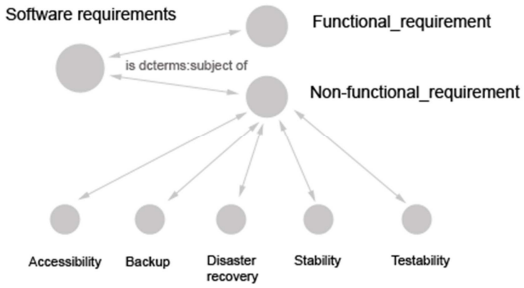

Figure 2: Example of hierarchical relations between nodes in LD

Synonyms are designated by the ontology relation: "*owl:sameAs*", which indicates true synonyms, and the relation "*skos:related*", which indicates related concepts.

2.3 DBpedia Spotlight

DBpedia Spotlight [4] is an open source software designed for automatic processing, analyzing and semantic enrichment of a text. It automatically annotates mentions of DBpedia resources in text, and goes through the whole analysis life cycle. It consists of entity detection (spotting), candidate selection and disambiguation based on related concepts.

DBpedia Spotlight is not just a tool but moreover it can be used as a web service. Nowadays there are these REST endpoints available:

- Spotting - this service takes text input and recognizes the potential surface forms e.g. names of entities. Several spotting techniques are available, such as dictionary lookup and Named Entity Recognition (NER). The output is the list of annotations in structured form like XML or JSON.
- Annotate – runs spotting and disambiguation. It retrieves the candidate DBpedia resources, disambiguates them if needed, and links the mentions to the best one. These output formats are available: XML, JSON, HTML, RDFa and NIF.
- Candidates – similar as annotate, but does not disambiguate the candidates for each mention. Rather it returns a ranked list of candidates. This list contains attributes in the form of scores expressing the significance of the word. There are different scores based on links from other resources and the significance in current context.
- Disambiguate – does not do spotting, it just selects the candidates for the given mentions and does disambiguation. This web service will be deprecated soon.

JSON (JavaScript Object Notation) is an open standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs. It is used primarily to transmit data between a server and web application, as an alternative to XML. It was originally derived from the JavaScript scripting language. Nowadays JSON is a language-independent data format and JSON data is readily available in a large variety of programming languages. Example of JSON data:

```
" entities ": [{
    " entity ":" Tim Berners - Lee",
    " type ":" Person ",
    " uri ":" http :// dbpedia . org / resource
    /Tim_berners_lee ",
    " nerdType ":" http :// nerd . eurecom .fr/
    ontology #Person ",
    " startChar ":30 ,
    " endChar ":45 ,
    " confidence ":1 ,
    " relevance ":0.5
}]
```

DBpedia Spotlight works in four-stages:

- Spotting stage – it recognizes in a sentence the phrases that may indicate a mention of a DBpedia resource.
- Candidate selection – it is subsequently employed to map the spotted phrase to resources that are candidate disambiguations for that phrase.
- Disambiguation - uses the context around the spotted phrase to decide for the best choice amongst the candidates.
- Configuration - the annotation can be customized by users to their specific needs through configuration parameters.

We will discuss only first three of them because there are the most important for understanding of our approach.

2.3.1 Spotting step

This step uses a lexicon that was generated from the extended set of labels from the lexicalization

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

dataset. The implementation uses the LingPipe[6] Exact Dictionary-Based Chunker which relies on the Aho-Corasick string matching algorithm [9] with longest case-insensitive match.

2.3.2 Candidate selection

The aim of this phase is to map resource names from spotting to candidate disambiguation. The DBpedia Lexicalization dataset was used for determining candidate disambiguations for each surface form. This phase can also be viewed as a way to pre-rank the candidates for disambiguation before observing a surface form in the context of a paragraph. The DBpedia resource with the highest prior probability for a surface form is selected as the default sense according to its usage in Wikipedia.

DBpedia resource contains:

- Common names for resource
- Redirects from other resources – alternative spellings, aliases
- Disambiguation pages – link a common term to other resources

2.3.3 Disambiguation

The Inverse Candidate Frequency (ICF) was introduced in Spotlight [4] for disambiguation – see formula (1). This measure supposes that the discriminative power of a word is inversely proportional to the number of DBpedia resources it is associated with.

$$ICF(w_j) = \log \frac{|R_s|}{n(w_j)} = \log|R_s| - \log n\,(w_j) \quad (1)$$

Where:

- R_s is set of candidates for a surface form s
- $n(w_j)$ is total number of resources in Rs that are associated with the word w_j

3. NAMED ENTITY RECOGNITION USING LINKED DATA

In this section we describe a number of practical approaches [9] that commonly constitute annotations in different use cases. We will start from the simplest and go through the most powerful ones that take into account the word context obtained from Linked Data.

3.1 Lexicon-based phrase recognition

A simple approach for phrase recognition is the usage of a string matching algorithm that relies on a lexicon of name variations for the target terms in the knowledge base. The lexicon-based phrase recognition does not select phrases with regard to their context but just searches for any phrases known as possible DBpedia entities. This method produces a high number of false positives. One example is the set of function words that have entries on Wikipedia, but whose annotation would be undesirable in use cases such as blog annotation, because it would confuse the reader with too many unnecessary links. However, eliminating those phrases from the lexicon upfront is not an option, as they may have other significant meanings. For example the word 'up' can be a fiction word in some contexts or name of a movie "UP" by Pixar.

3.2 Noun-phrase chunk heuristic

In many cases, the objective of annotation is to mark the things being talked about in text. Therefore, a simple heuristic to eliminate false positives early in the process is to only annotate terms that are within noun phrases. We therefore extended the Lexicon-based phrase recognizer with a simple heuristic that only allows phrases that contain at least one noun.

3.3 Noun-phrase chunking with probabilistic dictionary

This method is similar to previous one. It assumes that we are considering only noun-phrases. However, instead of heuristic, it uses NP chunks extracted by a NP chunker. For each NP chunk it chooses the longest expression contained in the set of acceptable phrases (in the lexicon).

The method can be enriched for detecting common words like "do" or "make". The Wikipedia guidelines explicitly instruct users to "avoid linking plain English words". Therefore it will be a good idea attempt the detection of common words at phrase recognition time.

3.4 Keyphrase extraction

Keyphrase extraction is used to extract most frequent words which are significant for the document with respect to the application. Keyphrase extraction is frequently used in search engines and other text mining tasks. Some of the common tools and approaches for key phrase extractions are:

- Carrot 2 – it uses two algorithms: STC (Suffix Tree Clustering) and lingo. STC [10] is an incremental, linear time algorithm, which creates clusters based on phrases shared between documents. Lingo [11] is based on SVD and search complete keyphrase with some other constraints

[6] http://alias-i.com/lingpipe/

Journal of Theoretical and Applied Information Technology
20th September 2014. Vol. 67 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

keyphrases. With STC, lingo also uses TF-IDF and LSA. Lingo provides flexibility in input. Indexed documents can be obtained by Nabble, Solr or google search desktop.

- KEA [12] – it is a standard algorithm for keyphrase extraction. It provides provision of learning from RDF dictionary in SKOS format. The dictionary contains hierarchical taxonomy and it also gives options for machine learning by a tool Weka [12]. It is a common prediction algorithm for Named Entity Recognition.
- Maui – it is basic KEA tool but also gives options to boost taxonomy from Wikipedia.
- Stanford topic modelling tool – this tool uses LDA for learning topic. It takes input and output in CSV format. It also provides options for Machine learning.

3.5 Named Entity Recognition (NER)

In use cases such as the concept tagging in blog posts or online newspapers, we are usually focused on specific types like people and places. In these cases it is viable to apply named entity recognizers as a strategy for phrase recognition.

NER extended by noun phrase n-grams was proposed by [13]. It is a hybrid approach mixing named entities and more general terms within noun phrase chunks. It consider as phrases only the expressions marked as named entities by the NER phrase recognizer, the noun-phrase chunks extracted by a NP chunker, and all sub-expressions of up to 5 tokens of the noun-phrase chunks. This increases the coverage of the NER phrase recognizer, which tends to generate fewer phrases.

3.6 NER with Linked Data

We distinguish two types of named entity extractors as in [14]. First type is able to identify interesting keywords and classify them in taxonomy. Second type is able to additionally provide a link pointing to URI that disambiguates the named entity.

The most widely used Linked Data named entity extractors are: AlchemyAPI[7], DBpedia Spotlight[8], Lupedia[9], OpenCalais[10], Saplo[11] and Zemanta[12]. Next we will compare some of these tools.

AlchemyAPI, OpenCalais and Zemanta offer both free and commercial versions of access to their web services. Commercial and free versions are basically the same, but there are limitations in a number of free requests per day. On the other hand DBpedia Spotlight is completely free without any restrictions. Table 1 contains basic information about these popular Linked Data tools.

Table 1: Basic information about selected LD tools.

	Alch.	Spotli.	OpenC.	Zem.
No languages	8	3	3	1
Entity types	272	272	39	81
LOD dataset number	7	1	9	1

The creators of the DBpedia Spotlight (Fig. 2.) have compared their web service with a number of other NER extractors according to a particular annotation task [4]. The experiment consisted in evaluating 35 paragraphs from 10 articles in 8 categories selected from "The New York Times" and has been performed by 4 human raters. The final goal was to create wiki links. The experiment showed how DBpedia Spotlight overcomes the performance of other services to complete this task. Therefore it was used as a primary annotation tool in our case.

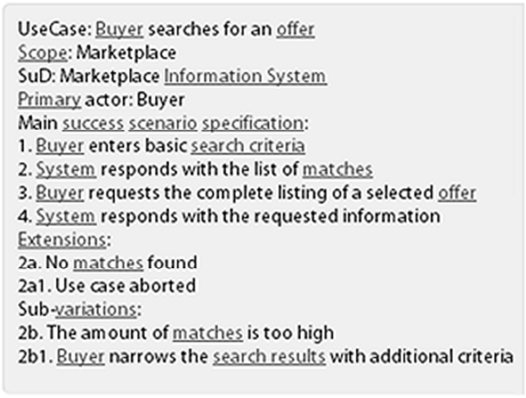

Figure 2: DBpedia Spotlight annotation

4. SW SPECIFICATION ANALYSIS

Our approach to semantic analysis of software specifications with Linked Data consists of these steps:

- Extraction and normalization of use-cases,
- extraction of functional and extra-functional properties,

[7] http://www.alchemyapi.com
[8] http://dbpedia-spotlight.github.com/demo/
[9] http://lupedia.ontotext.com/
[10] http://www.opencalais.com
[11] http://saplo.com/
[12] http://www.zemanta.com

Journal of Theoretical and Applied Information Technology
20th September 2014. Vol. 67 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

- actors detection.

4.1 Extraction and normalization of use-cases

Extraction of use-cases is based on combination of related keywords detection and use-case patterns recognition. The simplest case is to find words "use-case" or "UseCase" in a header or paragraph followed by a list of potential steps. Common software specifications usually involves structured lists of steps starting with a serial number or a special character like "-", "*", "#" etc. Normalization consists in replacing the starting character by a number and in the creation of a new list by merging several smaller. Example of use-case processing in the form of XML is in Table 2.

Table 2: Example of use-case processing

Input	Output
1. first step 2. second step	<use_case_step start_with="1." no="1">1. first step </use_case_step> <use_case_step start_with="2." no="2" >2. second step </use_case_step>
A. first step B. second step	<use_case_step start_with="A." no="1">A. first step</use_case_step> <use_case_step start_with="B." no="2" >B. second step </use_case_step>
- first step - second step	<use_case_step start_with="-" no="1"> - first step</use_case_step> <use_case_step start_with="-" no="2"> - second step</use_case_step>

4.2 Extraction of functional and extra-functional properties

Software Requirements Specification (SRS) documents contains all the requirements specifications for a software system, typically separated into functional requirements (FR) and non-functional requirements (NFR). NFRs usually impact the system as a whole and interact both with each other and with the functional requirements.

Fig 3 illustrates mappings from requirements items in a SRS document to elements in ontology. Mapping from requirements items to thesaurus can be written formally – formula (2):

$$F_{int} : ReqItem \rightarrow 2^{Con \cup Rel} \quad (2)$$

Where

- OntologySystem = (Con, Rel, Rules).
- Con is a set of concepts.
- Rel is a set of relationships.
- ReqItem is a set of requirements items in a document.
- F_{int} is the interpretation function. In case of Fig. 3: F_{int}(bbb) = {A, B, aggregate(A,B)} and F_{int} (ccc) = {D, E}

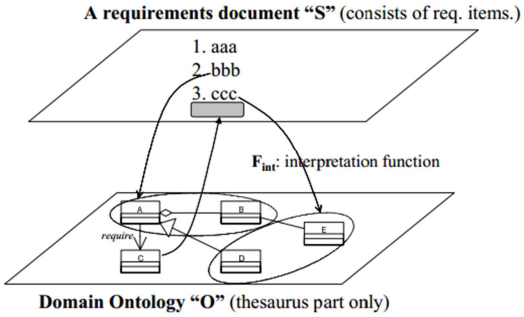

Figure 3: Mapping from SRS to ontology [15]

The mapping between the statements and ontology can also be done by using a frame of natural language. This approach is similar to NER but ontology is used instead of a lexicon. This approach provides following validation options:

- Inconsistency – approach tries to detect mutually contradicting elements where requirement items are mapped. For example relation between *run-time loading* and *short response time*.
- Completeness - completeness of a SRS document can be measured by number of ontology elements related to items in a document. Missing requirement items can be suggested to the user.

4.3 Actors detection

Actors detection is combination of methods for NER and requirements extractions proposed above. Example of entities extracted from case on Fig. 2. are:

- Concepts: search, search results, search criteria
- Actors: buyer, system
- Other entities: matches

4.4 Evaluation

Nowadays there is no public corpus for evaluation of approaches for SRS analysis. Therefore we had to evaluate our approach based on publicly available specifications found on Internet. We used a random set of 100 pages extracted from software specification documents. This set contained 100 use-cases, 351 actors and 235 features. Our simple ontology contained 120 triples in the form of a subject, predicate and object. Each

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

of these resources was identified by URI. The results are below:

- NER and actors detection – our experimental system found these entities with F1 = 0.8.
- Normalization of use-cases – 70% of use-cases was normalized correctly. The rest was not recognized due to missing titles like "Use case" and missing lists. Generally these use-cases were in the form of natural language sentences.
- Extraction of functional and extra-functional properties ended with F1 = 0,38 due to lacking ontology for these specifications. Domain ontologies are the best choice for related domain SRS. General ontologies are usually insufficient. It seems us as a good idea to use extraction ontology for a common combination of requirement property with SI base unit.

5. CONCLUSION

In this article we proposed an approach for semantic analysis of software specifications with Linked Data and ontologies. We introduced Linked Data, ontologies and relevant public web services like DBpedia Spotlight. The problem of named entity recognition extended by Linked Data was discussed. We evaluated application of Linked Data and ontologies to requirements engineering. The proposed methods for actors identification, extraction of functional and extra-functional properties based on Linked Data, standard ontologies and extraction ontologies are quite promising.

The evaluation of this approach was carried out using a subset from real life SRS documents. NER, actors detection and normalization of use-cases worked very well. On the other hand the requirement extraction is highly dependent on the quality of the used ontology. Only small, simple and general ontology was used in our case therefore the result in the form of F1 was relatively bad.

ACKNOWLEDGMENTS

This work was supported by the grants GAČR P103/11/1489 "Methods of development and verification of component-based applications using natural language specifications", European Regional Development Fund (ERDF), project "NTIS – New Technologies for the Information Society", European Centre of Excellence, CZ.1.05/1.1.00/02.0090, SGS-2013-029 "Advanced computing and information systems" and Ministry of Education of the Czech Republic - 7AMB14SK090 "MSMT Mobility".

REFERENCES:

[1] J. P. Leite L. Chung, "On non-functional requirements in software engineering," in Conceptual modeling: Foundations and applications, 2009, pp. 363-379.

[2] A. Jentzsch R. Cyganiak. (2011) LOD Community. [Online]. http://lod-cloud.net

[3] C., Heath, T., Berners-Lee, T. Bizer, "Linked data-thestory so far," in Int. Journal on Semantic Web and InformationSystems, Special Issue on Linked Data, 2009, pp. 1 - 22.

[4] M. Jakob, A. Garcia-Silva, and Ch. Bizer P. N. Mendes, "DBpedia Spotlight: Shedding Light on the Web of Documents," in 7th International Conference on Semantic Systems (I-Semantics), 2011.

[5] P. Loucopoulos and V. Karakostas, System Requirements Engineering., 1995.

[6] B., Easterbrook, S. Nuseibeh, "Requirements Enginnering: a road map," in Proceedings of ICSE'2000 - Future of Software Engineering Track, 2000, pp. 35-46.

[7] T. B. Lee. (2009) Design Issues. [Online]. http://www.w3.org/DesignIssues/LinkedData.html

[8] A. Jaffri, H. Glaser, and I. Millard, "URI Disambiguation in the Context of Linked Data," in LDOW 2008, Beijing, China, 2008.

[9] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," in Commun ACM, 1975, pp. 333–340.

[10] Joachim Daiber, Rohana Rajapakse, Felix Sasaki, Christian Bizer Pablo N. Mendes, "Evaluating the Impact of Phrase Recognition on Concept Tagging," in Proceedings of the Eight International Conference on Language Resources and Evaluation LREC'12, Istanbul, Turkey, 2012.

[11] Oren, and Oren Etzioni Zamir, "Web document clustering: A feasibility demonstration [STC]," in Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, 1998, pp. 46-54.

[12] Stanisław, Jerzy Stefanowski, and Dawid Weiss Osiński, "Lingo: Search results clustering algorithm based on singular value decomposition," in Intelligent information processing and web mining, 2004, pp. 359-368.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[13] Eibe, et al. Frank, "Domain-specific keyphrase extraction [KEA]," in Proc. of the 16th international joint conference on AI, 1999.

[14] Lev-Arie, et al. Ratinov, "Local and Global Algorithms for Disambiguation to Wikipedia [NER + ngrams]," in ACL vol. 11, 2011.

[15] R. Troncy G. Rizzo, "Nerd: evaluating named entity recognition tools in the web of data," in Workshop on Web Scale Knowledge Extraction, 2011.

[16] H. Kaiya, Shinshu Univ., Nagano, Japan Fac. of Eng., and M. Saeki, "Ontology based requirements analysis: lightweight semantic processing approach," in Fifth International Conference on Quality Software (QSIC 2005), 2005.

Journal of Theoretical and Applied Information Technology
20th September 2014. Vol. 67 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

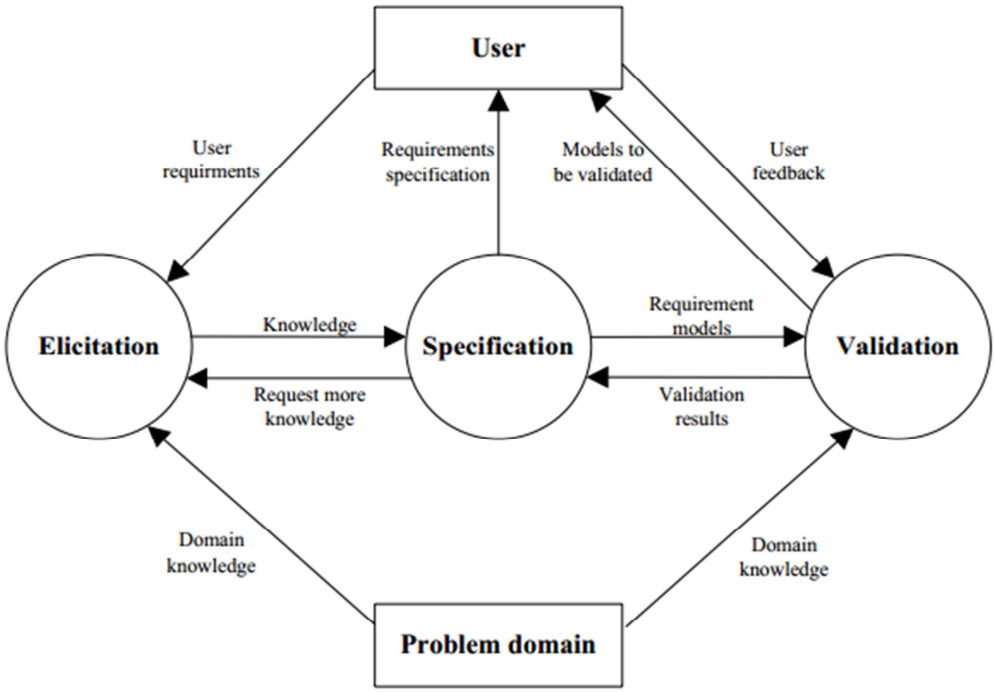

Figure 1: Requirements Engineering Consists Of Requirements Elicitation, Specification And Validation [7]