PAPER

# Autonomous Decentralized Semantic Based Traceability Link Recovery Framework*

**Khalid MAHMOOD**[†a)], ***Member***, **Mazen ALOBAIDI**[†], ***Nonmember***, ***and*** **Hironao TAKAHASHI**[††], ***Member***

**SUMMARY** The automation of traceability links or traceability matrices is important to many software development paradigms. In turn, the efficiency and effectiveness of the recovery of traceability links in the distributed software development is becoming increasingly vital due to complexity of project developments, as this include continuous change in requirements, geographically dispersed project teams, and the complexity of managing the elements of a project - time, money, scope, and people. Therefore, the traceability links among the requirements artifacts, which fulfill business objectives, is also critical to reduce the risk and ensures project's success. This paper proposes Autonomous Decentralized Semantic based Traceability Link Recovery (AD-STLR) architecture. According to best of our knowledge this is the first architectural approach that uses an autonomous decentralized concept, DBpedia knowledge-base, Babelnet 2.5 multilingual dictionary and semantic network, for finding similarity among different project artifacts and the automation of traceability links recovery.

***key words:*** *semantic web, NLP, Babelnet, GATE, DBpedia, document similarity*

## 1. Introduction

The development of large scale software has become geographically distributed in order to get the qualified resources at cheaper cost as well as to increase the efficiency in terms of time, space, and reliability. A Distributed Software Development (DSD) is software development model that distributes the development of product or service among remote sites [1]. In a geographical distributed software development environment, requirements specification becomes critical due to the characteristics of the temporal and spatial distances, and language/cultural differences [2], this, in turn, bring challenges like communication, collaboration and trust gap [3]–[5], asymmetry in processes, policies, standards, as well as several other factors that greatly influence project success.

The success of projects relies heavily on robust requirement management. As the backbone of requirements management, "Requirements Traceability" enables mapping of individual requirements artifacts with all other artifacts of the system (high/low level requirements, source code, quality control, etc). In addition, requirements traceability identifies and outlines the lineage of each requirement, apart from its backward traceability (derivation), its forward traceability (allocation) and its association with other project artifacts. Traceability, according to IEEE Standard Computer Dictionary, is defined as the degree to which a relationship can be established between two or more products of the development process [6]. Furthermore, the requirements traceability plays a vital role in ensuring the current requirements are met, traces the impact of any changes to the requirements, and defines the relationship among them and a delivered system which, in turn, ultimately lowers risk. Requirements traceability also reduces efforts for software maintenance, development, testing under constant evolving requirements, and implementation changes by developers till final release.

During development lifecycle of large scale products when teams are geographically distributed, it becomes impractical to update tractability links continuously. The inability to constantly update the links results in inconsistency in traceability matrix that ultimately results in project overruns, failure, and maintenance difficulty [7]. In the past researchers have focused on traceability link recovery for centralized software development environment by employing mainly Information Retrieval (IR) [8] techniques. However, these IR techniques [9], [10] obtain high recall [11] by linking each requirements to all source code entities (and other requirements artifacts) but precision [11] was reported almost close to zero. On other hand, few of these techniques [12] obtain high precision (almost 90%) by reporting only obvious links, but at cost of almost zero recall. Both of these extreme scenarios are undesirable as technical team members would then need to manually review generated candidates links to remove false positive by referring the updated source code and new requirement artifacts to recover missing links.

The main problem with IR techniques is that they don't consider knowledge of domain (application) and programming language used for development, and semantics of requirements particularly determining the context of term used, resolving ambiguity and vagueness present in requirements artifacts. Recently some semantic relatedness techniques have been applied for traceability link recovery, however, they are unable, a) to obtain high precision and recall mainly because of inaccurate multiple triplet extraction, b) making use of domain specific and cross domain knowledge, c) performing accurate disambiguation of polysemous

Manuscript received January 13, 2016.
Manuscript revised April 28, 2016.
Manuscript publicized June 7, 2016.
[†]The authors are with Oakland University, Rochester, USA.
[††]The author is with DTS Inc Japan, Tokyo, 108–0023 Japan.
*This work is based on "A Semantic Approach for Traceability Link Recovery in Aerospace Requirements Management System", by Khalid Mahmood et al. which appeared in Proc. IEEE International Symposium on Autonomous Decentralized Systems (ISADS 2015), Taichung, Taiwan, March 2015, © 2015 IEEE.
a) E-mail: mahmood@oakland.edu
DOI: 10.1587/transinf.2016EDP7018

terms, and d) to obtain accurate semantic similarity among various artifacts. Additionally, the centralized architecture for system like distributed software traceability bring the challenges of having online expansion, scalability, and fault tolerance in the system.

To address all these application and system challenges, AD-STLR architecture uses proposed a novel graph and semantic based disambiguation algorithm, triple extraction algorithm, algorithms to extract knowledge from Wikipedia, Babelnet, DBpedia, and semantic similarity algorithm. AD-STLR also employs Autonomous Decentralized System [13] based architecture to achieve scalability, online expansion, and fault tolerance (without introduction of single point of failure), and easy system maintenance. Thus, AD-STLR will allow distributed teams or organization units across the world to make their own decisions with the consideration of other teams by linking local artifacts to other teams' artifacts. In addition, the AD-STLR system improves the communication between distributed teams by allowing distributed teams to track each other's activities and exposing the impact of these activities on local projects. Similarly, our approach will make remote management more efficient by allowing project managers to manage, track and monitor schedule, cost, risks, quality, and preform integration management.

The main contributions of this paper are as follows:

1. proposes a new Autonomous Decentralized Semantic Traceability Links Recovery (AD-STLR) Framework to improve distributed teams collaboration by achieving same view of traceability matrix for all project managers based on requirement/project artifacts distributed in different geographical locations.
2. presents a new semantic relatedness base artifacts similarity and graph based disambiguation algorithms that uses DBpedia knowledge-base, and Babelnet 2.5 multilingual dictionary and semantic network for finding similarity among project/product artifacts.
3. unlike existing approaches [14], the proposed approach extracts multiple triplets from one sentence.

The rest of the paper is structured as follows: The subsequent section describes related works. Section 3 contains the preliminaries, and Sect. 4 contains the application. In Sect. 5, the architecture of AD-STLR is presented. The evaluation and results are shown and discussed in Sect. 6. Section 7 concludes the paper, and Sect. 8 gives future work.

## 2. Related Works

There are two effective methodologies applicable to facilitate Traceability Link Recovery that are related to our proposed approach, namely, Information Retrieval Techniques (IR) [15]–[18] and Semantic Relatedness Techniques [11], [19], [20].

### 2.1 Information Retrieval Techniques

Information Retrieval (IR) has been the subject of extensive debate as to its successes and failures in automated Traceability Links. IR is mainly defined as the task of discovering candidate traceability links on the basis of the similarity between software engineering artifacts that can be transformed in some unstructured text format [8]. Lei Liu et al. [17] measured word semantic similarity by using pattern vector space model where similarity between two words is calculated by the cosine of angle between their vectors. Antoniol, Giuliano et al. [21] proposed an approach that discovers traceability links between source code and software engineering artifacts based on Information Retrieval (IR) where software engineering artifacts are ranked against queries constructed from the identifiers of source code. Ali et al. [22] proposed an approach to establish and maintain traceability links between source codes and software requirements to improve the precision and recall of information retrieval (IR) techniques by discarding/re-ranking the reported links from IR based on the outcome links that are generated from software repositories. Diaz, Diana, et al. [23] proposed an approach that purifies the noise from the candidate links list generated by IR technique. This approach leverages the code ownership to improve the traceability link recovery. It uses the author's components and context to assess the candidate links. Marcus et al. [24] proposed approaches that advocate for the use of latent semantic indexing (LSI) to recover traceability links between documentation and source code. However, all the above proposed approaches based on IR, lack accuracy in term of precision, recall, and perform poorly in short context [22], [24], [25]. These approaches also disregard word order, syntactic relations, morphology, semantic relation, and word ambiguities [26].

### 2.2 Semantic Relatedness Techniques

Semantic Similarity and Semantic Relatedness have been the subjects of few studies that are associated with traceability link recovery. Semantic Similarity is usually defined by considering the lexical relations of synonymy, or equivalent words, and hyponymy, or the type-of relation [27]–[29]. Semantic Relatedness, on the other hand, extends the definition of similarity by examining all types of semantic relations that connect two concepts [11], [30], [31]. The Wu & Palmer measure calculates semantic similarity by attempting calculating the depths of the two synsets in the WordNet taxonomies, along with the depth of the Least Common Subsumer [20]. Zhang, Witte et al. proposed an approach that applies deep semantic similarity analysis based on the idea of ontology alignment [11]. This approach has four phases: building ontologies, modeling the domains of source code and software documents, creating a knowledge base by automatically populating these ontologies through code analysis and text mining, and finally establishing traceability links

between code and documents through ontology alignment. Falbo et al. attempted to extend semantic document management platform for the requirement domain by using semantic annotations in requirement documents [32]. Furthermore, there is a consideration of the conceptualization established by the proposed software requirements, with reference to its ontology and the generation of the traceability matrix, both of which are based on a dependency relationship and related axioms (reasons). Chan, Patrick, et al. [33] contributed to this semantic approach by improving the similarity measurement of low-frequency words and concepts. Furthermore, there is an exploration of the conceptualization established by computing the words locations in the wikipedia article and text style. Although, all of the above mentioned approaches improve accuracy, they have some limitations: only one database is used, which recovers traceability between source codes and software artifacts only, which in turn ignores word ambiguities.

## 3. Preliminaries

### 3.1 Babelnet

Babelnet [34] is a multilingual encyclopedic dictionary and utilizes different semantic knowledge-base, namely WordNet, Open Multilingual WordNet, Wikipedia, OmegaWik, Wiktionary, and Wikidata. In addition, its semantic network connects concepts and named entities in a very large network of semantic relations.

### 3.2 DBPedia

DBPedia, is a data-set extracted from Wikipedia and transformed to Resource Description Framework (RDF) model data, using the Notation3 (N3) data representation format [35]. In addition, DBPedia allows to run sophisticated queries against Wikipedia, and to link the different data sets on the Web to Wikipedia data.

### 3.3 Stanford NLP APIs

Stanford NLP APIs is a program that works out the grammatical structure of sentences by using the parser [36]. Note that similar to Stanford APIs we use the abbreviation of Part of Speech (POS) tagging that are widely used by NLP community, that are adopted from the Penn Treebank project. For convenience of readers, we mention the POS taggs used in our paper: NP (Noun phrase), NN (Noun, singular or mass), NNS (Noun, plural), NNP (Proper noun, singular), NNPS (Proper noun, plural), PRP (Personal pronoun), FW (Foreign word), WP (Wh-pronoun), RB (Adverb), VP (Verb phrases), VBN (Verb, past participle), VBZ (Verb, 3rd person singular present), VBG (Verb, gerund or present participle), VB (Verb, base form), VBD (Verb, past tense), VBP (Verb, non-3rd person singular present).

### 3.4 GATE

General Architecture for Text Engineering (GATE) is a development environment that provides a rich set of interactive tools for the creation, measurement and maintenance of software components for processing human language [37].

## 4. Application

AD-STLR can be applied in any distributed software development scenario [38], [39]. Our earlier work [40] mentions an application where AD-STLR can be applied on "Weather Services for Air Traffic Aviation in Detroit". This paper presents development of National Aeronautics and Space Administration (NASA) autonomous drone traffic management application. In this application all of its production units are geographically distributed with skilled workforce. However, distributed teams' presence poses an additional challenge to the development process, namely, effective communication, coordination, tracking and control, cooperation, and conformance with regulations and standards [32]. NASA strives to ensure that the application and the independent stakeholders' efforts in Software Development Life Cycle follow the Federal Aviation Administration's (FAA) governance policies and standards for Airspace. This requires frequent traveling of project managers, thus increases the risk of communication gap. This process is extremely costly in terms of time, resources, and risk. However, our proposed AD-STLR can offer an excellent solution because it increases effectiveness and efficiency while reducing costs and risks. Our approach can be used by product owners, product managers, business analysts, and developers alike in distributed product development.

### 4.1 Application & System Requirements

The application requirements include a) high accuracy: correct generation of Traceability matrix considering the continuous changing requirements, coding and testing artifacts. It is measured using recall and precision [11], and b) usability (user friendly).

The system requirements of AD-STLR are online-expansion, fault-tolerance with low complexities, and acceptable timeliness under dynamic conditions. Online expansion means that during process of addition of nodes, and during addition/updating of artifacts at any node the system should provide service with reasonable response time. Additionally, the joining process of new nodes should be done with low complexity, without stopping the system and maintain high-response (acceptable) for the system's users. Moreover, to cope with the dynamic changes in the network a fault-tolerance process is required that should be done with low complexity. Node failures must not lead to severely hampering the service provision. The decentralized nature of AD-STLR provides inherent fault-tolerance capability up to large extent.

## 5. AD-STLR Architecture

Distributed software development can be facilitated by having the same view of traceability matrix at all locations. In this regard we propose a traceability links recovery methodology that offers a clear and concise distributed project management and it is equally useful for all industry sectors and is very unique from existing models in the literature.

### 5.1 System Architecture

Figure 1 shows AD-STLR architecture that is based on Autonomous Decentralized System (ADS) concept [13], [41]. AD-STLR employs "*Content Code*" (CC) communication [42] to meet semantic interoperability. Figure 2 shows the format of the broadcasting message. The, "*Message ID*" uniquely identifies message and *CC* shows the type of the message: AR, UR, DR. "*Source Type*" specifies the source of the artifact e.g. requirements management tools, source control tools etc. to leverage semantic enabled searching. The parameter "*Artifact ID*" is used to determine the requirement identification, while "*Sender NodeID*" identifies the node that broadcasts the message. "*The Data parameter*" describes attributes of the artifact e.g. requirement name, requirement summery etc. AD-STLR uses three messages namely, Add Requirement (AR), Update Requirement (UR), and Delete Requirement (DR). Content code is registered at each node (peer). Each node may be in one of the following states: monitoring, listening, accepting, and processing. Below we illustrate in detail the autonomous operations and the description of states a node may acquire:

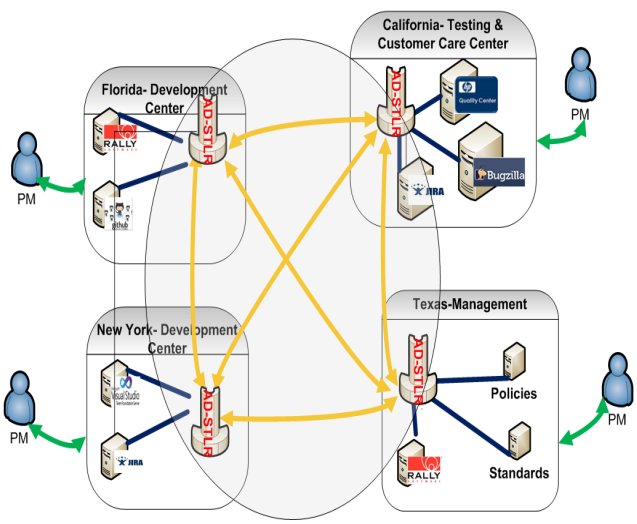


**Fig. 1** Autonomous decentralized semantic based on overlay network

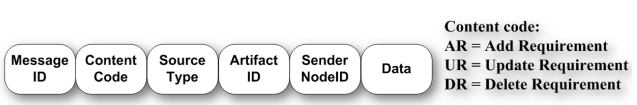


**Fig. 2** AD-STLR message format

1. Autonomous monitoring: each node constantly monitors the changes in requirement artifacts. As soon as the node detects that any requirement has been added, deleted, or updated, it generates corresponding message (AR, UR, or DR) and broadcasts to all nodes. Note that each node interacts with requirements management tools, source control tools, or quality management tools through given APIs/ interfaces to differentiate among addition, deletion, or updating of requirements.
2. Autonomous listening: during listening phase each node listens for the arrival of broadcasting messages and is based on predefined rules that enables the user to accept or reject the message autonomously.
3. Autonomous accepting: upon accepting the broadcasting message, the node autonomously examines its contents and registers *Message ID* and *Data* (Fig. 2)
4. Autonomous processing: in this state each node invokes the traceability link algorithm that measures the similarity among local requirement and relevant remotely located requirements, generates traceability links candidates, caches links candidates, and presents the traceability matrix in the form of a dashboard.

### 5.2 Node Architecture

A node is a machine having AD-STLR modules installed. As shown in Fig. 3, each node is comprised of seven components: "Natural Language Processing" (NLP), "Triple Extraction", "Triple Disambiguation", "Category Extraction", "Wiki Knowledge Extraction", "Inference", and "Aerospace Ontology".

### 5.3 NLP Module

The NLP module uses GATE framework [43] for tokenization, stemming, lemmatization [36], and Part-of-Speech Tagging [44]. Tokenization is process that splits the artifacts into tokens, where stemming and lemmatization are processes that convert or remove inflexional, derivational form to a common world form. Furthermore, the module exploits Stanford Parser [36] specifically LexicalizedParser function to construct a structure tree that can characterize each node as a noun phrase, verb phrase, or a full stop (.) and generates a syntactical structure of sentences. We then split the constructed tree into sub-trees based on specified tag, 'S', in our implementation (see Algorithm 1). Furthermore, NLP module has the task of passing the structure tree to the Triple Extraction Module.

Algorithm 1 illustrates the steps of how parsing tree and sub-trees are constructed. The triple-extraction function essentially takes the unstructured text and constructs the parsing tree using Stanford APIs (line 2). On other hand, function *extraction-trees* takes the parsing tree output from function *triple-extraction*, and splits the parsing tree into sub-trees based on 'S' node in line 5–6. In line 7–11, the

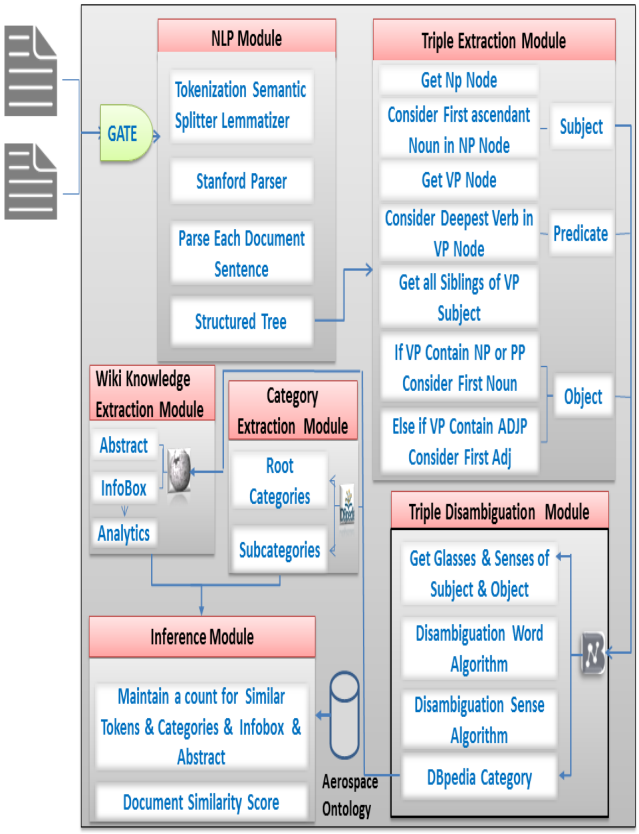


**Fig. 3** Autonomous decentralized semantic based traceability link recovery framework

**Algorithm 1** Pseudo code of NLP module

```
1: procedure TRIPLE-EXTRACTION(sentence)
2:     sentencetree ← find tree by using standford parser
3: end procedure
4: procedure EXTRACTION-TREES(SentenceTree)
5:     trees ← find S tag from sentenceTree
6:     trees ← find S tag from sentenceTree for multiple triples
7:     if trees = null then
8:         return sentenceTree
9:     else
10:        return trees
11:    end if
12: end procedure
```

sub-trees collection is examined and returned. However, the full complete input parsing tree is returned when the collection of sub-trees is empty.

### 5.4 Triple Extraction Module

The primary goal of this module is to extract Triples (subject, predicate, and object) from the document and pass them to the Triple Disambiguation Module. The subject is the resource that is being described by a predicate and an object. The object is either a resource referred to by a predicate or a literal value. The predicate is a relation between a subject and an object [45]. Our proposed algorithm is an extension

**Algorithm 2** Pseudo code of Subject Extraction

```
1: procedure EXTRACT-SUBJECT(tree)
2:     NP ← find NP from tree
3:     subject_i ← find NN, NNS, NNP, NNPS subject found in NP
4:     subject_j ← check again NN, NNS, NNP, NNPS for multiple
   subjects in NP
5:     if subject = EMPTY then
6:         subject_i ← find PRP, FW subject found in NP
7:         subject_j ← check again PRP, FW for multiple subjects in
   NP
8:     end if
9:     if subject = EMPTY then
10:        subject_i ← find WP, RB subject found in NP
11:        subject_j ← checkagain WP, RB for multiple subjects in NP
12:    end if
13:    if subject = EMPTY then
14:        return empty
15:    else
16:        return subject
17:    end if
18: end procedure
```

of "Triplet Extraction from Sentences" algorithm in [14] to cater: a) the recognition of multiple subjects and objects in one sentence b) consideration of multi-word (e.g. Information Technology) as one token. The process of this module is as follows:

#### 5.4.1 Subject Identification

To find the subject, we use the Noun Phrase (NP) sub-tree. Since the NP might have compound subjects, we search for all subjects NP sub-trees. If no subject is found in NP sub-tree, we search for all prepositions, and then all adverbs and Wh-adverbs.

The pseudo code in Algorithm 2 explains the subject extraction. EXTRACT-SUBJECT function takes a tree structure as input and invokes Breadth-first search (BFS) in line 2. In particular, we search for the Noun Phrase (NP) node. If found, line 3–4 extract the first node of Noun, singular or mass (NN) node or Noun, plural (NNS) node or Proper noun, singular (NNP) node or Proper noun, plural (NNPS) node and label it as subject. The same steps are repeated to find a second subject if it exists. Line 5 examines the subject: if it is not empty then it is returned, otherwise lines 6–8 invoke BFS searching for first Personal pronoun (PRP) node or foreign word (FW) as subject. The process is repeated again one more time to find a second subject. If not found, lines 10–12 invoke BFS searching for first Wh-pronoun (WP) or Adverb (RB). The search is repeated again for same types with second subject. Line 13 examines the subject and returns it if not empty.

#### 5.4.2 Predicate Identification

The Verb Phrases (VP) are obtained from the structured tree of the NLP module and the predicate is extracted from the deepest verb in the VP tree. The pseudo code in Algorithm 3 illustrates the predicate extraction. EXTRACT-

**Algorithm 3** Pseudo code of Predicate Extraction

```
1: procedure EXTRACT-PREDICATE(tree)
2:     VP ← find VP from tree
3:     if VP ! = null then
4:         predicate ← find deepest verb VBN, VBG, VBZ found in VPsubtree
5:         if predicate = null then
6:             predicate ← find deepest verb VB found in VPsubtree
7:             if predicate = null then
8:                 predicate ← find deepest verb VBD, VBP found in VPsubtree
9:                 if predicate = null then
10:                    predicate ← find deepest verb NN, NNS, NNP, NNPS found in VPsubtree
11:                    if predicate = null then
12:                        return Empty
13:                    end if
14:                end if
15:            end if
16:        end if
17:        return predicate
18:    end if
19:    return Empty
20: end procedure
```

PREDICATE function takes a tree structure as input and invokes BFS in line 2. In particular, we search for the VP node. Lines 3 examines the VP node if it is null, then predicate is returned empty, otherwise its deepest verb, namely, VBN, VBG, and VBZ are extracted using Depth-first search (DFS) (in line 4) and labeled as predicate. Lines 5–6 examine the predicate: if it is not empty then it is returned, otherwise, DFS is called to search for deepest VB. Once found it is labeled as predicate. Lines 7–8 repeat the same actions as in lines 5–6 but this time looking for deepest VBD or VBP and assign it as predicate. Finally, line 9–10 validates the predicate, if it is still empty then DFS is called again to search for NN or NNS or NNP or NNPS. If not found, then predicate is returned empty.

### 5.4.3 Object Identification

The Verb Phrases (VP) are obtained from the structured tree of the NLP module. To find the object, a search is performed to find the nouns first, then prepositions, and lastly adjectives. If no object is found then the next targets are past tense verbs in the VP tree. Algorithm 4 could be interpreted and understood further using explanation of for Algorithm 2 & Algorithm 3 (see Sects. 5.4.1 & 5.4.2).

## 5.5 Triple Disambiguation Module

The primary challenge of natural language processing is Word Sense Disambiguation (WSD) which is the task of identifying the correct meaning of the word or phrase within the context [46]. Our approach introduces a novel technique to identify which sense of the polysemous subject or object should be used in particular contexts by first fetching the senses and glosses of both from all resources of Babelnet (wordnet, Wikipedia, etc).

**Algorithm 4** Pseudo code of Object Extraction

```
1: procedure EXTRACT-OBJECT(tree)
2:     VP ← find VP from tree
3:     object_i ← find NN, NNS, NNP, NNPS, object found in VP
4:     object_j ← check again NN, NNS, NNP ,NNPS, object for multiple objects in VP
5:     if object = null then
6:         object_i ← find VBN, VB, PRP, VBP, RB, JJ found in VP
7:         object_j ← find VBN, VB, PRP, VBP, RB, JJ for multiple objects in VP
8:     end if
9:     if object = null then
10:        object_i ← find VBD object found in VP
11:        object_j ← check again VBD for multiple objects in VP
12:    end if
13:    if object = null then
14:        return empty
15:    else
16:        return object
17:    end if
18: end procedure
```

**Algorithm 5** Pseudo code of Disambiguation

```
1: procedure DISAMBIGUATE-WORDS(subject, subpostag, object, objpostag)
2:     bestsubjectsense ← most frequent sense for subject
3:     maxoverlap ← 0
4:     subjectsenses ← FETCH − SENSES (subject , subpostag)
5:     objectsenses ← FETCH − SENSES (object , objpostag)
6:     for each subsense in subjectsenses do
7:         for each objsense in objectsenses do
8:             subSignature ← set of glosses in the subsense
9:             objSignature ← set of glosses in the objsense
10:            overlap ← COMPUTEOVERLAP (MERGGLOSSES (subsignature) , MERGGLOSSES(objsignature))
11:            if overlap > maxoverlap then
12:                maxoverlap ← overlap
13:                bestsubjectsense ← subsense
14:                bestobjectsense ← objsense
15:            end if
16:        end for
17:    end for
18:    return (bestsubjectsens, bestobjectsense)
19: end procedure
20: procedure MERGGLOSSES(sigsense)
21:    tokens ← sigSense − wordNet gloss
22:    tokens ← sigSense − wikitionary gloss
23:    tokens ← sigSense − omegaWiki gloss
24:    tokens ← sigSense − wikiData gloss
25:    tokens ← sigSense − wikipedia gloss
26: end procedure
```

Algorithm 5 shows the pseudo code and Fig. 4 shows the overall process of our graph based approach of disambiguation. First, we enrich each polysemous subject and the objects for disambiguation by getting their senses (Lines 4 and 5 of algorithm) from all resources of Babelnet. The subject senses and object senses are then processed in nested for loops as shown in line 6–17, where we find the similarity between every sense of subject with every sense of object by looking at the overlapped number of tokens in glosses. Note that we send glosses of each sense to the NLP module to extract the triples before calling the LESK

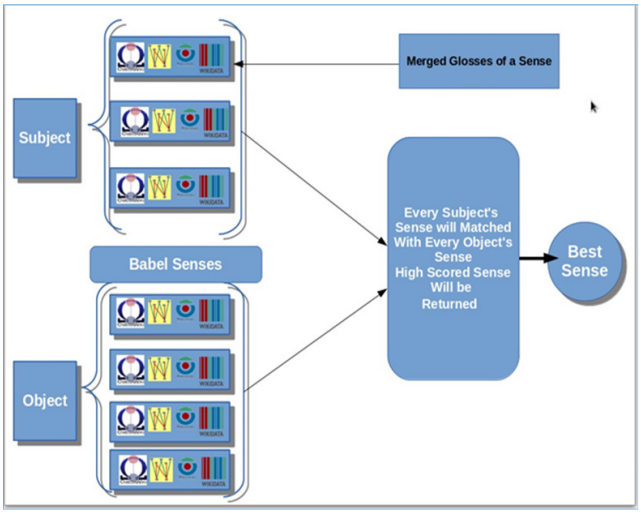


**Fig. 4** Disambiguation using various resources of Babelnet

algorithm [47]. In Fig. 4, merged glosses of each sense after NLP process on all senses obtained from resources of Bablent have been shown and are also represented in procedure MERGGLESSES (line 20–26) in pseudocode. Once the composite context using all resources of Bablnet has been defined, we use LESK similarity [47] to disambiguate senses of polysemous subjects and objects. In the pseudo code of Algorithm 5, LESK procedure was called in line 10.

### 5.6 Category Extraction Module

This module preforms following tasks: accepts the DBpedia category from Disambiguation module for each sense, finds the super-category and subcategory of the input category from DBpedia, fetches DBpedia categories for each disambiguated sense, and passes the result to the Inference Module (see Algorithm 6).

### 5.7 Wiki Knowledge Extraction Module

Algorithm 7 shows Wiki Knowledge Extraction Module that focuses on three primary tasks:

1. Inquiring wiki page: For each term in the document, we construct a unique URL that retrieves the corresponding wiki page for given term
2. Parsing Wiki page: the fundamental task of this step is to parse HTML and extract the abstract and infobox of the retrieved wiki page
3. Analytics: the primary task of this process is maintaining statistical info of matching terms of artifact N with M associated entities found in infobox and abstract of wiki page
4. Passing the calculating statistics to the Inference Module

**Algorithm 6** Pseudo code of Category Extraction

```
1: procedure GET_CATEGORIES (name_entity)
2:     categories ← find name entity categories using Babelnet
3:     RETURN categories
4: end procedure
5: procedure GET_BROADER_CATEGORIES (category)
6:     broader_categories ← find broader categories using DBpedia
   SKOS relationship
7:     RETURN broader_categories
8: end procedure
9: procedure GET_BROADER_CATEGORIES (category)
10:    sub_categories ← find broader categories using DBpedia
   SKOS relationship
11:    RETURN broader_categories
12: end procedure
13: procedure GET_SUB_CATEGORIES (category)
14:    broader_categories ← find sub categories using DBpedia
   SKOS relationship
15:    RETURN sub_categories
16: end procedure
```

**Algorithm 7** Pseudo code of Wiki Knowledge Extraction

```
1: procedure WIKI_KNOWLEDGE (entity)
2:     URL ← constructURL(Word)
3:     WikiPage ← getWikiPage(URL)
4:     HtmlDoc ← ParseHTML(wikiPage)
5:     Abstract_entities ← JerichoAPI_Extract(HtmlDoc))
6:     Infobox_entity[] ← JerichoAPI_Extract(HtmlDoc))
7:     RETURN Abstract_entities[], Infobox_entity[])
8: end procedure
```

### 5.8 Aerospace Ontology Module

We have developed a domain specific aerospace ontology which has only 200 concepts. This ontology contains only disambiguation knowledge and it uses three relationships: rdfs:label, rdf:type, and hasdomain. For example, Aerospace_manufacture rdfs:label “Lockheed Martin”, Aerospace_manufacture rdf:type “aerospace industry”, and Aerospace_manufacture hasdomain “aerospace”.

Aerospace domain knowledge is extracted by inference engine module of AD-STLR using Jena API [48]. As Aerospace ontology contains only disambiguate knowledge, therefore, inference engine knowledge, therefore, inference engine first refer this knowledge before considering general purpose ontology (See Algorithm 8, lines 4–9) of our system.

### 5.9 Inference Module

In inference engine, we first employ disambiguated knowledge of Aerospace ontology to quickly retrieve the category of any given token. After token is retrieved, firstly we extract “label” against given Name Entity, next “type” against resultant label is extracted to get category class. Finally, “hasDomain” relationship is used against the obtained type in previous step to get a resultant domain category. If the token is not inferred at this level, this inference module uses

**Algorithm 8** Pseudo code of Inference Algorithm

```
1: procedure SEMANTIC-SIMILARITY(Artifact1Triplets[], Artifact2Triplets[])
2:     //Triplet[] contains triplet alog with POS Tags.
3:     for each triple in artifcat1 and artifact2 do
4:         typesubj1 = Gettype(GetlabelAerospaceonto(subjectartifct1))";
5:         typeobj1 = Gettype(GetlabelAerospaceonto(Objectartifct1))";
6:         typesubj2 = Gettype(GetlabelAerospaceonto(subjectartifct2))";
7:         typeobj2 = Gettype(GetlabelAerospaceonto(Objectartifct2))";
8:         domain1 = GetDomain(typesubj1, typeobj1)
9:         domain2 = GetDomain(typesubj2, typeobj2)
10:        if domain1 = domain2 then
11:            semantic_similarity_score + I;
12:            break;
13:        end if
14:        Artifact1_categories ← Get_CATEGORIES(subjectArtifact1
    ,objectArtifact1);
15:        Artifact2_categories ← Get_CATEGORIES(subjectArtifact2
    ,objectArtifact2);
16:        Artifact1_wikiknowledge[] ← Wiki_KNOWLEDGE (subject
    _artifact1 , object_artifact1);
17:        Artifact2_wikiknowledge[] ← Wiki_KNOWLEDGE (subject
    _artifact2 , object_artifact2);
18:        semantic_Similarity_Score ← 0;
19:        for each t_cat in Artifact1_categories do
20:            CURRENT :
21:            for each n_cat in Artifact2_categories do
22:                broader_categoriesgets ←
23:                GET_BROADER _CATEGORIES(n_cat);
24:                sub_categories ← GET _SUB _CATEGORIES(n_cat);
25:                if t_cat = n_cat then
26:                    semantic_similarity_score ←
27:                    semantic_similarity_score + I;
28:                    continue CURRENT;
29:                end if
30:                for each b_cat in broader_categories do
31:                    if t_cat = b_cat then
32:                        semantic_similarity_score ←
33:                        semantic_similarity_score + J;
34:                        continue CURRENT;
35:                    end if
36:                end for
37:                for each s_cat in sub_categories do
38:                    if t_cat = s_cat then
39:                        semantic_similarity_score ←
40:                        semantic_similarity_score + K;
41:                        continue CURRENT;
42:                    end if
43:                end for
44:            end for
45:        end for
46:    end for
47:    for each pred_artifact1 in Artifact_1 Triplets do
48:        for each spred_artifact2 in Artifact_2 Triplets do
49:            if pred_artifact1 = pred_artifact2 then
50:                semantic_similarity_score ←
51:                semantic_similarity_score + M;
52:            end if
53:        end for
54:    end for
55:    for each nameentity_artifact1 in subject_artifact1[] and
    object_artifact1[] do
56:        artifact1subject_wikiknowledge_LIST = Wiki_KNOWLEDGE
    (subject_artifact1[]);
57:        artifact1object_wikiknowledge_LIST = Wiki_KNOWLEDGE
    (object_artifact1[]);
58:        artifact2subject_wikiknowledge_LIST = Wiki_KNOWLEDGE
    (subject_artifact1[]);
59:        artifact2object_wikiknowledge_LIST = Wiki_KNOWLEDGE
    (object_artifact2[]);
60:        if subject_artifact1[ nameentity_artifact1 ] OR Object_artifact1
    [nameentity_artifact1] == Artifact2subject_wikiknowledge_LIST
    OR Artifact2subject _wikiknowledge_LIST then
61:            semantic_similarity_score ←
62:            semantic_similarity_score + R;
63:            if subject_artifact1[ nameentity_artifact1 ]OR
    Object_artifact1 [nameentity_artifact1] ==
    Artifact1subject_wikiknowledge_LIST OR
    Artifact1object _wikiknowledge_LIST then
64:                semantic_similarity_score ←
65:                semantic_similarity_score + S;
66:            end if
67:        end if
68:    end for
69:    RETURNsemantic_similarity_score
70: end procedure
```

DBPedia knowledge.

We maintain a count for the overlapping of DBpedia categories and then calculate similarity score. In addition, it generates candidates for traceability links. Algorithm 8 shows the pseudo code of our weighting scheme. Here we assume that I >= J >= K and M >= R >= S. In the semantic similarity algorithm we have defined three levels for similarity measure namely *current* (most similar), *root* (least similar) and *sub levels* (moderate similar). We first determine the subject's and object's categories of all artifacts using Babelnet (mainly DBpedia). Next for each subject and object of first artifact we compare it with the category of each subject and object of remaining artifacts. For the sake of simplicity lets assume that there are only two artifacts. If the categories are matched at this level (i.e. current level) then we increment the similarity score by the value of *I* (line 22–29). Consider for example that the subject Florida is in artifact 1 and the artifact 2, has the subject Arizona. The category for both Florida and Arizona would be "States of the United States". Therefore a match, at this level, increments the similarity score by I. If the matching fails at this level then the next level is the sub category level where we determine the sub-categories of the artifact 1 subject and objects (entities). We then compare each of these sub-categories with the artifact 2 entity category already in process. A match, at this level, increments the similarity score by *J* (line 30–36). If we continue our previous example then the sub-categories for Arizona can be "Geography of Arizona", "Education in Arizona" etc. If we are not successful in matching the categories at the *current level* then we move forward to the next level, the root level. In this level we determine the broader categories for the first artifact. Next each broader category is matched with the category of the artifact 2 subject and object already in process. If the categories are matched at this level (i.e. root level), we increment the similarity score by value *K* (line 37–34). If we continue our previous example of Florida and Arizona the root category would be "United States", and therefore a match, at this

level, increments the similarity score by K. In case of failure of similarity matching using name entities at the above defined three levels (i.e. current, root and sub levels) we perform keyword matching of predicates of both artifacts and in case of a match, increment the similarity score by *M* (line 47–54). Lastly, we also extract the Infoboxes of all subjects and objects of artifacts and perform keyword matching with subjects and objects of other artifacts and their entities obtained from infoboxes. If match is found then constants S and R are added in semantic scores respectively. Note that DBpedia does not provide updated information against an entity because there is a certain timeline to update the information in the dump. For the sake of retrieving any information that belongs to the recent time, we parsed Wikipedia pages using Jericho HTML Parser [49] to get entities from abstract and infoboxes.

## 6. Evaluation

This section evaluates application requirements of AD-STLR: correct generation of traceability links using precision and recall [11] measures. Precision is the ratio of the number of true positive links retrieved over the total number of links retrieved, whereas, recall is the ratio of the number of true positive links retrieved over the total number of true positive links in control set. The evaluation and comparison were carried out as follows:

1. upon detecting the new/modified/deleted requirement in any node, the system selects relevant requirement artifacts
2. retrieves the similarly threshold set by local product manager (administrator)
3. reads requirement by requirement from the first set
4. loops through each requirement in selected sets
5. calculates the semantic similarity among selected requirements artifacts
6. collects and analyzes the results of the running experiments against existing answer sets
7. uses the results information to calculate precision and recall for evaluation

Evaluation os AD-STLR was carried out using General Architecture and Engineering Text (GATE), Stanford NLP APIs [36], Babelnet 2.5, and DBpedia. In addition, we conducted experiments by using 4-core Intel(R) Core(TM) i7-2720QM CPU @ 2.20GHz and 64 bits Java JVM.

### 6.1 DataSets

In order to simulate a real world scenario, we have selected two different datasets: MODIS & CM-1 [50]. To evaluate AD-STLR, we have chosen one benchmark algorithm from Information retrieval (IR) category while other one from semantic relatedness. Vector Space Model (VSM) [51] was chosen as benchmark because its average precision and recall is significantly better than others algorithms from IR class of traceability link recovery algorithms [7], [9], [10], [12]. Similarly Wu Palmer [20] was chosen for comparison as its precision and recall is close to human judgments and has high performance compared to other semantic similarity algorithms [11], [52], [53]. The details of datasets used are listed in Table 1.

**Table 1** DataSets

| DataSet | High Req | Low Req | Traces |
|---|---|---|---|
| ModisDataset | 19 | 49 | 41 |
| CM-1 | 235 | 220 | 361 |

#### 6.1.1 MODIS

The NASA Moderate Resolution Spectrometer (MODIS) dataset [50] is a small dataset built from the full Design (high- and low-level requirements documents) for the MODIS space instrument software. This dataset includes 19 high-level requirements, 49 low-level requirements, and a validated true positive 41 links that we refer to as the "True traces".

#### 6.1.2 CM-1

The dataset includes an exhaustive requirement and design document for a NASA space instrument [50]. The dataset contains 235 high level and 220 low-level requirements. The trace for the dataset was manually verified. The "theoretical true trace" (answer-set) built for this dataset consisted of 361 correct links. Each of the high and low-level files contain the text of one requirement element.

### 6.2 Results

The preliminary results show that both AD-STLR and the benchmark algorithms are suitable for the problem of recovering traceability links among software artifacts, however AD-STLR achieves higher values of precision and recall with higher threshold values towards 100 percent of precision and recall.

A possible explanation lies in the nature of the three models. The similarity measure of a VSM [51] is low because it only takes into account the terms that appear in both software artifacts, weight the frequencies of their occurrences, disregard their order and semantic relation, don't consider knowledge of domain (application), and disambiguate. On the other hand, Wu Palmer associates software artifacts based on the depth of two synsets in the WordNet taxonomies and ignore word ambiguities. Conversely, AD-STLR model links software artifacts based on semantic relation, word ambiguities using ontologies such as Babelnet, and DBpedia.

Our analysis shows that Wu Palmer performs very low in term of precision and recall mainly because of knowledge available in wordnet, therefore, our quantitative analysis mainly focuses on comparisons of AD-STLR and VSM. Figure 5 shows the comparison of AD-STLR, VSM, and Wu Palmer in term of precision where X-axis represents the

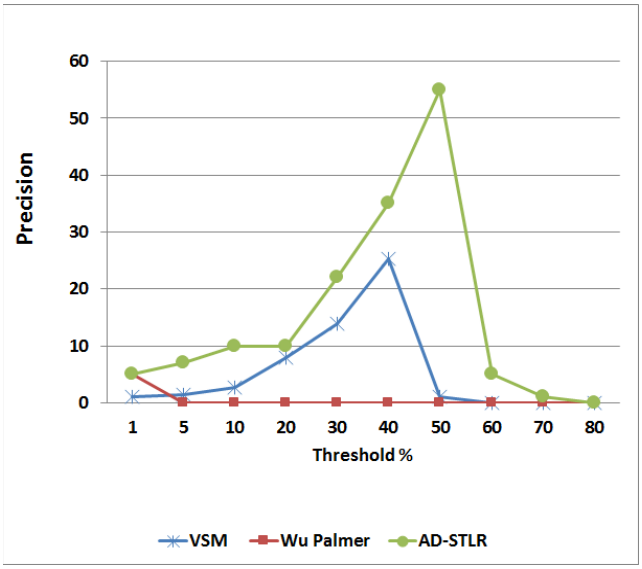


**Fig. 5** Comparison: precision of AD-STLR against VSM and Wu Palmer

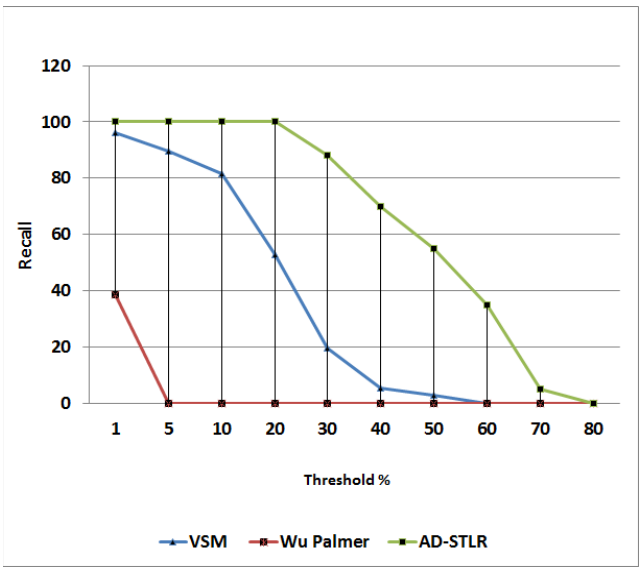


**Fig. 6** Comparison: recall of AD-STLR against VSM and Wu Palmer

threshold percentage that we used to retrieve all documents. It can be seen that AD-STLR outperforms VSM and Wu Palmer algorithms on all threshold points. Also, increasing the threshold value (up to 50%) leads to better precision for AD-STLR and VSM because threshold value has direct impact on precision. For example, when the threshold is set to 30%, our proposed approach generates a higher number of positive links retrieved such as an approximately 7 percent improvement compared to VSM and 22 percent to Wu Palmer approach. Our results show precision drops significantly when we increase the threshold beyond 50% for AD-STLR and 40% for VSM, whereas Wu Palmer dropped to zero after 5%. When threshold is increased by 5%, our quantitative analysis shows that there is on average 8 percent improvement in precision for AD-STLR and 4% in the case of VSM respectfully.

As shown in Fig. 6, lowering the threshold leads to higher recall because the total number of true positive links is independent from the threshold value. In addition, when threshold was varied from (0–20%), the recall for AD-STLR is 100% and starts to drop afterwords. On the other hand, for VSM recall is above than 80% when threshold is up to 10%, and then it starts to deteriorate significantly.

When precision and recall are plotted against recall, (not shown in graph) the area under the curve has maximum value when threshold is 50%: AD-STLR is returning accurate results (high precision), as well as returning a majority of all positive results (high recall).

## 7. Conclusion

Projects are constantly updated and modified in light of new risks and developing products. Traceability links are a primary part of requirements management for software development. Since Automated Traceability Links are an important element to ensure the success of software engineering projects, our proposed framework helps projects meet business requirements, improve the precision and recall of traceability links between requirements artifacts, and increase the success of projects by improving time, cost, risk, and quality.

## 8. Future Work

Currently we are extending our Aerospace ontology to further improve the the efficiency and effectiveness of our approach. Additionally, we will extend our semantic approach to include rules to ensure that AD-STLR should not only create traceability matrix but also ensure that documentation follow standards and regulation requirements. Furthermore, exhaustive comparison with other semantic related algorithms will be performed, in future we will add source code files in our model too.

### References

[1] T.W. Malone et al., The future of work, Audio-Tech Business Book Summaries, Incorporated, 2004.
[2] M. Webster, "The requirements for managing the geographically distributed development organization and the collabnet solution," White Paper, IDC, 2005.
[3] U.M. Apte, M.G. Sobol, S. Hanaoka, T. Shimada, T. Saarinen, T. Salmela, and A.P. Vepsalainen, "Is outsourcing practices in the usa, japan and finland: a comparative study," Journal of information technology, vol.12, no.4, pp.289–304, 1997.
[4] K. Ketler and J. Walstrom, "The outsourcing decision," International journal of information management, vol.13, no.6, pp.449–459, 1993.
[5] J. Dibbern, T. Goles, R. Hirschheim, and B. Jayatilaka, "Information systems outsourcing: a survey and analysis of the literature," ACM SIGMIS Database, vol.35, no.4, pp.6–102, 2004.
[6] A. Geraci, F. Katki, L. McMonegal, B. Meyer, J. Lane, P. Wilson, J. Radatz, M. Yee, H. Porteous, and F. Springsteel, IEEE standard

computer dictionary: Compilation of IEEE standard computer glossaries, IEEE Press, 1991.
[7] N. Ali, Analysing Source Code Structure and Mining Software Repositories to Create Requirements Traceability Links, Ph.D. thesis, École Polytechnique de Montréal, 2012.
[8] A. Qusef, G. Bavota, R. Oliveto, A.D. Lucia, and D. Binkley, "Evaluating test-to-code traceability recovery methods through controlled experiments," Journal of Software: Evolution and Process, vol.25, no.11, pp.1167–1191, 2013.
[9] A. Abadi, M. Nisenson, and Y. Simionovici, "A traceability technique for specifications," The 16th IEEE International Conference on Program Comprehension, pp.103–112, IEEE, 2008.
[10] S.K. Sundaram, J.H. Hayes, and A. Dekhtyar, "Baselines in requirements tracing," ACM SIGSOFT Software Engineering Notes, pp.1–6, 2005.
[11] A. Budanitsky and G. Hirst, "Evaluating wordnet-based measures of lexical semantic relatedness," Computational Linguistics, vol.32, no.1, pp.13–47, 2006.
[12] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia, "On the equivalence of information retrieval methods for automated traceability link recovery," 2010 IEEE 18th International Conference on Program Comprehension (ICPC), pp.68–71, IEEE, 2010.
[13] T. Koga, X. Lu, and K. Mori, "Autonomous decentralized high-assurance surveillance system for air traffic control," 2014 IEEE 15th International Symposium on High-Assurance Systems Engineering (HASE), pp.154–157, 2014.
[14] D. Rusu, L. Dali, B. Fortuna, M. Grobelnik, and D. Mladenic, "Triplet extraction from sentences," Proc. 10th International Multiconference Information Society-IS, pp.8–12, 2007.
[15] D.L. Lee, H. Chuang, and K. Seamons, "Document ranking and the vector-space model," IEEE Softw., vol.14, no.2, pp.67–75, 1997.
[16] U.L.D.N. Gunasinghe, W.A.M. De Silva, N.H.N.D. de Silva, A.S. Perera, W.A.D. Sashika, and W.D.T.P. Premasiri, "Sentence similarity measuring by vector space model," International Conference on Advances in ICT for Emerging Regions (ICTer), pp.185–189, 2014.
[17] L. Liu, M. Zhong, and R. Lu, "Measuring word similarity based on pattern vector space model," International Conference on Artificial Intelligence and Computational Intelligence, AICI'09, pp.72–76, 2009.
[18] V.V. Raghavan and S.K.M. Wong, "A critical analysis of vector space model for information retrieval," Journal of the American Society for information Science, vol.37, no.5, pp.279–287, 1986.
[19] S.J. Green, "Building hypertext links by computing semantic similarity," IEEE Trans. Knowl. Data Eng., vol.11, no.5, pp.713–730, 1999.
[20] Z. Wu and M. Palmer, "Verbs semantics and lexical selection," Proc. 32nd annual meeting on Association for Computational Linguistics, pp.133–138, Association for Computational Linguistics, 1994.
[21] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," IEEE Trans. Softw. Eng., vol.28, no.10, pp.970–983, 2002.
[22] N. Ali, Y.-G. Gueheneuc, and G. Antoniol, "Trustrace: Mining software repositories to improve the accuracy of requirement traceability links," IEEE Trans. Softw. Eng., vol.39, no.5, pp.725–741, 2013.
[23] D. Diaz, G. Bavota, A. Marcus, R. Oliveto, S. Takahashi, and A. De Lucia, "Using code ownership to improve ir-based traceability link recovery," IEEE 21st International Conference on Program Comprehension (ICPC), pp.123–132, 2013.
[24] A. Marcus, J.I. Maletic, and A. Sergeyev, "Recovery of traceability links between software documentation and source code," International Journal of Software Engineering and Knowledge Engineering, vol.15, no.05, pp.811–836, 2005.
[25] D. Metzler, S. Dumais, and C. Meek, Similarity measures for short segments of text, Springer, Berlin Heidelberg, 2007.
[26] Y. Zhang, R. Witte, J. Rilling, and V. Haarslev, "Ontological approach for the semantic recovery of traceability links between software artefacts," IET software, vol.2, no.3, pp.185–203, 2008.
[27] P. Resnik, "Using information content to evaluate semantic similarity in a taxonomy," arXiv preprint cmp-lg/9511007, 1995.
[28] J.J. Jiang and D.W. Conrath, "Semantic similarity based on corpus statistics and lexical taxonomy," arXiv preprint cmp-lg/9709008, 1997.
[29] P. Resnik et al., "Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language," J. Artif. Intell. Res. (JAIR), vol.11, pp.95–130, 1999.
[30] T. Pedersen, S. Patwardhan, and J. Michelizzi, "Wordnet:: Similarity: measuring the relatedness of concepts," Demonstration papers at hlt-naacl, Association for Computational Linguistics, pp.38–41, 2004.
[31] E. Gabrilovich and S. Markovitch, "Computing semantic relatedness using wikipedia-based explicit semantic analysis," IJCAI, pp.1606–1611, 2007.
[32] R. de Almeida Falbo, C.E.C. Braga, and B.N. Machado, "Semantic documentation in requirements engineering," 17th Workshop on Requirements Engineering (WER 2014), Pucón, Chile, 2014.
[33] P. Chan, Y. Hijikata, T. Kuramochi, and S. Nishida, "Semantic relatedness estimation using the layout information of wikipedia articles," International Journal of Cognitive Informatics and Natural Intelligence (IJCINI), vol.7, no.2, pp.30–48, 2013.
[34] R. Navigli and S.P. Ponzetto, "Multilingual WSD with just a few lines of code: the BabelNet API," Proc. 50th Annual Meeting of the Association for Computational Linguistics (ACL), Jeju, Korea, 2012.
[35] M. Völkel, M. Krötzsch, D. Vrandecic, H. Haller, and R. Studer, "Semantic wikipedia," Proc. 15th international conference on World Wide Web, ACM.
[36] "The stanford NLP (natural language processing) group."
[37] "GATE.ac.uk - biz/usps.html."
[38] M.B. Blake, G. Hamilton, and J. Hoyt, "Using component-based development and web technologies to support a distributed data management system," Annals of Software Engineering, vol.13, no.1-4, pp.13–34, 2002.
[39] D. Feldkamp, W. Siberski, B. Thönssen, and H. Wache, "E-government for distributed autonomous administrations," AAAI Spring Symposium: AI Meets Business Rules and Process Management, pp.28–38, 2008.
[40] K. Mahmood, H. Takahashi, and M. Alobaidi, "A semantic approach for traceability link recovery in aerospace requirements management system," ISADS IEEE, 2015.
[41] K. Mahmood, L. Xiaodong, Y. Horikoshi, and K. Mori, "Autonomous pull-push community construction technology for high--assurance," IEICE Trans. Inf. & Syst., vol.E92-D, no.10, pp.1836–1846, 2009.
[42] I.-L. Yen, R. Paul, and K. Mori, "Toward integrated methods for high-assurance systems," Computer, vol.31, no.4, pp.32–34, 1998.
[43] H. Cunningham, "GATE, a General Architecture for Text Engineering," Computers and the Humanities, vol.36, pp.223–254, 2002.
[44] M.P. Marcus, M.A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of english: The penn treebank," Computational linguistics, vol.19, no.2, pp.313–330, 1993.
[45] G. Klyne and J.J. Carroll, "Resource description framework (RDF): Concepts and abstract syntax," W3C Recommendation, 2005.
[46] N. Ide and J. Véronis, "Introduction to the special issue on word sense disambiguation: the state of the art," Computational linguistics, vol.24, no.1, pp.2–40, 1998.
[47] S. Banerjee and T. Pedersen, "An adapted lesk algorithm for word sense disambiguation using wordnet," in Computational linguistics and intelligent text processing, pp.136–145, Springer, 2002.
[48] B. McBride, "Jena: Implementing the rdf model and syntax specification.," SemWeb, 2001.
[49] H. Jericho, "Parser," 2011.
[50] J. Sayyad Shirabad and T. Menzies, "The PROMISE Repository of Software Engineering Databases," School of Information Technol-

ogy and Engineering, University of Ottawa, Canada, 2005.
[51] A.D. Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artifact management systems using information retrieval methods," ACM Trans. Softw. Eng. Methodol., vol.16, no.4, 2007.
[52] C. Corley and R. Mihalcea, "Measuring the semantic similarity of texts," Proc. ACL workshop on empirical modeling of semantic equivalence and entailment, pp.13–18, Association for Computational Linguistics, 2005.
[53] D. Lin, "An information-theoretic definition of similarity," ICML, pp.296–304, 1998.

**Khalid Mahmood** is currently working as an assistant professor at Oakland University, Michigan USA. Dr. Mahmood's research interests include integrated area of Internet of Things, cognitive mobile computing and semantic web. Research topics comprises of Semantics based Sensor Web, Semantic based Information Security & Data Loss Prevention, Autonomous Decentralized Systems, Big Data analytics using Linked data, Semantic based web filtering and Semantic based Cloud Robotics.

**Mazen Alobaidi** is a Ph.D. student at Department of Computer Science Engineering in Oakland University, USA. His research interests include autonomous decentralized systems, distributed application architecture and Semantic Web technologies.

**Hironao Takahashi** is Prof. ORIC Greenwich University. He received PhD degree in Computer Science in 2010 from Tokyo Institute of Technology and received the MS degree in MOT in 2006 Tokyo University of Science. His research area is High speed I/O system architecture on Autonomous Decentralized System. He invented Data Transmission System technology architecture and is holding nine patents of this field. He is member of IEEE, IEICE, IEE and IPSJ.