Traceability Challenge 2013: Query+ Enhancement for Semantic Tracing (QuEST)

Software Verification and Validation Research Laboratory (SVVRL) of the University of Kentucky

Wenbin Li, Jane Huffman Hayes
Computer Science Department
University of Kentucky
Lexington, Kentucky, USA
wenbin.li@uky.edu, hayes@cs.uky.edu

***Abstract*—We present the process and methods applied in undertaking the Traceability Challenge in addressing the Ubiquitous Grand Challenge, Research Project 3. Terms contained within queries (along with document collection terms, hence the "+") have been enhanced to include semantic tags that indicate whether a term represents an action or an agent. This information is obtained by calling the Senna semantic role labeling tool. The standard TF-IDF component in TraceLab is then used to recover trace links. The QuEST method was applied to four datasets. Results based on the provided answer sets show that QuEST improved Mean Average Precision (MAP) for two artifact pairs of two of the datasets when the artifacts used natural language, but generally did not outperform unaugmented datasets not using natural language. We provide insights on this finding.**

***Index Terms*—Traceability, Semantics, Trace Recovery, Challenge, Ubiquitous, Ubiquitous Grand Challenge, Research Project 2**

I. Introduction

As part of developing traceability that is ubiquitous (Ubiquitous grand challenge), we seek to automatically generate trace links of high quality (research project 3, "Total automation of trace creation and trace maintenance, with quality and performance levels superior to manual efforts."[1]). Toward that end, we look to semantics for assistance. The topic of semantics has long been discussed and studied in the traceability community: semantics of trace links, semantics of textual content, semantics of context, etc. Basically, *semantics* refers to our attempts to understand the meaning of the items with which we work.

When focusing on trace link recovery or generation, researchers have attempted to utilize or capture semantic meaning a number of different ways. Researchers have applied thesauri to augment the terms or words found in source (queries) and target (document collection) artifacts being traced - generally adding synonyms, acronyms, phrases, and/or addressing polysemy [1-8]. Researchers have applied techniques that attempt to understand the meaning or concepts inherent in text such as topic modeling, Latent Semantic Indexing (LSI), and Latent Dirichlet Allocation (LDA) [9-12].

To illustrate the importance of semantic information, consider the following. A textual requirement describes the behavior of an entity that will be part of the developed software system, such as "*the nodes shall forward all messages to the server*," while a target design element mentions the entity "in passing" in the text, such as "*the system contains x nodes*." In this case, it is less likely that there is a link between these two artifacts. Current non-semantic tracing methods may use the co-occurrence of the entity name as evidence of a link simply because the term appears in both artifacts.

Researchers have also used parts of speech (POS) tagging and have written rules to generate traceability links [13]. In this work, we go one step beyond POS tagging and attempt to understand the roles that various portions of an artifact element play in order to add semantic information to the source and target artifacts. POS information can indicate that a noun has been identified; it cannot indicate that the noun is an agent that can perform a specific action; it cannot identify the actions that an agent can perform. To accomplish these, we enhance the artifacts by adding semantic role information. Semantic role labeling refers to the identification and labeling of arguments in text [14]. Specifically, we annotate the agents within natural language textual statements that form the source and target artifacts. We also expanded our method by labeling the action verbs in the text. However, the performance was not improved with the action verbs labeled.

The paper is organized as follows. Section 2 discusses how we enhanced the source and target elements as well as how we used TraceLab to accomplish our work. Section 3 presents the results. Finally, we present conclusions and future work in Section 4.

[1] http://www.coest.org/index.php/research-directions/grand-traceability-challenges?view=gctchallenge&challengeId=12

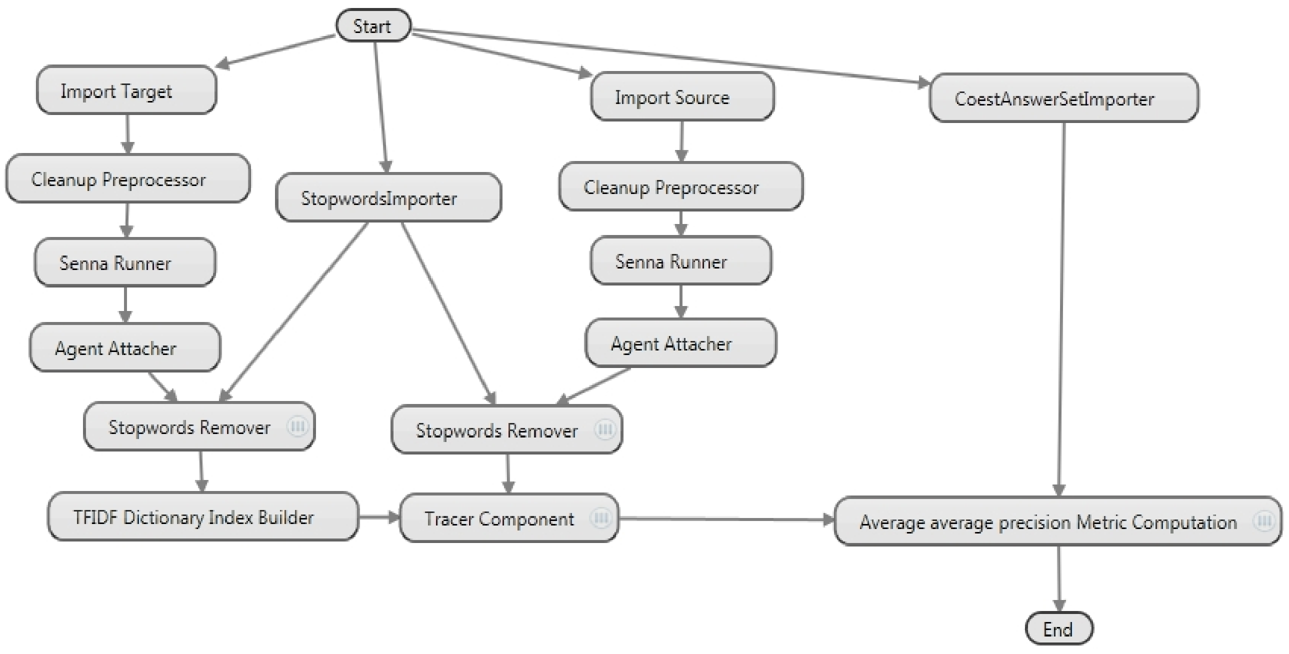

Figure 1. TraceLab Components for QuEST using Senna Role Labeler.

II. Enhancing Queries

In this section, we discuss our approach as well as our experiences with TraceLab and Senna.

A. QuEST Approach

Our approach aims to improve the performance of tracing by enhancing queries using semantic information. Current algorithms used for tracing decide if there is a link between two artifacts based on their shared keywords. However, the semantic meaning of the same keyword may be different in different artifacts, and this difference is ignored in current tracing methods. For instance, if one artifact describes the behavior of an entity while the other artifact merely mentions it in the text, then it is less likely that there is a link between these two artifacts; current non-semantic tracing methods will use this co-occurrence of keywords as evidence of a link simply because the term appears in both artifacts. The key idea of our approach is that we can convert one keyword into multiple new keywords based on its semantics in different artifacts. We do this by appending tags to the keywords. The resulting new keywords will not be used as a proof of the links among artifacts if their tags are different.

For now the semantic information we use is the role(s) of a word. A semantic role is the underlying relationship that a word has with the action, the behavior of entities that change the state of the system. Such roles include agent, object, goal, etc. For now we focus on the role of *agent*, the entities that perform actions. We do so because agent is the most common role, every action should be performed by an agent.

We use Semantic Role Labeling (SRL) to find the semantic information we need. SRL is a natural language processing task that detects the semantic arguments of verbs or predicates and the roles of these arguments. For example, given "*a system updates data*," SRL finds the verb *update* with *system* as its agent. In this case, our method appends the tag (AG) to *system* to represent semantic information. As a result, the *system(AG)* and other occurrences of simply *system* become different keywords, making it less likely that the artifacts with *system(AG)* and *system* will be linked.

We propose the approach to improve the precision of tracing. The major challenge of the approach is how to append tags to the correct words. Identifying the "agent" words is not an easy task due to the ambiguity and complexity of natural language.

We use the semantic role labeler Senna [15] in our approach. Senna provides Parts of Speech (POS) tags and chunks as well as SRL information. We use the Senna semantic roles and chunks to identify the words that should be appended with tags. For example, given the following requirement (from CM-1):

"The dpu ccm shall implement mechanism whereby large memory loads and dumps can be accomplished incrementally."

SRL finds two actions: *implement* and *accomplish.* The agent of *implement* is *dpu ccm,* which is a chunk being marked as a noun phrase, while the agent of *accomplish* is unknown. In this case, it is clear that *dpu ccm* performs the action. However, the agents found by Senna may contain more than one chunk. For instance, given a requirement:

“The node that has received the message should save the message.”

the agent is “*the node that has received the message.*” It is obvious that appending the tag to all words introduces too much noise because this will generate unnecessary new keywords, such as *message*(AG) and *received*(AG), and the only word we want to identify is *node*.

To address this challenge, we designed a heuristic method after reviewing multiple datasets. Our approach assumes that the first noun phrase chunk in the identified agent is the real “*agent chunk*. “ However, appending the tag to each word in this noun phrase may still cause a problem. For instance, given a requirement that states: “*the activated node should do something*,” the noun phrase is *activated node* and the tag should not be appended to *activated*. To solve this problem, the heuristic method only appends tags to the last word in the noun phrase. Thus, the heuristic method changes the previous example to:

“The dpu ccm(AG) shall implement mechanism whereby large memory loads and dumps can be accomplished incrementally.”

The *(AG)* tag is only appended to *ccm* even though the agent is *dpu ccm*. It should be noted that it is possible that all words in a noun phrase chunk form the name of the entity, such as *dpu ccm*. We decided that these cases should be handled by other preprocessing steps, such as replacing *dpu ccm with dpuccm*. Thus, the name of an entity is one word, which will be tagged.

B. TraceLab Components and Datatypes

To implement our approach, we first created four new TraceLab types to model the output of Senna: NLPSentence, NLPPredicate, NLPChunk, and NLPWord. The constructor of NLPSentence reads a string output by Senna which includes the POS, chunks, and SRL information, and creates a list of predicates, chunks, and words objects. Each NLPPredicate object also contains a set of arguments and their roles.

We then created two components: SennaRunner and AgentAttacher. SennaRunner calls Senna and passes the source and target artifacts to the semantic role labeler. Senna generates results for all the input artifacts. SennaRunner then creates and outputs a set of NLPSentence objects based on these results, one for each input artifact. AgentAttacher reads the output of SennaRunner and identifies the agent words using the heuristic method described above. The component outputs a list of artifacts with all identified words appended with the *(AG) and (AC)* tag.

We also conducted a study in the TraceLab environment to validate our approach. We designed two experiments; one for standard vector space model tracing and the other for tracing with the tags attached. We compared their results. The dependent variable is Mean Average Precision (MAP). The null hypothesis is that there is no difference between the MAP of both experiments, and the alternative hypothesis is that there is a difference.

Figure 1 shows the TraceLab experiment for QuEST. All components except for SennaRunner and AgentAttacher are reused TraceLab components. At the beginning of the experiment, source and target artifacts are imported separately. Then the Cleanup Preprocessor removes all non-characters (punctuation marks, numbers, etc.) in these artifacts. SennaRunner uses these “*clean*” artifacts as input and AgentAttacher returns updated artifacts in which all *agents* are appended with the tag *(AG)*. (We originally appended (AG) and (AC) but decided to study the effect of just (AG) for this Challenge) The next step is stop word removal. Then, the tfidf dictionary index is built based on the updated target artifacts and the Tracer component generates the similarity matrix. The CoestAnswersetImporter imports the baseline matrix, and the Metric Computation component computes the MAP for these two matrices.

The standard experiment is similar; the only difference is that the artifacts are not updated by SennaRunner and AgentAttacher. The output of Cleanup Preprocessor is directly input to the Stopwords Remover.

It should be noted that the sequence of Cleanup Preprocessor, AgentAttacher, and Stopwords Remover cannot be changed. AgentAttacher generates words such as *system(AG)*. If Cleanup Preprocessor is connected after AgentAttacher, the words above will become *system ag,* which cannot be used as a single keyword. Additionally, if stop word removal is performed before AgentAttacher, then the text will be hard for Senna to parse.

C. Experience with TraceLab

We found that designing and implementing experiments in TraceLab can save a great amount of time and effort. As mentioned earlier, we found most of the components we needed in the TraceLab library. This meant our main task was to create the workflow and assign the inputs and outputs for each component. The standard interfaces of the components made it easy to connect the components together. This saved us much time as we did not need to “reinvent the wheel.”

The process of creating our new components was also straightforward. We wrote the code for running Senna and attaching the agent tag previously while researching this idea; converting this code into TraceLab components was an easy task. Also, we studied the effect of stopword removal and preprocessing cleanup on our approach with a snap of the fingers. In TraceLab, we only had to change the order among the corresponding components. It would be much more complex if we had to write all those programs on our own as well as and change the call order of the methods. Last but not least, the Workspace View is very convenient for testing.

There were a few issues that we encountered during this study, largely due to TraceLab being in alpha testing. First, we found that the tracer component for using the TF-IDF dictionary generated strange results. When we traced an artifact to itself, we expected the similarity score to be 1; however, the score returned by TraceLab was much lower. Second, while TraceLab provides a component to visualize the results, there is no component available for exporting the results; this functionality would be very useful for data analysis. Third, sometimes TraceLab freezes up after we refresh the component library; fortunately this was easily worked around: we just had to close the workspace and start a new one. These issues have been reported and will be fixed in future releases.

III. Results

We ran the experiment on the EasyClinic dataset, provided for the TEFSE Challenge in 2009. This dataset contains four types of artifacts: use cases (*uc*), description of interaction diagrams (*id*), test cases (*tc*), and code classes (*cc*). We traced from use cases to the other three types of artifacts (*uc-id, uc-tc, uc-cc*). The results are shown below.

TABLE I. MAP Results for EasyClinic dataset

	Standard	Agent attached
uc-cc	**0.654**	0.591
uc-tc	**0.685**	0.526
uc-id	0.518	**0.629**

The results show that MAP improved by 21% for *uc-id*. However, the MAP decreases by 9% in *uc-cc*, and by 23% in *uc-tc*.

It appears that appending the agent tag makes the performance for two of the artifact pairs (uc-cc and uc-tc) worse. However, it should be noted that Senna parses structured natural language text, and the performance of our approach is dependent on this and the correctness of Senna. We therefore investigated Senna's performance for our artifacts.

Of the four types of artifacts in EasyClinic, only the descriptions of interaction diagrams are completely written in natural language. The use cases contain general descriptions which are written in natural language but also contain the description of users' steps such as:

1 View the mask to enter information needed
2 Inserts data about the anagrafica of laboratory
3 Confirm placement

First, these steps are hard for Senna to parse. Second, these steps specify "(users) do ...," and the agents are users; while interaction diagrams describe the structure of the software system and use the entities in the system as agents. Test cases and code classes contain even less structured natural language; the majority of *tc* texts are the specification of input/output data, such as:

Input Visit selected:
06/10/2003 hours 09 00 Visit control .

Moreover, the *tc* texts contain many formatted descriptions that Senna cannot parse:

Classes cover valid: CE4 CE8 CE13
Classes are not valid: None
High Priority .

The code classes are similar to the test cases. The results above imply that there is some relation between the amount of structured natural language contained in the artifacts and the performance of our approach (the more natural language text, the better our approach works, and vice versa). The presence of non-translated Italian terms in EasyClinic may also have caused issues for Senna.

We also performed the experiment on the WARC, PINE, and CM1 datasets. We traced from WARC functional requirements to design elements, from Pine requirements to use cases, and from CM1 requirements to design elements. Based on our findings for EasyClinic, we expected QuEST to work well for WARC and CM1, but not for Pine.

TABLE II. Results for WARC, Pine, and cm1 dataset

	Standard	Agent attached
WARC	**0.56**	0.532
Pine	**0.539**	0.528
CM1	0.492	**0.495**

The results of the two methods are close, though the standard approach performed slightly better. This may be explained by the discussion above, for instance, the use cases of Pine contain texts such as:

User is looking at main menu
User presses Folder List
User presses right one time, selecting sent-mail.

In such cases, the only agent is *User*, while the requirements of Pine are concerned with "*what the system should do.*" We also found that some natural language artifacts such as the requirements in WARC contain the header *FR* before the text. Such headers will not cause problems for the standard approach, because high level or low level artifacts share the same header. But these headers can cause mistakes for our approach: our heuristic method of identifying agents is looking for the first noun phrase in agents, and these headers are always parsed as the first noun phrase in an agent. In the case above, the AgentAttacher will generate *fr(AG)* for a large number of WARC requirements, which is meaningless. We eliminated these headers by using a stopword removal component in TraceLab. Interestingly, while the agents were correctly tagged, the performance of our approach did not improve. We found that although several true links were found because of the agents being correctly tagged, some other true links were missed. The reason is that at least one artifact of the missed links did not contain the agent keywords, and the header (tagged or not) caused coincidental matches.

As mentioned above, we also tried to append *(AC)* tags to the action verbs found by Senna. Interestingly, all the results using both *(AC)* and *(AG)* tags are worse than the results when only using the *(AG)* tags. It is possible that verbs do not assist with tracing.

It seems the experiment results do not prove that our QuEST approach is generally better than the non-semantic approach. However, the results elucidate many interesting lessons/observations. First, our approach should not be used to trace artifacts such as code or test cases which are not comprised of largely natural language text. Second, the semantic tagging of an agent is not useful if the artifacts are based on different points of view. For example, use cases are concerned with what the users want to do, while requirements specify what the system should do. Third, the format of the text, such as the header, may lead to errors for the natural language parsers and thus decrease the performance of the QuEST approach.

Nonetheless, our approach still proves to be very efficient in the tracing of EasyClinic *uc-id*. This shows that semantic information can be very helpful when both artifacts are written in natural language. It should be noted that less than half of the texts in the use cases are natural language, but the improvement was still impressive (21%).

IV. Conclusion and Future Work

It should be noted that the semantic work represented in this paper is merely a first step. As mentioned earlier, Senna provides much more semantic information than just the actions and agents. Given an action, Senna can also extract its goal, its object, its manner, etc. Also, there are many other natural language processing tools, such as Stanford Parser [16,17], that provide more functionality for analyzing syntax and semantics.

Although the semantic meaning of agent was not useful in the tracing between use cases and requirements in our study, other techniques may find information that indeed connects these two artifact types. Also, there may be better ways of enhancing queries and document collections than simply appending tags to words, considering how much information we can possibly add to the artifacts.

Additionally, the relation between our approach and existing preprocessing tasks is also worth studying. Some possible questions to study include: If we combine a stemming technique and our approach, will it make our approach perform better? How does the use of a splitter impact our approach? Will a thesaurus affect the results of our approach?

Acknowledgment

This work is funded in part by the National Science Foundation under NSF grants CCF-0811140 (research) and ARRA-MRI-R2 500733SG067 (benchmark development). This work was previously sponsored by NASA under grant NNG05GQ58G. We thank Dr. Maureen Doyle for helpful comments on an earlier version of the paper.

References

[1] S K Sundaram, J H Hayes, A Dekhtyar, and E A Holbrook. 2010. “Assessing traceability of software engineering artifacts.” Journal of Requir. Eng. 15, 3 (September 2010), 313-335.

[2] J H Hayes, A Dekhtyar, and S Sundaram, “Improving After-the-Fact Tracing and Mapping to Support Software Quality Predictions,” IEEE Software, Volume 22, Number 6, November/December 2005, pp. 30 – 37.

[3] P Maeder, M Riebisch, and I Philippow: “Traceability for Managing Evolutionary Change,” 15th International Conference on Software Engineering and Data Engineering (SEDE-2006): 1-8.

[4] J H Hayes, A Dekhtyar, and S Sundaram*, “Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods,” IEEE Transactions on Software Engineering, Volume 32, No. 1, pp. 4-19, January 2006.

[5] J H Hayes, A Dekhtyar, and J Osbourne, “Improving Requirements Tracing via Information Retrieval,” in Proceedings of the International Conference on Requirements Engineering (RE), Monterey, California, September 2003, pp. 138 – 148.

[6] J H Hayes, A Dekhtyar, S Sundaram, and S Howard, “Helping Analysts Trace Requirements: An Objective Look,” in Proceedings of IEEE Requirements Engineering Conference (RE) 2004, Kyoto, Japan, September 2004, pp. 249-261.

[7] J Lin, C C Lin, J C Cleland-Huang, R Settimi, J Amaya, G Bedford, B Berenbach, O B Khadra, C Duan, and X Zou, “Poirot: a distributed tool supporting enterprise-wide traceability”, Proceeding of the 14th IEEE International Conference on Requirements Engineering, Minneapolis, MN, 2006, pp. 356-357.

[8] X Zou, “Improving Automated Requirements Trace Retrieval Through Term-Based Enhancement Strategies,” DePaul University, Technical report 4-1-2009, 2009.

[9] H U Asuncion, A U Asuncion, and R N Taylor. 2010. “Software traceability with topic modeling.” In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1 (ICSE '10), Vol. 1. ACM, New York, NY, USA, 95-104.

[10] A Marcus and J I Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," Software Engineering, 2003. Proceedings. 25th International Conference on , vol., no., pp. 125- 135, 3-10 May 2003.

[11] A D Lucia, R Oliveto, and G Tortora. 2008. Adams re-trace: traceability link recovery via latent semantic indexing. In Proceedings of the 30th international conference on Software engineering (ICSE '08). ACM, New York, NY, USA, 839-842.

[12] H Jiang; T N Nguyen.; I X Chen; H Jaygarl; and C K Chang, "Incremental Latent Semantic Indexing for Automatic Traceability Link Evolution Management," Automated Software Engineering, 2008. ASE 2008. 23rd IEEE/ACM International Conference on , vol., no., pp.59-68, 15-19 Sept. 2008.

[13] G Spanoudakis, A Zisman, E Pérez-Miñana, and P Krause: Rule-based generation of requirements traceability relations. Journal of Systems and Software (JSS) 72(2):105-127 (2004)

[14] L Màrquez, X Carreras, K C Litkowski, S Stevenson, “Semantic Role Labeling: An Introduction to the Special Issue,” Computational Linguistics, June 2008, Vol. 34, No. 2, Pages 145-159.

[15] Senna, http://ml.nec-labs.com/senna/, last accessed January 25, 2013.

[16] Marie-Catherine de Marneffe, Bill MacCartney and Christopher D. Manning. 2006. Generating Typed Dependency Parses from Phrase Structure Parses. In LREC 2006.

[17] Dan Klein and Christopher D. Manning. 2003. Accurate Unlexicalized Parsing. Proceedings of the 41st Meeting of the Association for Computational Linguistics, pp. 423-430.

[18] Russel, S.J. Norvig, P. 1995. Artificial Intelligence: A Modern Approach, Prentice-Hall, Englewood Cliffs, New Jersey, USA.