OpenTrace: an Open Source Workbench for Automatic Software Traceability Link Recovery

Elian Angius and René Witte*
Semantic Software Lab
Department of Computer Science and Software Engineering
Concordia University, Montréal, QC, Canada
Email: e_angiu@encs.concordia.ca, witte@semanticsoftware.info

***Abstract*—Research in automatic traceability link recovery typically involves running a large number experiments on various datasets in order to compare the performance of different approaches in different configurations. Each experiment comprises a number of steps, including data preparation, artifact preprocessing, link generation, and filtering. Additionally, evaluation metrics need to be computed on the obtained results. We present OpenTrace, the first extensible open source workbench that facilitates reproducible experiments in traceability research through a pluggable component framework.**

I. Motivation

The automatic detection of traceability links between software artifacts facilitates impact analysis, program maintenance and reverse engineering activities. The literature [1]–[4] mentions several tools that were built to address specific parts of the traceability problem, but most of these are currently either unavailable, closed source, or lack features specifically targeting the *evaluation* of experiments. Moreover, while a high-level summary of the experimental results is typically published, the corresponding source code and its detailed configurations are not, which makes it difficult or impossible for a reader to reproduce the obtained results [5].

The primary goal of our *OpenTrace*[1] workbench is to support the scientific discovery workflow in traceability research throughout data preparation, experiment execution, and evaluation.[2] It is built based on a component-based framework that allows others to easily extend it, e.g., by implementing new filtering, preprocessing, or link generation techniques. A number of standard techniques are included with the distribution, in particular the Information Retrieval (IR) techniques Vector Space Model (VSM) and Latent Semantic Analysis (LSA).

We place a strong emphasis on *reproducibility* [7]: one of our goals is to complement a scientific publication with a corresponding online configuration that will allow any reader to easily download, run, and analyse the results, thereby obtaining the same values as in published experiments.

II. Experimental Process

The overall workflow of our approach, depicted in Fig. 1, consists of the following parts:

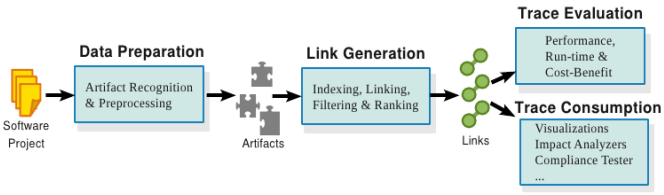

Fig. 1: Workflow in traceability experiments

Data Preparation mines software repositories for project data with the goal of recognizing artifacts, classifying them into types and preprocessing content used in linking.
Link Generation indexes artifacts for traceability discovery that are then compared with one another before candidate links are ranked and filtered.
Trace Evaluation measures the correctness and quality of links against a reference answer-set.
Trace Consumption of links in other software engineering tools to support activities like compliance testing, change impact analysis, design and architecture visualization and reverse engineering, among others.

III. OpenTrace Design & Implementation

Our traceability workbench is implemented based on the General Architecture for Text Engineering (GATE) [8], [3] an open source component-based framework that allows to dynamically configure complex analysis *pipelines*, where each component performs a specific analysis task, e.g., tokenization or stopword filtering. Using the GATE Developer GUI, our workbench can be remotely downloaded and installed through the GATE Plugin Manager. In this process, external library dependencies are automatically resolved through Apache Ivy.[4]

A. Analysis Pipelines

Pipelines are executed on a corpus of (here) software artifacts, such as requirements documents or source code files. An example for an OpenTrace pipeline is shown in Fig. 2. The first components perform standard pre-processing tasks, including tokenization, grouping words into sentences, and assigning parts-of-speech tags to words according to their grammatical usage. The pipeline then tags stop-words and performs lemmatization.

[1]OpenTrace, http://www.semanticsoftware.info/opentrace
[2]Similar goals are described within the TraceLab [6] project, but it was not publicly available at the time of writing.
[3]GATE, http://gate.ac.uk/
[4]OpenTrace can also be used as an embedded library for production use, including web service invocation.

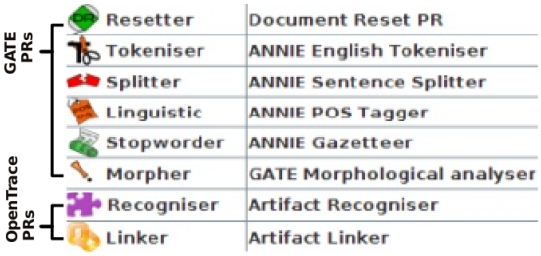

Fig. 2: Traceability Pipeline Components

After document purification, the *Artifact Recogniser* invokes custom JAPE rules [8] to create and preprocess the required input annotations that are fed into our *Artifact Linker* component. As the pipeline iterates through the corpus, our engine indexes artifacts of the specified type, according to the selected trace technique and vocabulary base. When all artifacts in the corpus have been processed, traceability links are created by querying the index against each artifact within scope. Both intermediate and end results of the analysis are stored in form of *annotations*. One of the input annotations is `Artifact`, which marks a single artifact used for linking, and can be configured according to both *type* (e.g., use case, test case) and *granularity* (e.g., document level, paragraph level).

The output `Link` annotations include a "src" and "dst" reference to the names of the related source and destination artifacts, along with a "sim" feature for the calculated degree of similarity between artifacts, as well as a "type" for the semantics of the links.

B. Traceability Link Generation

The Artifact Linker component provides a pluggable framework for executing concrete link generation strategies. The linker can be configured with both strategy-dependent and -independent parameters:

Link Scope: Defines the type of artifacts that should be linked (e.g., source code, use case, test case).

Link Directionality: Our links are uni-directional mappings between pairs of source to destination artifacts. Therefore, forwards, backwards and horizontal traceability [9] are supported.

Traceability Techniques: For selecting one of the implemented link modules (currently IR based, LSA or VSM strategies).

Vocabulary Base: Indexing terms that make up artifact content can be specified using only destination (partial vocabulary) or both source and destination artifacts (full vocabulary) [1].

Link Filtering: A trace can theoretically generate many links, where not all of them carry the same importance [10]. We provide for both absolute and relative filters as described in [11] that remove candidates based on link similarity and quantity.

The resulting links can be exported both as `Link` annotations (in XML stand-off markup) and as a traceability matrix file, as shown in Fig. 3.

C. Evaluating Results

Our separate *Trace Evaluator* component is responsible for benchmarking generated results against a given answer set. This component computes *precision*, *recall* and *selectivity* [1] metrics, among others. Analysts can build their own or use existing answer sets conforming to the structure in Fig. 3. These contain three parts: (a) a header indicating the link type and trace scope, (b) a similarity weighted mapping between linked artifact pairs,[5] and (c) an explicit list of unlinked artifacts. Breaking up answer-sets into this structure facilitates experimenting with various trace mapping combinations.

```
similarity
REQ                 :  UC
F5.txt(0,83)        :  UC4.txt(0,267)@0.90697575
F5.txt(0,83)        :  UC7.txt(0,260)@0.31838852
NF2.txt(0,139)      :  UC7.txt(0,260)@0.29290316
F21.txt(0,47)       :
                    :  UC32.txt(0,44)
```

Fig. 3: Example for a generated Traceability Matrix

IV. Conclusion and Future Work

OpenTrace is a flexible implementation of a generic traceability workbench, unifying various aspects of automated link recovery into a single tool. It is the first open source workbench for traceability experiments that supports evaluation and can be used to package experiments in a form that allows others to easily reproduce the obtained results. Future releases will add additional visualization capabilities of the traceability and evaluation results.

References

[1] A. Dekhtyar, E. Holbrook, J. H. Hayes, and S. K. Sundaram, "Assessing traceability of software engineering artifacts," *Requirements Engineering*, vol. 15, pp. 313–335, 2010, 10.1007/s00766-009-0096-6.

[2] S. Ratanotayanon, S. E. Sim, and R. Gallardo-Valencia, "Supporting Program Comprehension in Agile with Links to User Stories," in *Agile Conference (AGILE '09)*, August 2009, pp. 26–32.

[3] S. Klock, M. Gethers, B. Dit, and D. Poshyvanyk, "Traceclipse: an eclipse plug-in for traceability link recovery and management," in *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*, ser. TEFSE '11. New York, NY, USA: ACM, 2011, pp. 24–30.

[4] J. David, M. Koegel, H. Naughton, and J. Helming, "Traceability rearmed," in *33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC '09)*, vol. 1, july 2009, pp. 340–348.

[5] L. Hatton, J. Graham-Cumming, and D. Ince, "The case for open computer programs," *Nature*, vol. 482, pp. 485–488, Feb. 2012.

[6] A. Czauderna, M. Gibiec, G. Leach, Y. Li, Y. Shin, E. Keenan, and J. Cleland-Huang, "Traceability challenge 2011: using TraceLab to evaluate the impact of local versus global IDF on trace retrieval," in *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*, ser. TEFSE '11. New York, NY, USA: ACM, 2011, pp. 75–78.

[7] T. Pedersen, "Empiricism is not a matter of faith," *Comput. Linguist.*, vol. 34, no. 3, pp. 465–470, Sep. 2008. [Online]. Available: http://dx.doi.org/10.1162/coli.2008.34.3.465

[8] H. Cunningham et al., *Text Processing with GATE (Version 6)*. University of Sheffield, Dept. of Computer Science, 2011. [Online]. Available: http://tinyurl.com/gatebook

[9] A. van Lamsweerde, *Requirements Engineering*. John Wiley and Sons, 2009.

[10] A. Egyed, S. Biffl, M. Heindl, and P. Grünbacher, "Determining the cost-quality trade-off for automated software traceability," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, ser. ASE '05. ACM, 2005, pp. 360–363.

[11] 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, 09 2007.

[5] Artifact naming is based on the document filename, e.g., "UC4.txt", and character start and end offsets, e.g., "(0,267)", within documents.