TraceLab: An Experimental Workbench for Equipping Researchers to Innovate, Synthesize, and Comparatively Evaluate Traceability Solutions

Ed Keenan[1], Adam Czauderna[1], Greg Leach[1], Jane Cleland-Huang[1], Yonghee Shin[1]
Evan Moritz[2], Malcom Gethers[2], Denys Poshyvanyk[2], Jonathan Maletic[3], Jane Huffman Hayes[4], Alex Dekhtyar[5]
Daria Manukian[1], Shervin Hussein[1], Derek Hearn[1]

DePaul Univ.[1], *Chicago, IL 60604*, *keenan@cs.depaul.edu*, *jhuang@cs.depaul.edu*

College of William and Mary[2], *Williamsburg, VA 23185*, *denys@cs.wm.edu*

Kent State Univ.[3], *Kent, OH*, *jmaletic@kent.edu*

Univ. of Kentucky[4], *Lexington, KY 40506*, *hayes@cs.uky.edu*

CalPoly[5], *San Luis Obispo, CA*, *dekhtyar@calpoly.edu*

***Abstract*—TraceLab is designed to empower future traceability research, through facilitating innovation and creativity, increasing collaboration between researchers, decreasing the startup costs and effort of new traceability research projects, and fostering technology transfer. To this end, it provides an experimental environment in which researchers can design and execute experiments in TraceLab's visual modeling environment using a library of reusable and user-defined components. TraceLab fosters research competitions by allowing researchers or industrial sponsors to launch research contests intended to focus attention on compelling traceability challenges. Contests are centered around specific traceability tasks, performed on publicly available datasets, and are evaluated using standard metrics incorporated into reusable TraceLab components. TraceLab has been released in beta-test mode to researchers at seven universities, and will be publicly released via CoEST.org in the summer of 2012. Furthermore, by late 2012 TraceLab's source code will be released as open source software, licensed under GPL. TraceLab currently runs on Windows but is designed to port to Linux and Mac environments.**

***Keywords*-Traceability, Instrumentation, TraceLab, Benchmarks, Experiments.**

I. Introduction

Requirements traceability, defined as "the ability to follow the life of a requirement, in both a backward and forward direction" [7] provides essential support for the development of large-scale, complex, and/or safety-critical software systems. In practice, organizations struggle to implement successful and cost-effective traceability, primarily because tracing is time-consuming, costly, arduous, and error prone [7], [9]. These difficulties have created a compelling research agenda that has been funded by agencies such as NSF and NASA, and by individual corporations such as Siemens and Microsoft.

Although extensive research efforts in the past decade have led to new discoveries and traceability solutions that have improved the reliability, safety, and security of IT systems, these advances are hampered because the stovepipe solutions of various research groups make it difficult to comparatively evaluate and cross-validate solutions, or synthesize different algorithms in new and exciting ways. Furthermore, new researchers must invest significant time recreating basic traceability functions and frameworks before they can even start to investigate new solutions.

To address these problems, we have developed an environment designed to facilitate innovation and creativity, increase collaboration between traceability researchers, decrease the startup costs and effort of new traceability research projects, and foster technology transfer [2]. This research environment lays a foundation for future advances in the field of traceability, and has the potential to accelerate and shape future research and to remove currently inhibitive research roadblocks. The TraceLab project is funded by the National Science Foundation and conducted by members of the Center of Excellence for Software Traceability (CoEST) [1].

II. TraceLab Overview

TraceLab provides a fully functioning experimental environment in which researchers can compose experiments from a combination of existing and user-defined components, utilize publicly available datasets, exchange components with collaborators, and comparatively evaluate results against previous benchmarks. TraceLab is constructed in .NET using the Windows Presentation Foundation (WPF).

TraceLab experiments are composed from a set of executable components and decision nodes, all of which are laid out in the form of a precedence graph on a canvas. This is illustrated in Figure 1, which depicts a simple experiment that was conducted in response to the TEFSE 2011 challenge [3]. This experiment evaluated two different techniques for building a term dictionary as part of a requirements trace retrieval task [5]. Components included importers, preprocessors, trace algorithms for generating similarity scores between source and target artifacts, and a results components

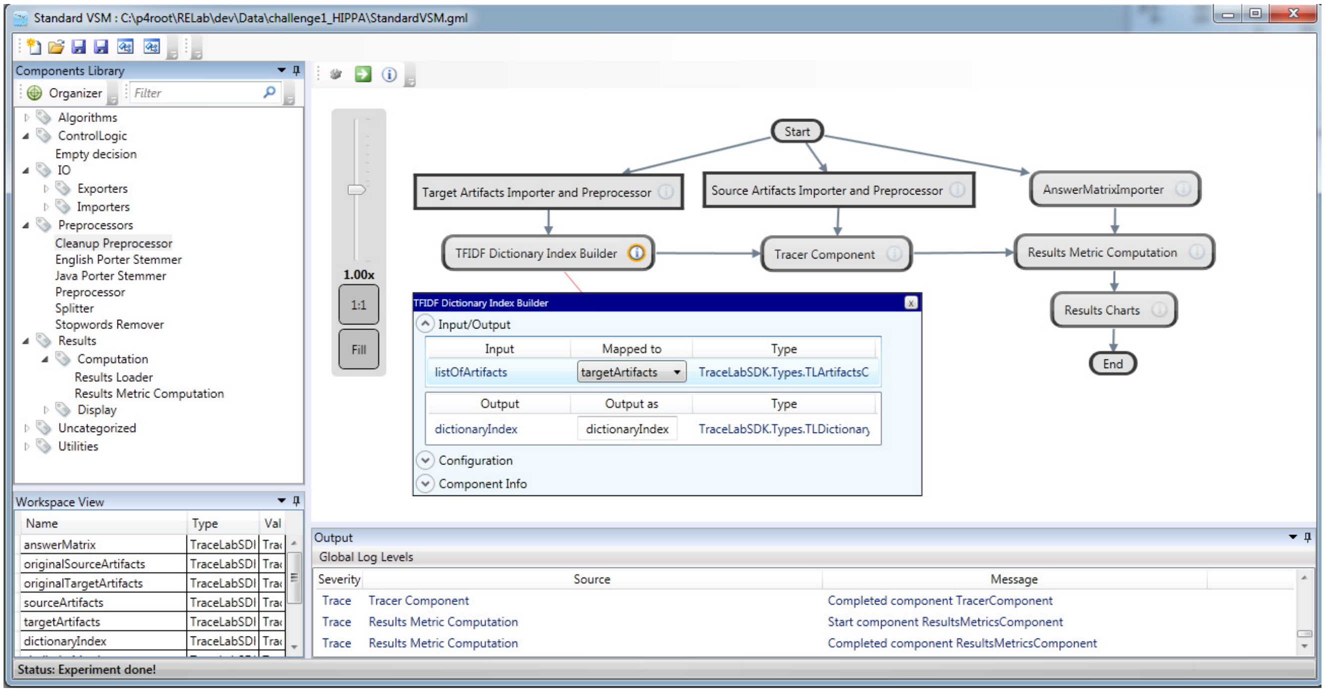

Figure 1. The TraceLab Integrated Research Environment

for collating and reporting results. Individual components used in this experiment were written in C# and Java.

Components can be primitive or composite, depicted in the graph with sharp or rounded corners respectively. For example, the *Targets Artifact Importer Preprocessor* is a composite component which contains a lower level graph of components responsible for the mundane tasks of importing XML files, removing stop words (common words), and stemming words to their morphological roots. This hierarchical arrangement of components allows TraceLab to handle complex experiments. TraceLab's component library is shown in the upper left hand side of the screen, while the data workspace depicting standard data structures used in this experiment is shown in the lower left hand side. This workspace is used at runtime to exchange data between components. Other features, not shown here include debugging utilities, decision nodes, and export functions. At runtime, execution starts with the *start* node and ends with the *end* node. Intermediate nodes can be executed in parallel as long as all of their prerequisite nodes have completed execution.

```
using TraceLabSDK;
using TraceLabSDK.Types;

namespace DictionaryIndexBuilder
{
    [Component(GuidIDString = "1C30B7B5-3E04-433D-817F-B0BE187B154F",
                Name = "TFIDF Dictionary Index Builder",
                DefaultLabel = "TFIDF Dictionary Index Builder",
                Description = "Creates a tf-idf dictionary. ",
                Author = "DePaul RE Team",
                Version = "1.0")]
    [IOSpec(IOSpecType.Input, "listOfArtifacts",
            typeof(TraceLabSDK.Types.TLArtifactsCollection))]
    [IOSpec(IOSpecType.Output, "dictionaryIndex",
            typeof(TraceLabSDK.Types.TLDictionaryIndex))]
    public class TFIDFDictionaryIndexBuilderComponent : BaseComponent
    {
        public TFIDFDictionaryIndexBuilderComponent(ComponentLogger log)
            : base(log) { }

        public override void Compute()
        {
            Logger.Trace("Start component TFIDF Dictionary Index Builder");

            TLArtifactsCollection listOfArtifacts =
                (TLArtifactsCollection)Workspace.Load("listOfArtifacts");

            TLDictionaryIndex dict =
                TFIDFIndexBuilder.build(listOfArtifacts);

            Workspace.Store("dictionaryIndex", dict);

            Logger.Trace("Completed component TFIDF Dictionary Index Builder");
        }
    }
}
```

Figure 2. A C# program modified for integration with TraceLab

III. Features

In this section we highlight some of the most important features in TraceLab.

A. Components

A TraceLab component can be written using almost any memory-managed language such as C#, Java, or Visual Basic. As depicted in Figure 2, an ordinary class, or set of classes, can be integrated into TraceLab, by adding metadata information to the code, and then by importing the compiled code into the TraceLab component library. Metadata includes a name, a set of input parameters, a set of output parameters, and a description of the component. Input and output parameters must be standard TraceLab

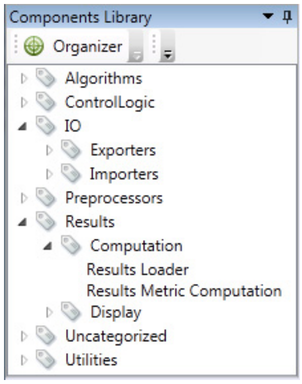

Figure 3. TraceLab's Component Library. Users tag items to support extensive search features. New components are easily imported by copying them into a component directory.

datatypes, which means that the programmer must either develop the component to use compatible datatypes, or else create an adapter. TraceLab datatypes support a wide array of primitives such as arrays, lists, integers, and strings, as well as community-defined data structures such as trace matrices, artifact lists, and dictionaries of terms. Researchers can also define their own datatypes.

To facilitate the reuse of components, TraceLab's component library, depicted in Figure 3, provides a flexible hierarchy based around user defined categories. It also allows users to tag components and to perform tag-based searches.

B. Working with Components

In a TraceLab experiment, data is exchanged between components via the workspace. Each individual component must be configured prior to use according to the input, output, and configuration parameters defined by the programmer of the component. For example, the *TFIDF Dictionary Index Builder* component shown in Figure 1 reads in a *listOfArtifacts* of type *TraceLab-SDK.Types.TLArtifactsComponent*, which is mapped by the researcher to a variable named targetArtifacts. Similarly, the dictionaryIndex is mapped to an output variable named *dictionaryIndex*. In this way, the component exchanges data from TraceLab's workspace during runtime. Although not shown here, the component developer can also define configuration parameters which the researcher must also specify prior to using the component.

To import a user-defined component into TraceLab, the developer adds meta-data (as described above) to the main class of the component, maps any imported or exported TraceLab datatypes to the internal data structures, compiles the project into a .NET assembly, and copies the assembly to a TraceLab component directory. In the case of java files, the developer first compiles their java project into a jar file. That jar file must then be recompiled using IKVM to a .NET assembly, which is usable as a component in TraceLab. Our experience has shown that in most cases components can be

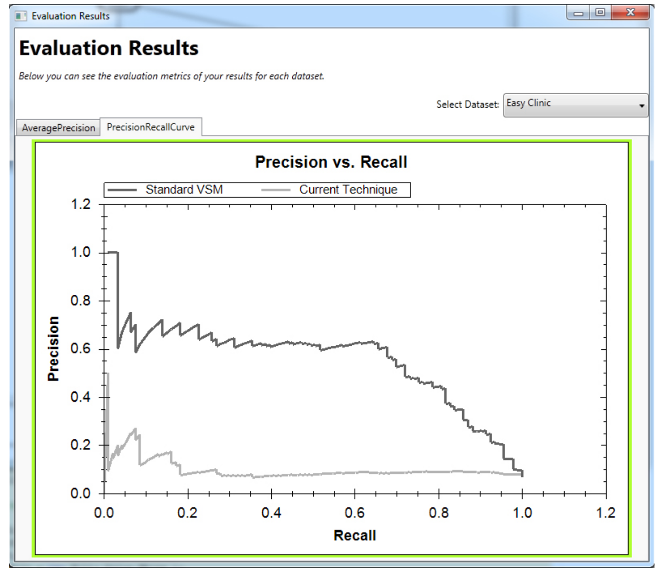

Figure 4. A TraceLab component used to evaluate contest results

integrated into TraceLab with little effort.

C. Running an experiment

At runtime, TraceLab visually depicts the progress of an experiment by highlighting the components that are currently being executed. Any logging information defined in the component is output to the screen, and the current state of the workspace is also dynamically updated.

IV. Benchmarking

One of the primary goals of TraceLab is to support the comparative analysis of competing techniques through the concept of *research contests*. A contest defines a specific traceability task such as "retrieve traces from requirements to code," provides the datasets on which the task is to be performed, and specifies the metrics by which the results are to be evaluated. Components, such as the one shown in Figure 4, are provided as part of the experimental environment. In this case, the component visualizes precision vs. recall of the contestant's solution versus the current benchmark, and shows that the benchmark significantly outperformed the proposed solution. A more complete explanation of TraceLab's support for contests is provided in our paper on Software Engineering Contests [4].

V. Usage Example

TraceLab has already been used to conduct several different experiments [5], [10]. In this section we describe one particular experiment [6], [8] designed to empirically evaluate an integrated approach for combining orthogonal techniques. The experiment, which is depicted in Figure 5, compared and combined the Vector Space Model (VSM), probabilistic Jensen and Shannon (JS) model, and Relational Topic Modeling (RTM) techniques. Researchers were able

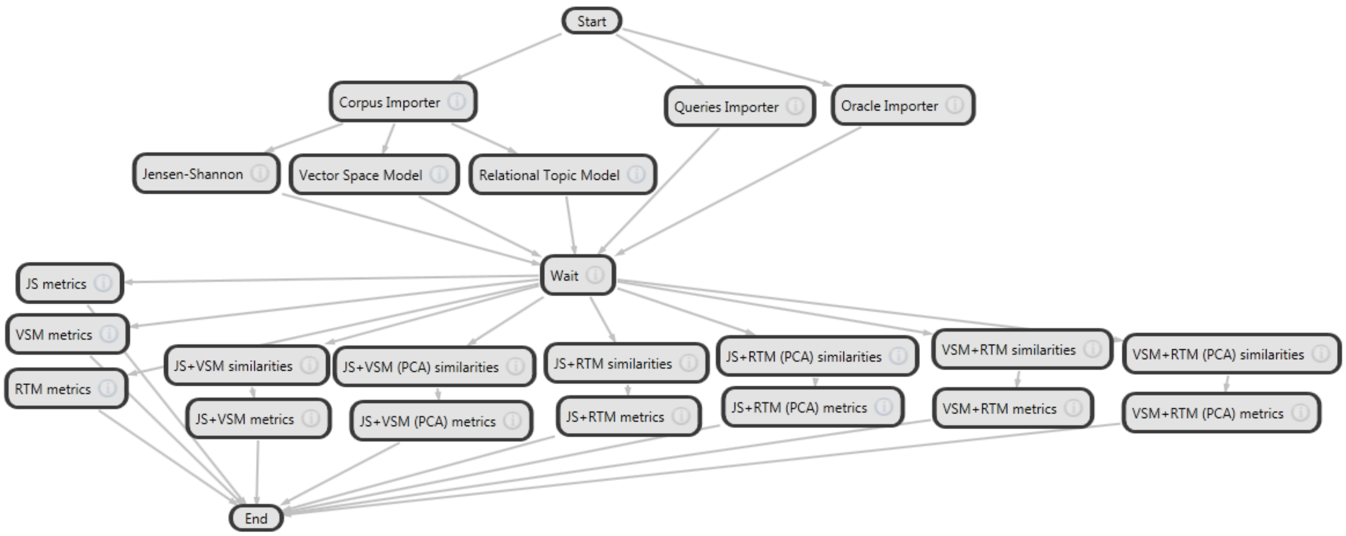

Figure 5. An application of TraceLab to empirically explore ways to integrate orthogonal tracing techniques

to construct the experiment using several built-in components as well as custom-built components for implementing additional functions such as Jensen-Shannon divergence, affine transformation, etc. In addition, they also developed new data types needed by the custom components. The experiment was executed on six datasets, namely eAnci, EasyClinic (English and Italian versions), eTour (English and Italian versions), and SMOS. Results showed that combining RTM with IR methods significantly outperformed stand-alone IR methods as well as any other combination of non-orthogonal methods.

VI. The Future of TraceLab

The current version of TraceLab increases runtime by a factor of approximately 2X. Although we are working to improve performance, this is certainly a tradeoff to be considered. However, the reduction in human effort needed to set up a traceability experiment could be measured in terms of months or even years.

To date, TraceLab has been used to perform both computationally expensive and user intensive experiments. Current efforts are focused on improving usability and performance, enhancing benchmarking and contest features, and on populating the library with a rich set of reusable components. TraceLab will be released to the public by July 2012. An online demo of our tool can be found at http://tinyurl.com/TraceLabDemo1.

VII. Acknowledgments

The work described in this paper was funded by the U.S. National Science Foundation under grant # CNS 0959924.

References

[1] CoEST: Center of excellence for software traceability, http://www.CoEST.org.

[2] Grand Challenges, Benchmarks, and TraceLab: Developing Infrastructure for the Software Traceability Research Community. *International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, 6, 2011.

[3] TEFSE (Traceability in Emerging Forms of Software Engineering) 2011 Traceability Challenge, May 2011.

[4] J. Cleland-Huang, Y. Shin, E. Keenan, A. Czauderna, G. Leach, E. Moritz, M. Gethers, D. Poshyvanyk, J. Huffman Hayes, and W. Lu. Toward actionable, broadly accessible contests in software engineering. In *New Ideas and Emerging Results (NIER Track), 34th International Conference on Software Engineering (ICSE)*, 2110.

[5] 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. *International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, 6, 2011.

[6] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. DeLucia. On integrating orthogonal information retrieval methods to improve traceability link recovery. In *Int'nl Conf. on Software Maintenance (ICSM'11)*, pages 133–142, 2011.

[7] O. Gotel and A. Finkelstein. Contribution structures (requirements artifacts). In *International Conference on Requirements Engineering*, pages 100–107, 1995.

[8] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. DeLucia. On the equivalence of information retrieval methods for automated traceability link recovery. In *International Conference on Program Comprehension (ICPC'10)*, pages 68–71, 2011.

[9] B. Ramesh and M. Jarke. Toward reference models of requirements traceability. *IEEE Trans. Software Eng.*, 27(1):58–93, 2001.

[10] Y. Shin and J. Cleland-Huang. A comparative evaluation of two user feedback techniques for requirements trace retrieval. In *27th Symposium on Applied Computing (SAC)*, 2012.