Querying Big Source Code

Carlos Garcia-Alvarado
Autonomic LLC
Palo Alto, CA 94304, USA
carlos@autonomic.ai

Carlos Ordonez
Dept. of Computer Science
University of Houston
Houston, TX 77204, USA
carlos@central.uh.edu

***Abstract*—Software compliance, auditing, and maintainability of large application repositories force organizations to rely on source code analysis tools to identify code vulnerabilities, data flows, technical debt, and bugs. We propose a novel method to identify data flows within an application by analyzing the code traces or 'links' that exist between the code and the data. Our application, SourceDB, leverages a relational database system as the backend to perform such discovery and computations. Our experiments show that SourceDB is able to process, analyze, and query the data source, logs, and source code in seconds.**

***Index Terms*—keyword search, source code analysis, SQL**

I. Introduction

ISO compliance, certifications, auditing requirements, security practices, and software maintainability force organizations of all sizes to monitor their source code repositories. This is especially challenging when the number and size of the code bases is very large and the number of active projects keeps increasing. As such, many companies have decided to rely on the automation of source code analysis tools to expose security vulnerabilities and technical debt, and reduce the risks of an unintended faulty routine [9]. While there are many solutions that perform static and dynamic source code analysis, there are very few that focus on exploring the data flow of an application.

The reason why most code analysis tools do not extract the data flow is due to the fact that the abstraction layers hide the "links" between the storage layer and the business logic portions of the application, the difficulty of identifying the underlying data source schema (schema and schema-less sources), and the nature of the typing languages (static and dynamic typing). In SourceDB, we focus on discovering hidden links that exist between a set of source code files and the physical model of a set of databases or log schemas.

Our approach exposes more additional information than that of traditional source code analysis tools since it is possible to follow the data flow via: a) an augmented source code documentation (e.g. Javadoc or Doxygen); b) traversing the recursive structure of the code; c) identifying critical/popular tables and source code methods and classes; d) tracing which source code functions use certain tables or columns; and e) analyzing online application logs to trace data accesses in the source code.

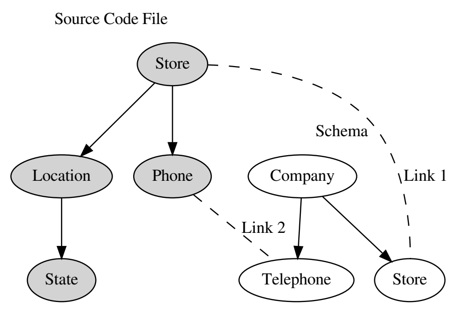


Fig. 1. Discovered Links: Link 1 (Store:class, Store:table, store:keyword) and Link 2 (Phone:field, Telephone:column, phone:keyword).

The closest related work is a static code analysis tool based on ontologies and semantic analysis [1]. SourceDB does not use priors (e.g. ontologies) and builds from our previous work in [5] and [6] by introducing data flow tracing visibility via Doxygen documentation, and especially log analysis.

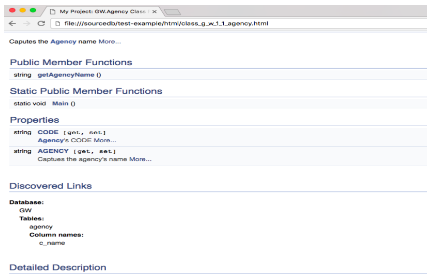
II. SourceDB

SourceDB is a system that allows tracing data flows through the discovery of links. It also provides the capabilities for performing complex analysis of the resulting links via OLAP-like aggregations and ranked searches. This approach was originally implemented to link semistructured data with structured data in [7]. Later on, we extended this approach to perform efficient approximate keyword matching inside the DBMS. Our linking strategy and algorithms are fully explained in [3], [4], [8]. For completeness, we provide a brief description of our architecture and linking strategy.

We initially focus on keyword searches between a source (e.g. DBMS schema or log file) S and a source code repository C. Therefore a data trace or link l_i is defined as:

$$l_i : \{S_p, C_q, k\}$$

where p and q are approximate keyword matches to k (see Figure 1). To power these features, our system is divided into three modules. The expanded link extractor module focuses on entity extraction (through keyword matching) and linking between the code, logs, and other data sources. This module generates and extends user Doxygen documentation (see Figure 2a). The link querying module allows data flow analysis through link exploration and ranking (see Figure 2b). The third


My Project: GW.Agency Class
file:///sourcedb/test-example/html/class_g_w_1_1_agency.html

Caputes the Agency name More...

Public Member Functions

string getAgencyName ()

Static Public Member Functions

static void Main ()

Properties

string CODE [get, set]
Agency's CODE More...

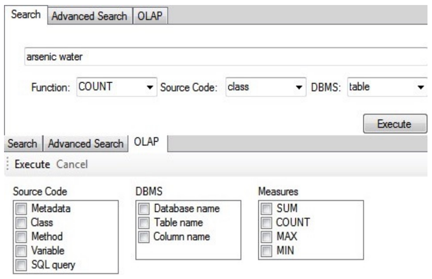
string AGENCY [get, set]
Captues the agency's name More...

Discovered Links

Database:
GW
Tables:
agency
Column names:
c_name

Detailed Description

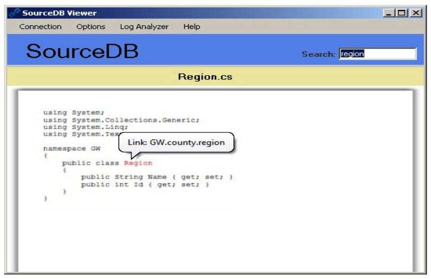
a) Doxygen Extension.



arsenic water

Function: COUNT Source Code: class DBMS: table

b) Assisted Exploration.



Search: region

Region.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Tex

namespace GW
{
    public class Region
    {
        public String Name { get; set; }
        public int Id { get; set; }
    }
}
```

c) Source Code View.

Fig. 2. SourceDB.

module presents an interactive view of the extracted links in the code (see Figure 2c).

A. *Link Extraction*

Data traces or links are the result of keyword matches between elements in the source code and the data source. The first phase of link extraction gathers all the shared keywords between the source code text files and the content in the database, starting with the database schema. Since the application's logs do not have a formally defined schema, the system performs a discovery process to infer the schema structure. Once the corresponding keywords have been identified, SourceDB then performs an approximate keyword matching to generate links between the sources. The result is triplets containing both sources and the linking keyword. To enhance the relationships among links, we also capture the membership relationships among the keywords. For example, if the keyword ‘location‘ is contained within a class named ‘store‘, SourceDB will capture the membership relationship (see Figure 1). The result of the link extraction phase is a set of connected graphs, where each graph represents a source, each node in the graph is a keyword, the relationship between keywords in the same graph are memberships, and the links between matched keywords are the discovered links. In this work, we improved this module with log analysis and documentation generation (Doxygen support).

1) Log Analysis: SourceDB takes previously generated logs and performs a schema inference process. In order to do so, SourceDB assumes a comma-separated type of log files (CSV), then proceeds to sample the log files and extract the most common keyword in each column. SourceDB will then take this keyword as the element to match against the other sources.

2) Doxygen Extension: After all the sources have been analyzed and all the links have been generated, our system generates a Doxygen report that presents the extracted links connecting keywords among the different sources. These appear in a section called ‘Discovered Links‘.

B. *Exploration and Querying*

This module explores the discovered links in three ways, by 1) allowing a boolean retrieval of all the links that approximate match a given keyword, 2) ranking the results of the search given the proximity of the match, and 3) providing OLAP-like summarization of the keywords. As a result, SourceDB is capable of answering analytical questions such as, “Which are the columns with the maximum number of links?” or “What is the column with the most links within a class?”

SourceDB is capable of querying and ranking the resulting links efficiently by taking advantage of materialized views and indexes. SourceDB reduces the number of string comparisons by an efficient early pruning of approximate matches that results in an integer key that is used to generate optimized queries that exploit indexed tables. The interactive querying and exploration mechanism exploits SQL queries and indexed summarization tables to provide an efficient answer to our users. Analytical queries are computed from materialized views that support OLAP-like exploration of the links. Details of the proposed OLAP algorithm are shown in [2].

C. *Visualization*

The visualization module is focused on simplifying the interaction and exploration of the resulting graph. We provide two main mechanisms for such exploration: 1) an interactive source code view and 2) a simplified OLAP exploration module with predefined queries. The first module renders each source code file and allows the users to observe the matching keywords and their sources. In other words, the users can observe the data flow between the code and the data sources. The second visualization mechanism presents a simplified interface that allows the customer to obtain aggregation and summarization on the links.

This exploration tool allows the user to query the links interactively (shown as advanced querying) and in an assisted form for obtaining basic summarizations and OLAP cubes (e.g. drilling-up or down). Also, a simple rank of links can be obtained by not selecting a function. The set of available functions includes SUM, COUNT, MAX and MIN. The main advantage of using assisted querying instead of the advanced querying is that optimized SQL is generated automatically by the application for efficient retrieval. On the other hand, advanced querying allows interactive querying capabilities and the possibility to extend the system through UDFs.

D. *Implementation*

Each of the described modules in SourceDB is implemented entirely in C#. However, the link extraction and querying modules are mostly driver code since the bulk of the computations and retrieval is performed in a DBMS. In other words, the entire process of link extraction (with the exception of log schema inference) and querying is performed within the DBMS via SQL queries or the database's extensibility features such as User Defined Functions (UDFs) or Stored Procedures.

The log schema inference mechanism is implemented in C# and the extracted keywords are stored in the DBMS for link discovery. Once the links have been generated, the user could choose to augment the Doxygen documentation to include the newly discovered links information. For this process, SourceDB groups all the corresponding links per source code file and adds that information to the html file.

The visualization mechanism is the largest component implemented outside the DBMS and allows for a user-friendly exploration of the resulting links. The visualization module presents a view of the source code and highlights the corresponding links. A second screen allows assisted querying for DOLAP-type of queries. In this view, the user could interact with the system by just selecting pre-canned computations that exploit the materialized views. The last view of this module allows the user to perform interactive queries by using our SQL interface.

III. Experiments

We evaluated our system with four source code repositories of varying sizes and complexities. The source code and sources, in this case, databases, are the following: 1) an exploratory tool of the water quality of wells in the State of Texas (WP), 2) the Sphider open source search engine (SE), 3) an inventory system (PJ1), and 4) a CRM-like system (PJ2). Table I and Table II contain the details of each repository and schema. Our experiments were run on an Intel Xeon E3110 server at 3.00 GHz with 750 GB of hard drive and 4 GB of RAM. The server was running an instance of a commercial database management system. All the experiments are the result of an average of 30 runs and all times are in seconds.

TABLE I
Source Code Repository.

Description	WP	SE	PJ1	PJ2
Num. Files	52	44	119	316
Num. Classes	52	0	158	460
Avg. File Size (KB)	1,409	4,787	12,388	11,474
Lines of Code (LOC)	5,488	5,463	28,180	70,815
Avg. Num. Variables	8	29	19	35

TABLE II
Sources.

Description	WP	SE	PJ1	PJ2
Num. Tables	52	25	21	95
Avg. No. Columns	7	4	6	5
Max No. Columns	110	12	15	29
Min No. Columns	1	2	2	2

SourceDB explored the largest source in less than 40 seconds and managed to process the rest in 20 seconds or less. As part of our experiments, we observed that performance of the link discovery phase is subject to two dominant operations: I/O operations due to the loading of the source files, and the approximate keyword matching. An important observation is that the number of discovered links is not correlated to the size of the source code repository but rather to the business logic of the application. For example, CRM-type of applications that perform a large number of insertions into a data store tend to generate a larger number of links.

Since our system is backed by rich database optimizations and materialized views, it is possible to perform complex graph exploratory queries in less than 20 seconds for the sources with the largest set of discovered links. Query performance is dependent on the number of keywords to search and their selectivity.

IV. Conclusion and Future Work

In summary, our original approach allows users to simultaneously explore, query, and search databases, logs, and source code repositories within seconds in order to understand their data flows. This allows further debugging capabilities and offers better guidance to software engineers for understanding software vulnerabilities and application maintainability. We believe that our main contribution is that our approach augmented with log analysis can be used to present a different view of traditional source code analysis tools.

Our future work will focus on further exploiting the extracted links from the application with some machine learning and artificial intelligence algorithms in order to extract additional information from the dependencies within the legacy program. In addition, we want to expand our log analysis tool to provide a better data flow detection. Finally, we have also identified that it is possible to parallelize our approach, or even use cloud infrastructure, to reduce the link discovery time significantly.

References

[1] Atzeni, M., and Atzori, M. Codeontology: Querying source code in a semantic framework. In *Proc. of ISWC)* (2017), pp. 4.

[2] Garcia-Alvarado, C., Chen, Z., and Ordonez, C. Olap-based query recommendation. In *Proc. of ACM CIKM* (2010), pp. 1353–1356.

[3] Garcia-Alvarado, C., and Ordonez, C. Keyword Search Across Databases and Documents. In *Proc. ACM SIGMOD KEYS Workshop* (2010), pp. 2.

[4] Garcia-Alvarado, C., and Ordonez, C. Integrating and querying web databases and documents. In *Proc. ACM CIKM Conference* (2011), pp. 2369–2372.

[5] Garcia-Alvarado, C., and Ordonez, C. Integrating and querying source code of programs working on a database. In *Proc. ACM SIGMOD KEYS Workshop* (2012), pp. 47–53.

[6] Garcia-Alvarado, C., Ordonez, C., and Baladandayuthapani, V. Querying External Source Code Files of Programs Connecting to a Relational Database. In *Proc. of ACM PIKM* (2012), pp. 9–16.

[7] Garcia-Alvarado, C., Ordonez, C., and Chen, Z. DBDOC: Querying and Browsing Databases and Interrelated Documents. In *Proc. ACM SIGMOD KEYS Workshop* (2009), pp. 47–48.

[8] Ordonez, C., Chen, Z., and García-García, J. Metadata management for federated databases. In *ACM CIMS Workshop* (2007), pp. 31–38.

[9] Telea, A., and Voinea, L. Interactive visual mechanisms for exploring source code evolution. In *Proc. IEEE VISSOFT Workshop* (2005), pp. 1–6.