Sourcerer: A search engine for open source code supporting structure-based search

Conference Paper · October 2006

DOI: 10.1145/1176617.1176671 · Source: DBLP

CITATIONS
197

READS
938

7 authors, including:

Erik Linstead
Chapman University
92 PUBLICATIONS **1,451** CITATIONS

SEE PROFILE

Paul Rigor
University of California, Irvine
14 PUBLICATIONS **1,038** CITATIONS

SEE PROFILE

Pierre Baldi
University of California, Irvine
466 PUBLICATIONS **32,804** CITATIONS

SEE PROFILE

Sourcerer: A Search Engine for Open Source Code Supporting Structure-Based Search

Sushil Bajracharya[1], Trung Ngo[1], Erik Linstead[2], Yimeng Dou[2], Paul Rigor[2], Pierre Baldi[2], Cristina Lopes[1]

[1]Institute for Software Research [2]Institute for Genomics and Bioinformatics
Donald Bren School of Information and Computer Sciences
University of California, Irvine
{sbajrach,trungcn,elinstea,ydou,prigor,pfbaldi,lopes}@ics.uci.edu

Abstract

We present *Sourcerer*, a search engine for open-source code. *Sourcerer* extracts fine-grained structural information from the code and stores it in a relational model. This information is used to implement a basic notion of *CodeRank* and to enable search forms that go beyond conventional keyword-based searches.

Categories and Subject Descriptors H.3.3 [*Information Storage and Retrieval*]: Information Search and Retrieval—Search process

General Terms Languages

Keywords Source Code Search, Code Rank

1. Introduction

This paper focuses on the current research goals and search capabilities of *Sourcerer*. *Sourcerer* enables searches that are based not just on keywords but also on the structural properties and relations among program elements. Current version of *Sourcerer* works with the open source projects implemented in Java. The first release of *Sourcerer*, as of time of submission of this paper, is publicly available in its development version at `http://sourcerer.ics.uci.edu/`.

We are progressing towards the following goals with our research efforts in *Sourcerer*:

Improved search results and novel search capabilities. By clarifying the concept of code search, *Sourcerer* is able to match the search needs better than the other search engines. In the case of structural search, *Sourcerer* isn't just improving, but it actually is providing a search service that the other search engines don't provide. For this purpose we formulate three major search forms: implementations, uses, and structure, at different granularities. Currently *Sourcerer* supports five types of searches: 1) components; 2) component use; 3) functions; 4) function use; and 5) program structures (fingerprints) under these forms.

Code search benchmark. We are currently working on a benchmark for open source code search that can be used by other code search related projects to compare their results with ours. A benchmark like this will foster friendly competition and is essential if we are to see rapid improvements in this field.

Experimental evaluation. We have implemented a first working version of the *Sourcerer* infrastructure. To explicitly test its ability to find relevant code, we populated our database with more than 1500 real-world open source projects corresponding to 254,049 Java classes, and over two million entities and eleven million relations. We have issued thousands of search requests as part of our ongoing work. Some sample queries and the result from *Sourcerer* are presented in Section 2.

Three highlights of the *Sourcerer* system are: its use of relational representation of the source code that facilitates indexing fine grained information from the source code, use of *CodeRank* as ranking technique for prioritizing search results that is based on Google's Pagerank technique [3], and use of *Fingerprints* - a vector representation of many interesting attributes of the entities from the source code that facilitates various structured based search modes. Fingerprints denote both the presence and multiplicity of specific programming constructs within individual code entities. Currently *Sourcerer* provides three forms of fingerprint search: 1) Control Structure: provides information about concurrency, iteration, and branching constructs within the code; 2) Java Type: captures information about object oriented constructs such as classes, methods, attributes, constructors etc; and, 3) Micro Patterns: provides information as to whether or not simple design patterns are present within a code entity [1].

The *Sourcerer* infrastructure comprises various components ranging from source code repositories as data sources to custom built parsers. We heavily use many successful open source products in building the system itself. Further details on each system components are available at the project web site `http://sourcerer.ics.uci.edu/about.html`.

2. Sample Queries and Results

Table 1 shows the search capability of *Sourcerer* using the following sample queries.

Q1. Find a XML parser

Q2. Find code that uses XML parser

Q3. Find implementation of Depth First Search

Q4. Find the use of Depth First Search

Q5. Find example code that creates a ZIP file

Q6. Find the implementation of Newton Raphson Method

Q7. Find the use of Newton Raphson Method

Q8. Find the implementation of Semaphore

OOPSLA'06 October 22–26, 2006, Portland, Oregon, USA.
ACM 1-59593-491-X/06/0010.

Qry.	Keywords	Mode	Results	H/P*
Q1	XMLparser	C	org.apache.jasper.xmlparser	1/1
			org.japano.jasper.xmlparser	2/1
			org.apache.lucene.xmlparser	4/1
			com.ibm.uima.util.XMLparser	8/1
Q2	XMLparser	C-use	org.japano.jasper.compiler	1/1
			org.apache.jasper.compiler	2/1
			com.ibm.uima.taeconfigurator.editors.ui	4/1
Q3	Depth + First + Search	F	org.mojave.jeat.search .dfs.BasicDepthFirstSearch	1/1
			jopt.csp.spi.search.technique .Depth-FirstSearch	10/1
Q4	Depth + First + Search	F-use	org.openscience.cdk.graph.SpanningTree	1/1
Q5	create + zip + file	F	jwsgrid.util.Zip.createZipFile(..)	1/1
		F-use	jwsgrid.jobhost.JobHost	1/1
		F	de.siemens.fast.core.container.plugin .transfer.zip.ZipArchive.create(..)	2/1
		F-use	de.siemens.fast.core.container.plugin .transfer.zip.ZipArchive	2/1
			de.siemens.fast.core.container.plugin .transfer.zip.ZipTransferClient	7/1
Q6	Newton + Raphson	F	org.openscience.cdk.modeling.forcefield .NewtonRaphsonMethod	1/1
Q7	Newton + Raphson	F-use	org.openscience.cdk.modeling.forcefield .AngleBending	1/1
			org.openscience.cdk.modeling.forcefield .GeometricMinimizer	2/1
Q8	semaphore	F	net.sf.camelseye.util.thread.Semaphore	1/1
			com.marringtons.util.Semaphore	3/1
			serp.util.Semaphore	6/1
Q9	semaphore	F-use	com.marringtons.object.Database	1/1
			net.sf.camelseye.DialogManager	3/1
			tyRuBa.util.pager.DiskManager	2/4
Q10	NA	Fi-C	initex	1/1
			koala.dynamicjava.parser.Parser	2/1
			bsh.parser	3/1
Q11	NA	Fi-C	ictk.util.Log.errorReport(..)	1/1
			naarani.Exception.SequenceException	2/1
Q12	NA	Fi-T	cern.clhep	1/1
			com.dbxml.util.ObjectQueue .ObjectQueueItem	3/1
			tyRuBa.util.LinkedList.Bucket	6/1

Table 1. Sample Search results (*H/P corresponds to Hit/Page, where x/y means hit x on page y; Mode listed are: C = component, C-use = Component USE, F = Function, F-use = Function USE, Fi-C = Control Structure Fingerprint, Fi-T = Type Fingerprint ; '+' in the keywords imply logical AND for the individual keyword)

Q9. Find the use of Semaphore

Q10. Find methods containing more than 50 switch statements

Q11. Find methods containing 1 switch statement, 1 if statement and no memory allocations (new operator)

Q12. Find private entities that have fields of their own type and do not have any declared methods

Q1 is an example of finding complete open source components. This is useful for projects that need to incorporate functionality provided by those components as a whole. Q3, Q6, and Q8 aim at finding implementations of specific functions that can be available either as complete projects or parts of projects. This is useful for using those functions directly or simply for incorporating their implementation strategies into the searcher's project. Q2 aim at finding where certain projects are being used. This is useful for developers of open source projects, so that they can track what other projects are using their code. Q4, Q5, Q7, and Q9 aim at finding example code that *uses* certain functions or APIs. This is useful for quickly getting base code that successfully interacted with certain APIs; that base code can then be adapted to the searcher's needs. Q10, Q11, and Q12 aim at finding code with specific structural properties. This is useful for projects whose domain is software itself, and that need uncontrolled, real-world code for testing.

All queries resulted in multiple pages of search results (1 page = 10 results) and most of the the results were above hundred in numbers. We were able to find interesting results right in the first page. Results shown in Table 1 are some selected results that closely satisfy the purpose of their respective queries.

3. Related Work

There has been some earlier works in ranking code entities. Mandelin et al. developed a heuristic to rank *jungloids* (code fragments) by length, assigning the top ranks to the shortest jungloids [4]. Their heuristic is elegant but too simple to rank the relevance of search results in a large repository of programs. Inoue et al. designed a component rank model in order to search software systems more efficiently [2]. Using the component rank model, classes frequently used by other classes generally have higher ranks, whereas nonstandard and special classes typically have lower ranks. We are extending this notion of ranking to much finer granularity and experimenting with various ranking schemes using several combinations of relations we have modeled. It is our intention to modify our code ranking scheme in the near future, to tune the ranking according to different criteria. For example, we expect to be able to detect class frameworks vs. class libraries, since the former tend to have several external classes that inherit from them, and the latter are usually instantiated and invoked. For purposes of ranking, certain links seem to be more important than others.

Recent work is trying to bring the capabilities of web search engines into code search. Koders (`www.koders.com`), Codase (`www.codase.com`) and Krugle (`www.krugle.com`) are few prominent systems. They work by narrowing down the information space, i.e. by indexing code files, but they only touch the surface of what can be done in extracting features from the code itself.

4. Conclusion

The *Sourcerer* infrastructure is scalable enough to crawl the open source repositories in the Internet and, index the database of entities and relations extracted from the source code. Currently we have successfully indexed more than 1500 Java projects from Sourceforge exceeding 18 million lines of code. The infrastructure and the indexed information so far is serving as a basis for various empirical studies on source code such as effects of various ranking schemes in search results.

References

[1] J. Y. Gil and I. Maman. Micro patterns in java code. In *OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object oriented programming systems languages and applications*, pages 97–116, New York, NY, USA, 2005. ACM Press.

[2] K. Inoue, R. Yokomori, T. Yamamoto, and S. Kusumoto. Ranking significance of software components based on use relations. *IEEE Transactions on Software Engineering*, 31(3):213–225, 2005.

[3] R. M. Lawrence Page, Sergey Brin and T. Winograd. The pagerank citation ranking: Bringing order to the web. *Stanford Digital Library working paper SIDL-WP-1999-0120 of 11/11/1999*.

[4] D. Mandelin, L. Xu, R. Bodík, and D. Kimelman. Jungloid mining: helping to navigate the api jungle. In *PLDI '05: Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, pages 48–61, New York, NY, USA, 2005. ACM Press.