An Unabridged Source Code Dataset for Research in Software Reuse

Werner Janjic*, Oliver Hummel†, Marcus Schumacher*, and Colin Atkinson*
*Software-Engineering Group,
University of Mannheim, Germany
{werner, schumacher, atkinson}@informatik.uni-mannheim.de
†Institute for Program Structures and Data Organization,
Karlsruhe Institute of Technology, Germany
hummel@kit.edu

***Abstract*—This paper describes a large, unabridged data-set of Java source code gathered and shared as part of the Merobase Component Finder project of the Software-Engineering Group at the University of Mannheim. It consists of the complete index used to drive the search engine, www.merobase.com, the vast majority[1] of the source code modules accessible through it, and a tool that enables researchers to efficiently browse the collected data. We describe the techniques used to collect, format and store the data set, as well as the core capabilities of the Merobase search engine such as classic keyword-based, interface-based and test-driven search. This data-set, which represents one of the largest searchable collections of source and binary modules available online, has been recently made available for download and use in further research projects. All files are available at http://merobase.informatik.uni-mannheim.de/sources/**

I. Introduction

The creation of a searchable software repository has long been seen as a key step for increasing reuse levels in software development [10]. However, even 15 years ago, collecting more than just a few hundred reusable code entities was a nearly hopeless undertaking since most development companies kept their source codes private (and are still doing this today for understandable reasons). However, the rise of the Internet and the "open-source revolution" opened a new opportunity for building large collections of searchable software, prompting Mili et al. [9] to predict a need for more powerful software retrieval solutions back in the late 1990s. Nevertheless, harvesting publicly available software over the Internet is not as easy as it might appear. The widely-recognized failure of the *UDDI Business Registry* in 2006 [1] demonstrated that service and component repositories cannot passively wait for developers to populate them, since the quality of voluntarily uploaded information invariably degrades over time. Instead, it is necessary to actively crawl for searchable material and take regular steps to continuously maintain the quality of an index built upon it. This mirrors the evolution of search mechanisms in mainstream web search engines where the first generation of manually maintained web directories like *Yahoo!* were almost completely superseded by crawler-based search engines such as *Google*.

[1]The data set includes code retrieved from CVS/SVN repositories without http sources, which are downloaded by the search engine just-in-time.

II. The Crawling Process

The main part of the data collection and index infrastructure described in this paper were created in 2006 and 2007 in two distinct crawling phases. Starting from a few hand-collected seed URLs we initially crawled the open web using the *Nutch* crawl tool which is part of the *Lucene* search engine project of the Apache Software Foundation [5]. The second major crawling effort involved the collection of data from major open source hosting sites such as *SourceForge*, *JavaForge*, etc. In the following, we provide a more detailed description of the approach used to crawl the dataset, the data structures used to store the gathered information in Lucene and the techniques used to make these resources searchable.

We used the open source tool Nutch to gather the data for building the index, regardless of whether the resources were stored locally (e.g., files downloaded from the version control systems of hosters for open source projects) or were available over the open web (via the http-protocol). Crawling the web for reusable source files had limitations since the discovered files were often relatively isolated. Furthermore, most open source hosters excluded crawlers from their browsable repositories on the web (via *robots.txt*) so that using the locations of CVS and SVN repositories was a more sustainable solution. Hence, the bulk of the index was created by downloading complete open source projects in order to locally crawl the "mirrors" with Nutch. In order to further increase the output of our crawls we also used Nutch to download further URLs contained in the analyzed source files.

The collected data was indexed and made searchable using the popular document-based data indexing framework called Lucene, which is also available as an open-source project. In contrast to relational databases, which might appear a more natural choice for this purpose at a first glance, Lucene is the much more efficient solution for text-based searches over software artifacts since source codes can be better stored and more efficiently analyzed when stored as textual documents than as tables in a relational database. As well as being optimized to store relationships rather than to support keyword-based searches, the open-source relational databases available at that time did not offer useful relevance ranking mechanisms

978-1-4673-2936-1/13 © 2013 IEEE

MSR 2013, San Francisco, CA, USA

TABLE I
CONTENT OF THE CVS/SVN SOURCE FILE DATASET.

Granularity	# Files
All files	8,964,433
Java source files	2,429,999
Java class files	203,689
.jar containers	27,168
Java source files within .jar containers	40,692
Java class files within .jar containers	4,342,376
Java methods	22,262,954
LOC	182,224,390
C files	1,046,239
C# files	193,949

for ordering the results. In other words, a row in a relational database is either recognized as belonging to the return set for an SQL query or not. With the application of a few "tricks" to the index data, a *Full-Text Search Framework* like Lucene can be enabled to support advanced searches for things like the interface of Java classes.

III. REPOSITORY CONTENT

To make the contents of Merobase's code repository available for other researchers we have created a downloadable bzipped tarball file (compressed approximately 15 GB, unpacked approximately 50 GB) archived on June, 2nd 2010. Its containment is listed in table I.

An interesting aspect of the archive is the size that individual classes can reach. For example, the largest file in the dataset, the `AMD64RawAssembler` class, contains 51,860 lines of code within 7,550 methods, followed by the `StdOverloadedList` class with 50,680 lines of code in 4,854 methods. Furthermore, in addition to these numerical properties we analyzed various other characteristics like the used open source licenses or the types of artifacts (i.e. whether they are applets, EJBs or test cases) and also measured several software metrics.

IV. INDEX STRUCTURE

Like Merobase, other code search engines – from academia and industry – such as Sourcerer [2] or Krugle [4] also use the Lucene search framework to index their data. Today, Lucene is the first choice in terms of scalability and performance for text-based search applications. However, in contrast to other search engines, Merobase not only stores the textual elements of the software artifacts (i.e. usually their source code, it also extracts a variety of analytical information and stores it in additional fields of the index. Examples include an artifact's project name, type and hash value, which can be used to determine the number of duplicates in the dataset. Table II shows the basic attributes in the schema used in the Merobase index. Mili et al. [9] call such an approach a faceted classification scheme, where each field of the scheme describes a different facet of the software artifact.

As depicted in Table II, not all fields of the schema are unique for each software artifact. Instead several fields have a multiplicity that is greater than one, such as *method*, for example. This reflects the fact that a software component can contain more than one method or, if one considers method overloading, that even a method with the same name can appear more than once. To handle such cases, Lucene allows multiple fields to have the same tag name what we illustrate with the multiplicity indicator. In contrast, there can also be fields that are not filled when no appropriate information can be extracted from the indexed artifact.

Lucene is able to perform searches on all available fields to find and retrieve artifacts matching a user's query. Queries can also be composed over various fields using boolean operators. The fact, that these searches can be carried out with high performance results from the special tokenizing technique used to store the indexed text. The so-called Lucene Analyzer is responsible for extracting the contents for each field and splitting them into several tokens that can be stored within Lucene's dedicated tree structure. While this is usually an advantage, it can, however, have some undesirable consequences when it comes to more complex searches for software interfaces. Consider the following example illustrating how the interface of an operation is stored in the index in order to support concrete searches:

```
mn:doSomethingUseful:pt_int:pt_int:rt_String
```

In order to find a method `doSomethingUseful` with two integer parameters returning a string, for example, we have to circumvent Lucene's normal tokenizing behavior so that we are able to store method interfaces within a class in their full form. As Lucene would normally split signatures (as well as any other text) containing white space characters,

TABLE II
INDEX SCHEMA

Field	Description	Mult.
content	source code	1
url	source URL of the artifact	1
host	the hostname in the URL	1
name	the artifact's name	1
lang	the programming language	1
form	source or binary	1
dependency	recognized dependencies	0..*
kind	applet, test case, EJB, etc.	1
namespace	the artifact's namespace	0..1
extends	direct superclass of this artifact	0..1
implements	names of implemented interfaces	0..*
method	the contained operation names	0..*
methodSig	Signature of contained methods	0..*
methodParamsSyntax	parameters of the contained methods	0..*
license	(open source) license	0..1
author	author(s) of the artifact	0..*
duplicates	amount of verbatim duplicates	0..1

we use the underscore to separate the method names (“mn”), the parameters (“pt”) and the return types (“rt”) from each other. In order to create an efficiently searchable string from a parsed signature, the parameters of each method are ordered alphabetically so that even signatures with varying parameter orders can be matched. Obviously, all search requests have to be transformed into this format by a query parser before searches are performed on the index. For this purpose, we created a special Java parser that is able to translate Java files into an appropriate search request. Since we wanted to be able to also search for pure method signatures (i.e. ignoring the method names), we defined an additional field that omits the *mn:doSomethingUselful* in the above case.

As mentioned previously, Lucene is much more efficient than relational databases when it comes to full-text searches, but it is not as good at storing relationships. This is a disadvantage that manifests itself whenever a user is interested in searches that depend on relationships within components, such as defining multiple methods for one class or relationships in between classes such as imports and inheritance dependencies. While the former can be easily solved by concatenating method signatures, the latter involves more effort and is currently under investigation in our research group to extend the underlying repository of the Merobase search engine in a way that offers the best of both technologies.

A. Search Capabilities

Based on the index structure explained before, Merobase supports three main types of searches, namely pure Lucene-based text matching searches that can use any combination of free text and the fields listed above as well as interface-driven searches for operations or complete classes. As an example of the first, assume that we are looking for a poker game class in Java that contains a deal method:

name:pokergame lang:java method:deal

Lucene queries are by default case insensitive and AND concatenated. As mentioned before, Merobase is able to parse Java/C# code so that the following query for a Matrix class containing an add and a multiply method is also possible:

```
public class Matrix {
   public Matrix add(Matrix) {}
   public Matrix multiply(Matrix) {}
}
```

If no exact match on such a class can be found special heuristics are applied in order to retrieve the potentially most relevant candidates first. These heuristics are also the reason that a plain search for “matrix”, for example, also delivers usable matrix classes first.

In order to be able to execute searches for method signatures while ignoring the method name the so-called Merobase query language (MQL) needs to be used to formulate a query like *$(int):boolean;* in which the dollar sign is used as a wildcard for the method name so that any method expecting an int parameter and returning a boolean value is delivered.

Due to space limitations we are unable to provide more details here, and refer the interested reader to the merobase.com website for more example queries. More details on the index structure have already been published elsewhere [6].

B. Tool Support

In order to make it easier to browse and study our data set without the need for creating an own search engine, a tool called MeroL is also included in the downloadable resources that serves as a local front-end to the index and the sources. In this subsection we give a brief overview of how MeroL can be used to browse our data-set. It is necessary to navigate to

http://merobase.informatik.uni-mannheim.de/sources/

where the crawled sources, the Merobase index and MeroL can be downloaded. Provided that a Java VM is installed, the tool can be started from the command line with

```
java -jar -Xms2048m MeroL-<version>.jar
```

The startup parameter ensures that there is enough memory allocated to the VM so that even large result sets can be processed. After the program has started, the user should create a new project using the File menu. This is achieved by defining the location where the project data should be stored and entering the location of the index. If the downloaded index was for example unpacked in */home/user/* the correct information provided to the program would be */home/user/indices/public/finalIndex/*. It is important that the *public* folder does not contain anything but directories (like, for example a hidden .DS_Store file on a Mac) since this seems to confuse Lucene in some settings.

Subsequently, the location of the source files can be set via the *Project* menu. The CVS and SVN directory location should point to the *MEROCVSROOT* respectively *MEROSVNROOT* directory of the unpacked source files. After this minimal configuration effort, the tool is ready to perform searches and to let the user inspect the retrieved results. Figure 1 shows a screenshot of a search for a downloaded method that takes a String as input parameter and returns a File object as result.

V. Ongoing Work

In order to address the central weakness of a pure Lucene index, which makes it hard to handle relational associations between two software artifacts like associations dependencies, we are currently working on a new and enhanced index which is able to store this information. The obvious solution is to combine the power of Lucene in the area of text-based searches with an SQL compliant database for the relational information. This approach has already been investigated by Sourcerer [2] for example. Additionally, we are currently also analyzing different NoSQL solutions, as well as graph-based databases, since software artifacts with their various connections induce a high volume of joins into SQL databases, bearing an increased risk of performance degradation. For the crawling process we continue to use Nutch, as the latest version offers a further simplified integration of self-written parsers, as well as a

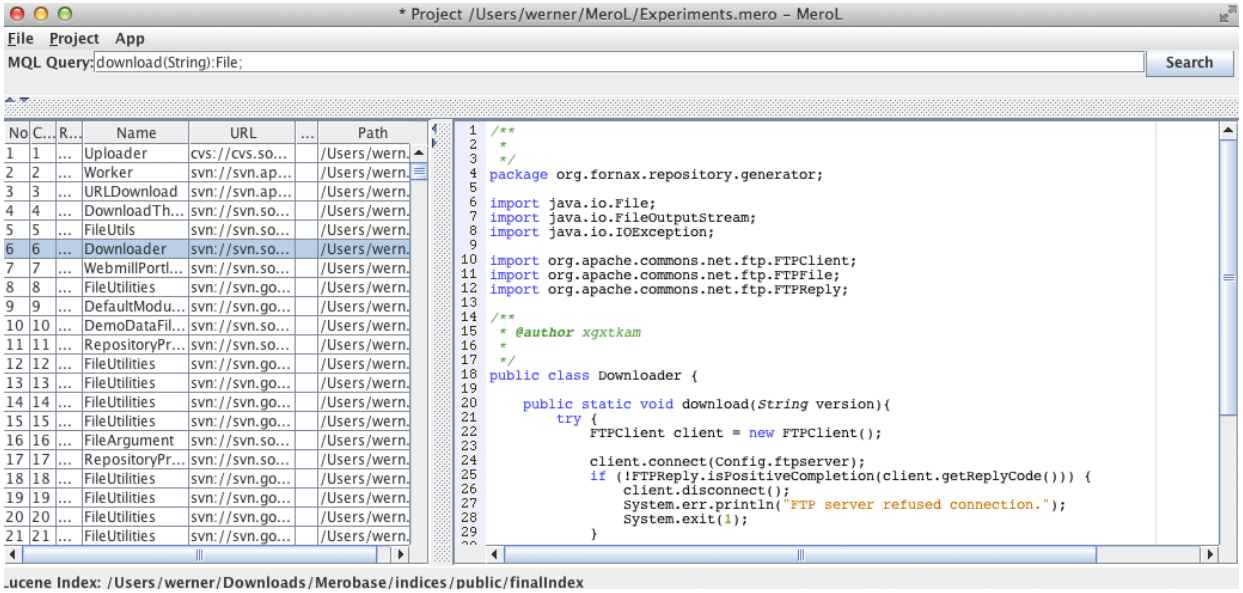

Fig. 1. A search for a "download" method with a String parameter for the source URL returning a File object for the downloaded resource.

new MIME-type detection which simplifies the recognition of source-code artifacts on the world wide web, even if they are not available with their normal file extension. The only weakness of Nutch, as with most other existing web crawlers, is the missing ability to crawl and gather the content of AJAX based websites. This weakness arises through the fact, that AJAX based web sites do not provide the whole contained information at loading time. Instead, based on the events a user can trigger, additional content is dynamically loaded and injected into existing HTML containers, while old content may be removed. Like described in Duda et al. [3], a crawler that would be able to gather the whole information provided by AJAX based websites, needs to know all existing events in order to get all possible representations of a site. However, since many source code hosters, like SourceForge or GitHub, are using this technology, we are facing this problem during the crawl process and are currently trying out several solutions to detect and parse dynamically loaded source code.

VI. Conclusion

We have made the data set and the index underlying the Merobase software search engine publicly available with the hope that researchers find the artifacts contained useful and will perhaps create novel innovative applications for it. In recent work we have talked about the importance to establish a common set of reusable artifacts and retrieval challenges in order to create a reference collection that can serve as a benchmark for large-scale software search and reuse tools [7], which could also be one usage scenario of this data set. Hence, we would like to renew our call to other researchers to join forces in this area and to use the Merobase data as a basis to create a significant reference collection allowing to evaluate software retrieval approaches. As stated in earlier publications, our initial proposal in this field includes creating definitive queries for concrete reusable artifacts in the form of test cases that can be used to determine free of doubt whether a delivered candidate will be usable in a given context specified by the test case [8].

Acknowledgements

We would like to thank all participants of the Merobase project for their contributions. Especially, we would like to thank our former "Crawlmeister" Philipp Bostan for his outstanding work in compiling the index.

References

[1] C. Atkinson, P. Bostan, O. Hummel, and D. Stoll. A practical approach to web service discovery and retrieval. In *IEEE International Conference on Web Services, 2007.*, pages 241–248. IEEE, 2007.

[2] S. Bajracharya, T. Ngo, E. Linstead, Y. Dou, P. Rigor, P. Baldi, and C. Lopes. Sourcerer: a search engine for open source code supporting structure-based search. In *21st ACM SIGPLAN symposium on Object-oriented programming systems*, OOPSLA '06, pages 681–682, New York, NY, USA, 2006. ACM.

[3] C. Duda, G. Frey, D. Kossmann, R. Matter, and Chong Zhou. Ajax crawl: Making ajax applications searchable. In *Data Engineering, 2009. ICDE '09. IEEE 25th International Conference on*, pages 78 –89, 29 2009-april 2 2009.

[4] E. Hatcher, O. Gospodnetic, and M. McCandless. Lucene in action (2nd edition), 2010.

[5] O. Hummel and C. Atkinson. Using the web as a reuse repository. In *Proc. of the 9th Intl. Conference on Reuse of Off-the-Shelf Components*, ICSR'06, pages 298–311, Berlin, Heidelberg, 2006. Springer-Verlag.

[6] O. Hummel, C. Atkinson, and M. Schumacher. *Finding Source Code on the Web for Remix and Reuse*, chapter Artifact Representation Techniques for Large-Scale Software Search Engines. Springer, 2013.

[7] O. Hummel and W. Janjic. Towards better comparability of software retrieval approaches through a standard collection of reusable artifacts. *Proceedings of the 7th International Conference on Software Engineering Advances*, page 450 to 458, November 2012.

[8] O. Hummel, W. Janjic, and C. Atkinson. Code conjurer: Pulling reusable software out of thin air. *IEEE Software*, 25(5):45–52, September 2008.

[9] A. Mili, R. Mili, and R. T. Mittermeir. A survey of software reuse libraries. *Ann. Softw. Eng.*, 5:349–414, January 1998.

[10] R. C. Seacord. Software engineering component repositories. In *Proc. of the Intl. WS on Component-Based Software Engineering*, 1999.