Federated Search for Open Source Software Reuse

Fotios Kokkoras[1]	Konstantinos Ntonas[2]	Apostolos Kritikos[3]	George Kakarontzas[1,3]	Ioannis Stamelos[3]
fkokkoras@teilar.gr	kntonas@gmail.com	akritiko@csd.auth.gr	gkakaron@teilar.gr	stamelos@csd.auth.gr

1 TEI of Larisa
Dep. of Computer Science & Telec/tion,
411 10, Larisa, Greece

2 International Hellenic University
14th km Thessaloniki – Moudania
570 01, Thermi, Greece

3 Aristotle University
Dep. of Informatics
541 24, Thessaloniki, Greece

Abstract - **Source code search engines assist the software development process by providing a way of searching for free source code in code repositories. Although their use is rather straightforward, there exist a few of them and the differences in the way they index and provide access to their assets require considerable time and effort from the programmer to use them. In this paper, we present a federated open source code search engine that simultaneously asks, in real time, existing open source code search engine sites and detail the way we overcome the integration obstacles, by combining provided APIs, browser automation and web content extraction techniques.**

Keywords - open source code search engines; federated search; web extraction

I. INTRODUCTION

The concepts of Software Reuse [1] and Rapid Development have been adopted by large software development companies, small and medium enterprises (SMEs), research institutes and freelancers. According to a survey conducted in [2], software reuse in general and Free/Libre Open Source Software (F/LOSS) reuse in particular are important for the software development SMEs for a series of reasons:

- Reuse has a positive effect on lowering the development costs (91%), shortening the development and testing time (83%), increasing the quality of the final product (76%) and shortening time to market (72%).
- In relation to the different artifacts that can be reused, source code is the most important (87%), followed by design (80%) and documentation (75%).
- Almost half of the organizations (51%) have an in-house reuse repository whereas 39% have some formal process for reusing components they develop.
- The vast majority of the respondents (80%) said that their organization supports OSS reuse.

Meanwhile, millions of lines of reusable code have become available in the different source code forges and, in many cases, lots of alternatives exists for specific functionality [3]. This availability (and "redundancy") uncovered the need for effective ways of discovering reusable source code.

Source code forges (SourceForge, Git, Bitbucket, etc.) provide ways for internal navigation and search (such as categories, tags and internal search engines). For the reuse engineer however, who's primarily goal is to find the component that best fits the needs of the functionality he wants to implement, searching to each source code forge separately, creates a significant overhead. This is also depicted in [2], where the most important factors preventing OSS reuse were the lack of documentation (80%), the uncertainty on the quality of OSS components (76%), and the difficulty in searching and retrieving OSS components (66%).

Web-based source code search engines follow the architecture of classic, web search engines. Despite the differentiation of the nature of their data (that is, source code files), they provide crawling, indexing, reporting and ranking mechanisms identical to those of a typical web search engine. This is probably the reason why only 12% of the responders in [2] said that they have used a specialized OSS code search engine. They seem to prefer general purpose web search engines instead (e.g. Google). In fact, general purpose web search engines contribute more reusable components than specialized code search engines (65% vs. 31%). More significant is the fact that this 31% comes from the aforementioned 12% [2]. It seems like there is quality in the specialized OSS code search engines.

The above observations suggest that the current status of specialized OSS search engines leave much to be desired for the developers [4], since although they can be potentially an important source of reusable components, the developers do not view them as important enough to use them.

Finally, one cannot overlook the diversity of important sources of reusable components: in-house and public code repositories, specialized (code) and classical search engines. This, together with the difficulty reported earlier in searching and retrieving OSS components asks for a search mechanism able to provide results collectively, from different free/open source code sources.

Focusing on website-based code search tools, Krugle [5] and Koders [6] are among the most popular. They host source code on which they provide search services and also index other forges like Sourceforge. Merobase [7] can be mostly described as a code meta-search engine since it does not own a code repository but rather indexes and collects metadata from other sources on which it provides search services. Moreover it defines itself as component oriented search engine, meaning that, it can return sets of source code classes that implement a specific functionality.

These specialized code search engines are valuable but each one poses specific requirements to the user, like searching using a diversity of search forms with different criteria in each or interpreting differently presented results. This definitely creates cognitive overhead to the end user. Even avail-

ability is sometimes questionable and thus a source of frustration. Finally, dealing with more than one search points is more time consuming. Eventually, the "Google solution" becomes more attractive and the findings in [2] get justified!

Software reuse in general and OSS reuse in particular is important for the software development SMEs. To alleviate the problems mentioned and make the use of web-based code search engines more attractive, we propose a federated code search engine that provides the user with a single point to define his criteria-based search query, propagates the question to other code search engines in real time and finally presents the aggregated results to the user ([14], [15]). The proposed architecture can cooperate with individual code search engines either through an API or by using browser automation and web content extraction techniques.

The rest of the paper is organized as follows: Section 2 describes the proposed system whereas Section 3 gives implementation details by focusing on the web content extraction infrastructure used. Usage details are given in Section 4 whereas Section 5 concludes the paper and gives insight for future work.

II. PROPOSED SYSTEM

The federated code search engine we propose should be flexible enough to incorporate individual existing or future code search engines. Typically, one can retrieve data from another web source either through an API or via web content extraction. Having an API is preferred because it is faster and more reliable. Merobase belongs to this case. However, in cases like Koders and Krugle, which give their answers as http pages only, web extraction is the single option.

Web content extraction is the non-trivial process of collecting unstructured web data and storing them in a database or an XML file [16]. This is usually accomplished by pattern matching an html pattern (extraction rule or wrapper) with a target web page. Upon a successful match, a data record becomes available as data from the web page is unified with variables in the extraction rule. Tricky cases like record-data scattered on different html sub-trees, pages with more than one data records, data records distributed in many web pages as a result of some pagination procedure, incomplete data records that break the html pattern used, etc., make the extraction task non-trivial. Flexibility in extraction rule management by means of visual/GUI tools, deployment and orchestration (use many extraction rules in a cooperative fashion) are all required features from a web extraction solution that we want to last long.

The deployment diagram of the proposed system is depicted in Figure 1. Queries submitted via the single search form provided, are forwarded as http calls to one or more query services (this is a user preference) utilized by the query engine. Each of these services forwards the query to its own code search engine, collects the results in XML format and sends them back to the main system where they are collated and presented in HTML to the user. The system aims at reducing the time and effort required by a user to search all the individual search engines, offering a transparent search solution. It does not perform any actual asset indexing or search by itself.

A prototype (code name OCEAN [10]) of the federated code search engine described in Figure 1 has been implemented, in the context of the OPEN-SME project [11]. This project aims to develop a set of methodologies, associated tools and business models centered on SME Associations, which will enable software SMEs to effectively introduce Open Source Software Reuse practices in their production processes. In this scope, software reuse is regarded as the sharing of software modules across different development teams, organizations and diverse application domains.

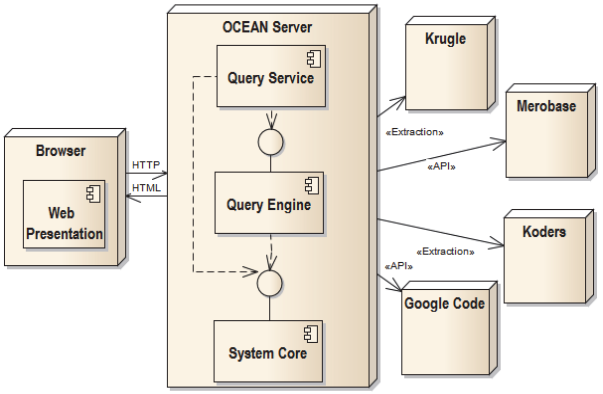

Figure 1. Deployment diagram of the federated code search engine.

III. IMPLEMENTATION DETAILS

In this section, we give implementation details of the Query Engine subsystem (Figure 1), which actually implements the federation. It supports two types of foreign search engine integration: API-based and Extraction-based.

A. API-based Integration

1) Merobase

Merobase [7] integration belongs to this case and was implemented by means of a JAR search client provided by the Merobase creators. A Perl web service was written utilizing this API and returning the results for a user-specified query in a suitable XML format. The Merobase API supports 2 parameters: *s* for the search keyword and *n* for the number of results requested. An upper limit of 30 results per query has been set by Merobase developers. An example http call the OCEAN sends to call this service is:

http://<*system*>/cgi-bin/merobase.pl?s=java&n=25

It is clear that, adding another API based search engine into the OCEAN's federation, is just a matter of pipelining its API with OCEAN's search form by means of a Perl script (as merobase.pl does in the example above).

2) Google Code Search

Google Code Search was integrated through its API. It turned out though that soon after the integration Google announced that the service will be no longer available. This is a nice example of the value of a federated search that continues to serve its users even though some sources are not available. Given the situation described, we do not give further details on this case.

B. *Extraction-based Integration*

When APIs are not available, web extraction does the integration. This requires the availability of an easy to use, robust and flexible web content extraction framework. DEiXTo, developed by our team independently of this work, was the tool of choice. It is briefly described right after.

1) *DEiXTo – A web content extraction framework*

DEiXTo [12] is a powerful web data extraction tool that is based on the W3C Document Object Model (DOM). It provides the user with an arsenal of features aiming at the construction of well-engineered extraction rules that describe what pieces of data to scrape from a website. DEiXTo consists of three separate components: a) *GUI DEiXTo*, implementing a friendly graphical user interface that is used to manage extraction rules. b) *The Command Line Executor* (CLE for short) massively applies wrapper project files built with GUI DEiXTo, on the desired web pages. CLE is actually a specialized instance of DEiXToBot. c) *DEiXToBot* is a Perl module aiming at tailor-made scraping and browser automation solutions. It facilitates the combination of multiple extraction rules as well as the post-processing of their results through custom code. Therefore, it can deal with complex cases and cover more advanced web scraping needs at the cost of the programming skills it requires.

Since there was no API access available for Koders [6] and Krugle [5], DEiXTo-based wrappers were successfully deployed in order to enable the extraction of the *N* first results returned from these search engines.

2) *Koders*

Koders [6] integration was smooth, in the sense that the html result pages were fully accessible by DEiXTo. The service supports 4 URL parameters: *s* for the search keyword, *li* for license type, *la* for language and *n* for the number of results requested.

3) *Krugle*

Krugle [5] integration on the other hand raised some difficulties mostly due to the heavy use of AJAX calls in its search results pages. Currently, DEiXTo does not support JavaScript automation. As a result we used Selenium [13] which actually automates a Firefox instance and were able to get Krugle's HTML results properly, and then forwarded them to DEiXTo for the actual extraction. Again, OCEAN sees Krugle as a web service supporting 4 URL parameters: *s* for the search keyword, *pro* for the aiming project, *lic* for the desired code license and *n* for the number of results desired.

IV. THE SYSTEM IN USE

The main screen of the search facility of OCEAN consists of the form depicted in Figure 2. In the textbox entitled "Search" the user can specify the keyword(s) of his search, separated by spaces. The search space can then be narrowed down by using the three combo boxes labeled language, license and type, respectively. *Language* refers to the programming language of the source code the user wishes to retrieve (e.g. Java, Perl, PHP, etc.). *License* refers to the type of the license under which the source code retrieved has been initially published. Finally, *type* refers to the type of the file the user is looking for. This type can be *class*, *interface* or *enum* (enumeration type). If any criteria do not apply to some search engine, they are simply omitted.

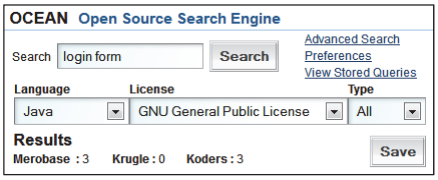

Figure 2. OCEAN's Search form in action.

OCEAN provides a set of preferences allowing the user to customize the service to her own needs. In preferences (Figure 3), the user can review his account details (if he is a subscribed user) and manage his account credentials and saved queries. He can also set the number of results per search engine OCEAN is going to return as well as the individual search engines he would like to include in his query.

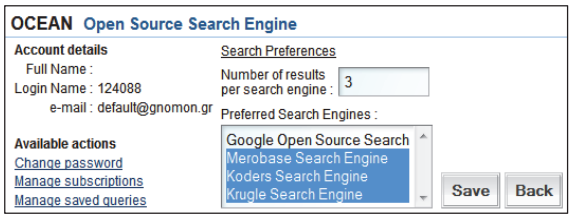

Figure 3. OCEAN's Preferences Screen.

In the query of Figure 2, the system returned six results, three of which were retrieved by Merobase and three other from Koders. For this specific query, Krugle did not return any results although it was engaged. The aforementioned summary of results is reported at the bottom of Figure 2. The detailed results (Figure 4) include the following information: the source search engine (Search Entry), the title of the source file returned and the URL of the repository to which it is hosted (Result Entry), lines of code (loc) for the source code file (Metrics), any possible metadata (Metadata) and finally the type of license under which the source code file was originally published. For search engines providing more detail, OCEAN can also provide more detail since we do extract all the data available. Currently, OCEAN does not perform any global ranking on the results. They are displayed in the order returned by their search engines.

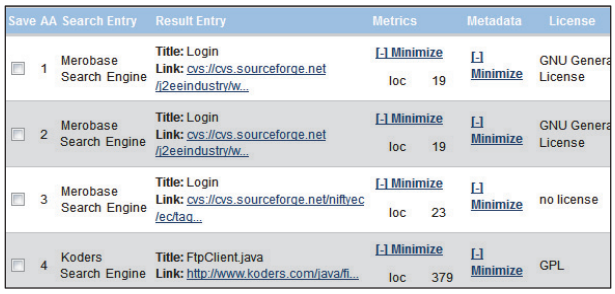

Figure 4. Detailed search results.

Table 1 presents some performance results of a few, informal, comparative runs. The aim was to get an idea of the

overhead introduced by the Query Engine of OCEAN. For the rest of the system, the overall performance depends on the performance of the slowest code search engine used (they used in parallel). OCEAN's performance in terms of speed and response time is quite satisfactory, at least to a large extent. The slowest search service is the one that queries Krugle. This is due to the fact that Selenium automates a Firefox instance and in its current implementation needs to get launched each time a query is posed to OCEAN. This introduces significant delay.

As far as Merobase and Koders are concerned, the required time for a search to complete is relatively low. Especially for Merobase, because of the API-based integration, the time needed was a bit smaller than the time the native Merobase web interface requires.

TABLE 1: DELAYS FOR ANSWER RECORDED IN VARIOUS QUERY SCENARIOS (ALL TIMES ARE IN SECONDS).

keyword: calendar **pref/nces**: "all languages" "all licenses"	
KODERS: 3.9	OCEAN w/ KODERS (10results): 9.6 OCEAN w/ KODERS (25 results): 10.0
KRUGLE: 7.0	OCEAN w/ KRUGLE (10 results): 32.6
MEROBASE: 7.8	OCEAN w/ MEROBASE (10 results): 5.7
OCEAN w/ KRUGLE+KODERS+MEROBASE (10 results): 36.6	

Finally, it should be noted that the Query Engine was run separately from the rest of the OCEAN. It was accessed through a large and constantly busy academic network that definitely contributes slightly in the delays recorded.

V. CONCLUSION & FUTURE WORK

We have presented OCEAN, a federated open source code search engine that is capable of integrating other web-based code search engines results accessible either by proprietary APIs or by web page extraction. The result has proven to be expandable, robust and modular and aims toward reducing the difficulty in searching and retrieving OSS components (as reported in Section 1). Our work is a first step towards comparing and contrasting the different offerings in an attempt to comprehend the current shortcomings.

Besides the unification of source code search engines, the integration of other information sources essential for reuse would also be interesting. In our survey mentioned in Sec. 1, we have found that the factors preventing OSS reuse are in order of importance the following: 1) Lack of documentation (80%), 2) Quality uncertainty (76%), 3) The difficulty in searching and retrieving OSS components (66%), 4) Licensing (59%) and 5) The 'Not invented here' syndrome (46%).

Currently, OCEAN addresses the searching and retrieval of OSS components. A future goal is to include other sources of information to enable the more effective reuse of the retrieved source code. For example to address the preventing factor (1), related documentation that could assist re-users in comprehending and reusing the source code could also be retrieved. We think of this as a combination of source code search engines and "traditional" search engines such as Google. For example the re-user could see along with the discovered source code a link to a tutorial related to the project of the code.

To address (2) we could further enhance OCEAN with the capability to discover automatically related unit tests for a discovered class for example, so that the re-user can start testing the retrieved source code to address the quality uncertainty issue. Unit tests could also provide a usage example of the code as well as a starting point in carrying out the integration of the code to the re-user's project.

Finally we could try to automatically analyze the reusability of the result sets' code, using software reusability metrics that can be found in literature. By doing so we will be investigating the reusability of code parts that although not autonomous, implement certain functionality.

ACKNOWLEDGMENT

This work is partially funded by the European Commission in the context of the OPEN-SME "Open-Source Software Reuse Services for SMEs" project, under the grant agreement no. FP7-SME-2008-2 / 243768.

REFERENCES

[1] T. R. Madanmohan and Rahul De', "Open source reuse in commercial firms", IEEE Software, vol. 21, 2004, pp. 62-69.

[2] G. Kakarontzas, P. Katsaros, I. Stamelos, "Component certification as a prerequisite for widespread OSS reuse", vol. 33, Electronic Communications of the EASST, 2010

[3] J. M. Gonzalez-Barahona, A. Martínez, A. Polo, J. J. Hierro, M. Reyes, J. Soriano, and R. Fernández, "The Networked Forge: New Environments for Libre Software Development", in "Open Source Development, Communities and Quality", B. Russo, E. Damiani, S. Hissam, B. Lundell, and G. Succi (eds.), IFIP Advances in Information and Communication Technology, vol. 275, Boston Springer, 2008, pp. 299–306.

[4] J. Howison and K. Crowston, "The perils and pitfalls of mining SourceForge", Proc. of the International Workshop on Mining Software Repositories (MSR 2004), 2004, pp. 7-11.

[5] Krugle official website: http://www.krugle.com/

[6] Koders official website: http://www.koders.com/

[7] Merobase official website: http://www.merobase.com/

[8] Free Softwarer Foundation official website: http://fsf.org/

[9] Open Source Initiative official website: http://opensource.org/

[10] OCEAN official website (beta): http://ocean.gnomon.com.gr/

[11] OPEN-SME project official website: http://opensme.eu/

[12] DEiXTo official website: http://deixto.com/

[13] Selenium official website: http://seleniumhq.org/

[14] A. Scherlen, (Balance Point column editor) J. Boyd, P. Pugh, M. Hampton, P. Morrison and F. Cervone, "The One-Box Challenge: Providing a Federated Search that Benefits the Research Process" Serials Review. 32 (4), 2006, pp. 247-254.

[15] Xiaotian Chen, "Metalib, WebFeat, and Google: The strengths and weaknesses of federated search engines compared with Google", Online Information Review, 30 (4), 2006, pp.413–427.

[16] A. Laender, B. Ribeiro-Neto, A.S. da Silva, and J. Teixeira, "A Brief Survey of Web Data Extraction Tools", ACM SIGMOD Record, 31 (2), 2002, pp. 84-93.