Integrating Linked Data Driven Software Development Interaction into an IDE

Aftab Iqbal, Oana Ureche, and Michael Hausenblas

DERI, National University of Ireland, Galway, Ireland
{firstname.lastname}@deri.org

Abstract. With "Linked Data Driven Software Development" (LD2SD) we have introduced a light-weight, linked data-based framework that allows to integrate software artefacts, such as version control systems and issue trackers, as well as discussion forums. The so created interlinked data-space enables uniform query and browsing facilities. In this paper we elaborate on the interaction part of LD2SD, and demonstrate how the LD2SD-interaction can be integrated into an Integrated Development Environment (IDE). We have performed an end-user evaluation and report on our findings and outline future steps.

Key words: linked data, interaction, Software Engineering, IDE

1 Introduction

In the software development process, both humans and so called *software artefacts* are involved (cf. Fig. 1 from [IUHT09]). Some of the software artefacts are directly under the control of the developers, while others are shared among users and developers, such as bug tracking system, documentation, discussion forums etc.

Developers use different mediums of communication to interact with each other and to solve problems. For example, Java source code and bugs are often discussed in forums or project mailing lists. However, the interconnections among software artefacts in the various data sources are typically not explicit. Developers nowadays have to perform keyword-based searches on the Web to find examples for source code or need to manually trace discussions about a bug on a blog. The attraction of using semantic technologies in order to address this issue is based on the idea to transform software artefacts into an conceptually organised and interlinked data-space, incorporating data from different software artefacts [DB08].

With "Linked Data Driven Software Development" (LD2SD) [IUHT09] we have introduced a linked data [BHBL09] based, light-weight framework that allows to integrate software artefacts. The so created interlinked data-space enables uniform query and browsing facilities. In this paper we elaborate on the interaction [Hea08] part of LD2SD, and demonstrate how the LD2SD-interaction can be integrated into an Integrated Development Environment (IDE).

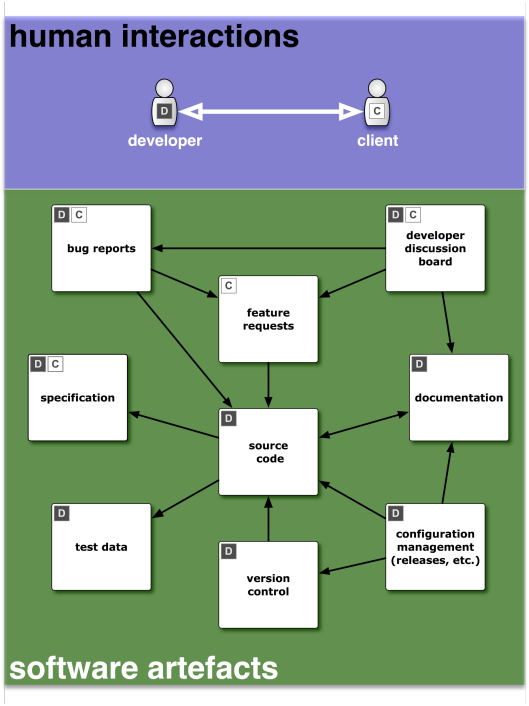

Fig. 1: The software development process, including its participants.

With the work at hand, we aim at enabling Java developers to explore related information about software artefacts. We present an interface that developers can use as a plug-in in their development environment (IDE), such as Eclipse[1] to find related information about Java source code, which might be found in a bug tracking systems, Subversion logs or in a blog discussion.

The paper is structured as follows: in Section 2, we discuss our LD2SD-based approach. We report on a concrete use case and an implementation of the LD2SD interaction in Section 3. The Section 4 presents the results from an end-user evaluation. In Section 5 we review related and existing work and eventually, in Section 6, we conclude our work and give an outlook on future work regarding LD2SD.

2 Linked Data Driven Software Development (LD2SD)

There are manifold ways to integrate different software artefacts. In order to provide a unified access to the different software artefacts, one needs to interlink and integrate these software artefacts, as we have argued in [IUHT09]. The overall concept of *Linked Data Driven Software Development* (LD2SD)—depicted in

[1] `http://www.eclipse.org/`

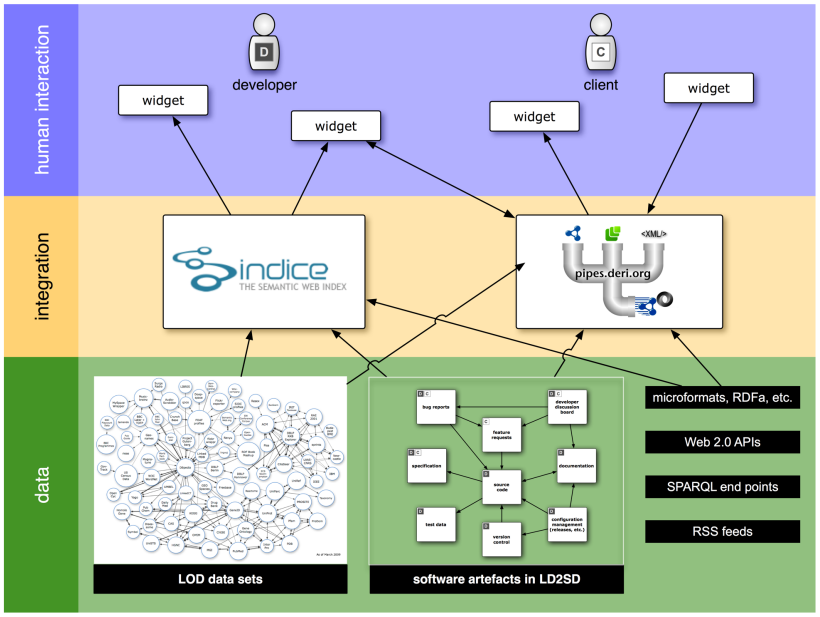

Fig. 2: The layered LD2SD approach for software artefacts integration.

Fig. 2 from [IUHT09]—is a layered, linked data based integration of software artefacts.

In this paper we elaborate on the top-most part of the LD2SD approach, the *interaction part*. We have chosen a particular setup, assuming a Java developer using Eclipse. The goal was to create an Eclipse plug-in that allows the developer to consume LD2SD data.

To enable a rapid development of our demonstrator, we decided to build upon an already available indexing service for the integration layer. The data layer, including RDFication and Interlinking was reused from [IUHT09].

In Fig. 3 our setup is described in detail:

1. RDFizing the software artefacts based on the linked data principles, yielding LD2SD datasets;
2. Using an indexing engine, *Apache Lucene/Solr*[2] to index the LD2SD datasets;
3. Develop a lookup service on top of the indexing service to enable keyword-based search over the LD2SD datasets;
4. Deliver the information to the developer via an Eclipse plug-in.

In order to achieve the linked data functionality for software artefacts, one needs to generate RDF data and interlink them. We have shown elsewhere [IUHT09] how to achieve the RDFizing and interlinking of the software artefacts.

[2] `http://lucene.apache.org/solr/`

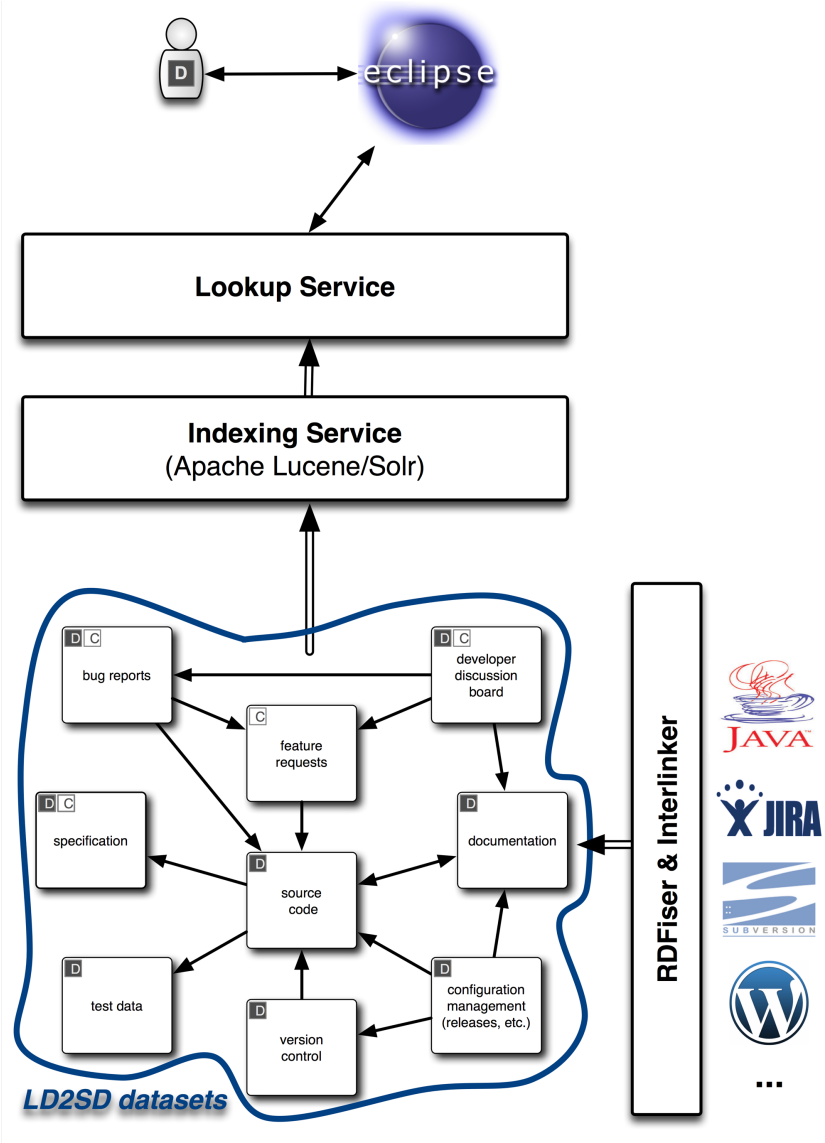

Fig. 3: LD2SD Interaction Setup.

After RDFizing and interlinking, the datasets are indexed by the *Apache Solr* indexing service. Each document is split into multiple documents based on the number of resources described in it and each resource is indexed as a separate document. The advantage of splitting an RDF document into sub-documents is that the lookup service returns the specific resource as a result rather than the whole document. For example, the Java2RDF [IUHT09] parser generates a single RDF dump of a software project, which contains description about the Java packages, the Java source in each package and the Java methods in each class along with JavaDoc.

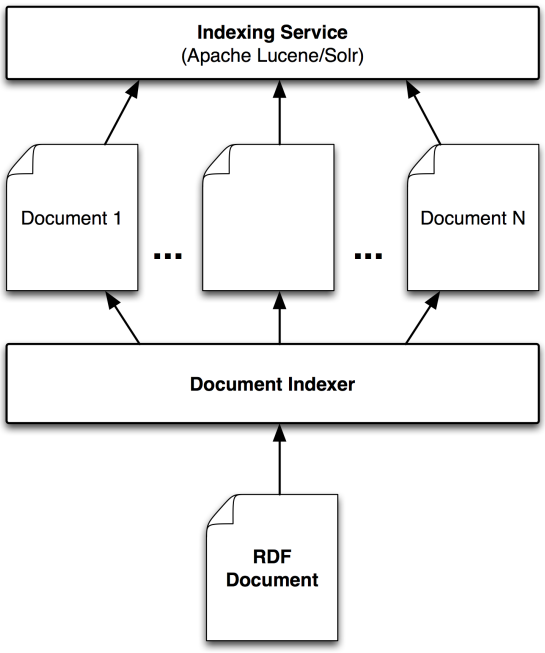

Fig. 4: Document indexing process.

The *Document Indexer* (cf. Fig. 4) indexes each package, Java classes and Java methods as a separate document. This approach allows the user to query for a Java class or a Java method and the *Apache Solr* indexing service will return the specific document instead of the RDF document for the entire software project. In order to query for the documents stored by the indexing service, the lookup service enables keyword-based queries against the indexing service. Although, different software artefacts are indexed by the indexing service, the *Document Indexer* does not only store content, but also the URI of document and the type of document, for example `JavaClass`, `Package`, `Discussion` etc.

3 Implementation

For the concrete implementation of the LD2SD plug-in, we have chosen Eclipse, a popular Java IDE. Software developers spend most of their time in IDEs such as Eclipse, though need to access bug tracking system to log bugs or discuss development issues in blogs or mailing lists, etc. As we wanted to offer a cross-platform, extensible solution, we decided to implement the actual interface of the plug-in as a linked data application [Hau09] based on HTML and utilising the Eclipse-internal Web browser. Alternatively, the LD2SD-interaction is also possible via a standalone Web browser.

3.1 Interaction via Eclipse plug-in

To enable developers to search for documents or to find related information about software artefacts without leaving their development environment, we have implemented an Eclipse plug-in. This enables the developer to retrieve related information about entities (such as classes, methods, etc.) in the Java source code of a software project. Say, the developer is interested in related information about a certain Java class. One way to trigger the LD2SD plug-in is to right click on the Java class and select the "Show Related Information" command (cf. bottom of Fig. 5) from the context menu.

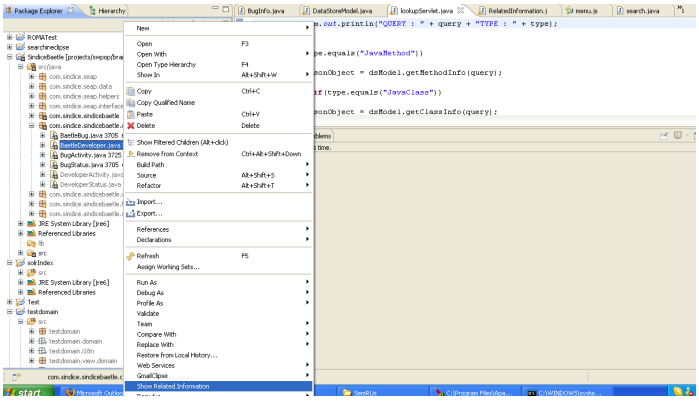

Fig. 5: Triggering the plug-in via the context menu.

In response, the lookup-service provides URIs as entry points and issues automatically a SPARQL query, which is executed on the entire LD2SD datasets to retrieve related information about that Java class as shown in Fig. 6.

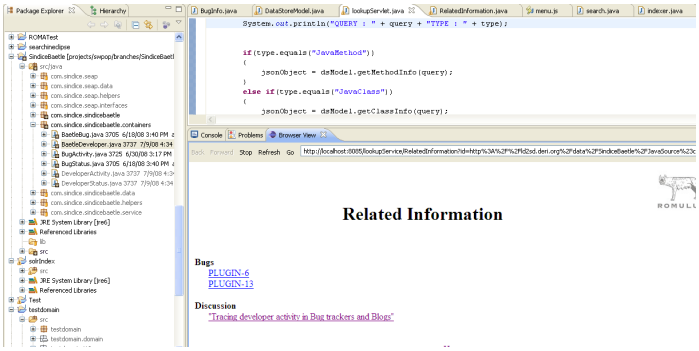

Fig. 6: Related information about a Java class in the Eclipse IDE.

3.2 Interaction via standalone Web browser

Alternatively to the Eclipse plug-in, the interaction is possible via a standalone Web browser. Developers can issue simple keyword-based queries and can navigate the search results concerning related information for that resource.

Let us assume the developer is interested about related information concerning a certain Java package. He can query the LD2SD data by using Web browser. Typically, the developer will enter, say, a package name and the lookup service returns relevant information about all Java classes belonging to that package (Fig. 7). Additionally, the developer can browser for related information about a Java class by clicking the "Related Information" link displayed next to it.

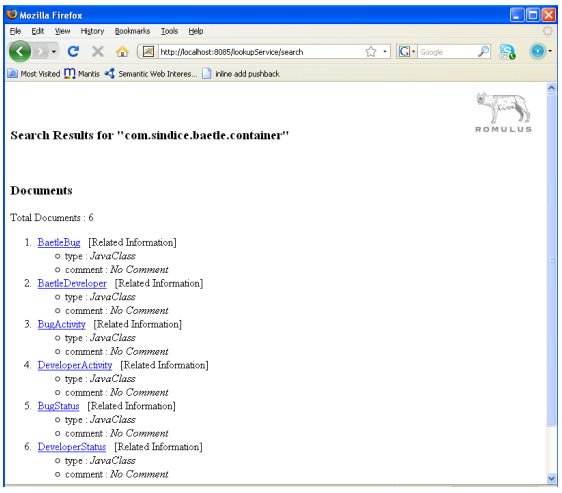

Fig. 7: Search results for a Java package in the Web interface.

4 Evaluation

To evaluate our approach, we have prepared data from a software project (bug tracking data, subversion logs, blogs and Java source code). We assessed the usability of our Eclipse plug-in approach via end-user evaluations requiring participants to perform a set of tasks. We measured the time it took them to complete the tasks and asked questions around the usability.

No.	Software Artefact	Familiarity (%)
1	Eclipse IDE	80
2	Bug tracking systems	60
3	Discussion blogs	100
4	Subversion plug-in for bug tracking system	42
5	Mylyn plug-in for Eclipse IDE	45

Table 1: Familiarity of the participants with the tools.

Twelve participants took part in the evaluation. The participants have development experience ranging from one to five years with different backgrounds. Table. 1 shows the familiarity of the participants with the different software artefacts.

The participants were asked to carry out a set of tasks on the test data:

Task 1 Identify all blog posts that mention a specific Java class.
Task 2 Identify all bugs that have been fixed by modifying a specific Java class.
Task 3 Identify all developers that are working on a Java package.
Task 4 Identify all blog posts that mention a specific Java package.
Task 5 Identify all bugs that belong to a specific Java package.

For the *Task 2* and *Task 5* we have installed a Subversion plug-in for, which shows the logs for bugs, if any. We conducted our evaluation in two phases:

Manual Approach In the first phase, we gave participants access to the software project, the bug tracking system, and the blog. The participants searched through blog posts, traversed bug reports and searched source files for authors to carry out each task.

Plug-in Approach In the second phase, we asked participants to carry out the tasks by using our LD2SD plug-in for the Eclipse IDE.

In the first phase, we found that participants used different heuristics to list the results of each task. After the second phase, we asked the participants to compare the results of both approaches. We found that some participants missed certain results using the first approach while carrying out the tasks. Further, participants apparently had difficulties going through each bug report to identify corresponding bug entries in Subversion. During the evaluation, we asked the participants to answer a set of *yes/no* questions as shown in Table. 2.

Question	Yes	No
Is the tool useful to discover related information?	12	0
Does our approach added value compared to the usual exploration of related information?	12	0
Is the design and layout of the tool suited enough for usage?	9	3
Does the integration of software artefacts as an Eclipse plug-in offer an advantage?	10	2

Table 2: User-study regarding tool.

The time for each task has been measured in both phases of the evaluation; the resulting graph (Fig. 8) is plotted based on the average time each participant spent in carrying out the task.

A big majority of the participants found our approach of extracting related information and presenting it in an integrated manner inside Eclipse interesting and useful. For example: *"... single point access within the Eclipse IDE seems a natural tool to use. It provides information much faster than accessing individual sources. The integrated view is very convenient"*.

Participants liked the LD2SD approach to interlink the Java sources with subversion logs and bugs using linked data principles. They were able to answer questions such as: (1) who has fix the certain bug and which source files he has modified in fixing the bug, (2) who should i talk to, and (3) which blog posts are talking about that certain bug; an exemplary comment highlights this: *"... it saves time for a developer and provides a unique interface to have a look at all relevant information in a single view ... all relevant information is available on a single click ... the idea looks promising, the tool would be more useful as it evolves and add features"*.

The evaluation helped us identifying the limitations of the tool as well. Some participants commented on the interface: *"... interface is relatively small, might be an usability issue for large amount of data ... would be interesting to see how*

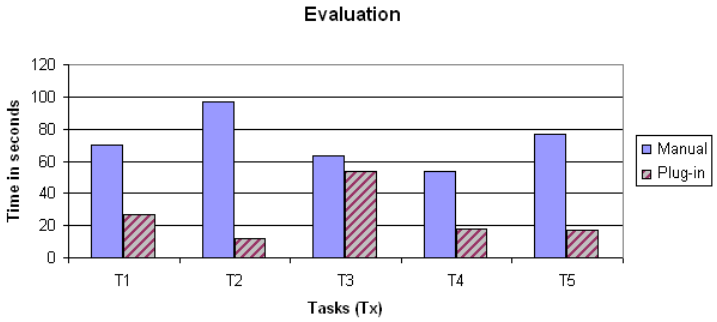

Fig. 8: Time required to perform tasks (manual vs. plug-in approach).

well it works with large amount of data–may be difficulties with presentation". Participants showed considerable interest in using our tool, especially in the case when they had to search information within hundreds of blog posts and bugs for a software project. (e.g., *"... I would definitely use the tool once it is available"*).

5 Related Work and Discussion

There are technologies available in the open source community that allow the integration of software artefacts. An interesting and closely related approach is Tesseract [SMWH09], a socio-technical dependency analyzer to enable exploration of cross-linked relationships between artefacts, developers, bugs and communication.

A related work concerning the combination of software artefacts has been described by Damljanovic et. al. [DB08]. The annotations are based on Key Concept Identification Tool(KCIT) which is capable of producing ontology-aware annotations. To interlink documents based on mentions of key concepts, the authors have used the PROTON KM ontology[3]. To enable semantic-based access through text-based queries, they used QuestIO (Question-based Interface to Ontologies), allowing to translate text-based queries into the relevant SeRQL queries, execute them and presents the results to the user. Their approach differs from our approach in that we have used *Apache Solr*, which provides high-performance engine for full-featured text search.

Another related approach has been described by Ankolekar et. al. [AS+06]. Dhruv is a Semantic Web enabled prototype to support problem-solving processes in web communities. The main focus of their research is how open source communities deals with bugs in their software under development. Their approach helps to connect the communication of developers (via forums and mailing lists) with bug reports and source code of the software. They have provided an ontology to model software, developers and bugs. The ontology is semiautomatically populated with data from different information sources to support bug resolution. Their approach assists the communication between developers for bug resolution. In contrast to their approach, we have provided an Eclipse plug-in which allows software developer or project manager to search for related information about a certain Java class or Java package on a single click without leaving their IDE.

In [AGS07], Antunes et. al. have presented SRS, a Semantic Reuse System designed to store and reuse knowledge for software development. The knowledge about software artefacts is represented using *Representation Ontology*, mapped with the concepts of *Domain Ontology* and stored in the *SDKE* (Software Development Knowledge Element) repository, which is managed using *Apache Lucene*[4]. Concepts are extracted from software artefacts using linguistics tools from *Natural Language Processing* (NLP) [JM02] prior to indexed by the *Apache Lucene*.

[3] `http://proton.semanticweb.org/2005/04/protonkm`
[4] `http://lucene.apache.org`

In [KBT07], Kiefer et. al. have presented EvoOnt[5], a software repository data exchange format based on OWL. EvoOnt includes software code, code repository and bug information. The authors have used the iSPARQL[6] engine which is an extension of SPARQL, to query for similar software entities. iSPARQL is based on *virtual triples* which are used to configure *similarity joins* [Coh00]. Their approach differs from our approach in that we have provided a methodology [IUHT09] to integrate the software artefacts by RDFizing and interlinking them.

Mylyn[7] is a sub-system for the Eclipse IDE allowing multitasking for developer and task management. It provides the means to render bug-related data in the Eclipse IDE for developers to work efficiently in their development environment without having to log in to the Web based application to update or create new bugs. Mylyn is limited to issue trackers such as JIRA[8] or Bugzilla[9], and hence not able to cope with the variety of software artefacts as we desire it.

Further, there are plug-ins[10] which integrate Subversion with bug trackers. The plug-in display all Subversion commit messages related to a specific issue.

To the best of our knowledge there is no tool available that is able to deal with software artefacts in a flexible and open way as we have provided it based on the LD2SD approach.

In [IUHT09] we have already demonstrated the methodology of RDFizing and interlinking heterogeneous software artefacts. In LD2SD we address the issue of heterogeneous software artefacts by using a common data model (RDF) along with global unique identifiers (URIs). The current version of LD2SD relies on the open access to the different data sources to extract metadata, interlink and query them. As pointed out in [IUHT09], a number of ontologies (BAETLE, FOAF, SIOC, iCalendar) have been used to deal with the domain semantics.

6 Conclusion and Future Work

We have motivated and described LD2SD, a light-weight, linked data-based framework allowing to integrate software artefacts, such as version control systems and issue trackers, as well as discussion forums. In this paper we have focused on the interaction layer, that is, providing means for developers to consume LD2SD-based data from existing software artefacts.

Based on a concrete use case and implementation of the LD2SD interaction part, an Eclipse plug-in, we have conducted an end-user evaluation. Our evaluation shows that the plug-in is relatively easy to use and valuable for developers.

Our future work will focus on the improvement of the current Eclipse plug-in. Currently, our keyword-based lookup service returns a list of all relevant

[5] `http://www.ifi.uzh.ch/ddis/evo/`
[6] `http://www.ifi.uzh.ch/ddis/isparql.html`
[7] `http://www.eclipse.org/mylyn/`
[8] `http://www.atlassian.com/software/jira/`
[9] `http://www.bugzilla.org/`
[10] `http://subversion.tigris.org/links.html#misc-utils`

documents, without ranking. Based on [GGM97] we will focus on the ranking of the results, taking into account the context of the developer. Motivated by the outcome of the evaluation, we want to provide plug-ins for other IDEs, for example for NetBeans.

We want to integrate even more software artefacts (such as mailing lists, Java test cases, developers profiles, etc.) and aim at increasing the interlinking quality and quantity.

Eventually, we plan to perform follow-up evaluations on a real world open source project with a broader target audience (real world developers, etc.).

Acknowledgements

Our work has partly been supported by the European Commission under Grant No. 217031, FP7/ICT-2007.1.2—"Domain Driven Design and Mashup Oriented Development based on Open Source Java Metaframework for Pragmatic, Reliable and Secure Web Development"(Romulus)[11].

References

[AGS07] B. Antunes, P. Gomes, and N. Seco. SRS: A Software Reuse System based on the Semantic Web. In *3rd International Workshop on Semantic Web Enabled Software Engineering (SWESE) of the 4th European Semantic Web Conference (ESWC 2007)*, Innsbruck, Austria, 2007.

[AS+06] A. Ankolekar, K. Sycara, , J. Herbsleb, R. Kraut, and C. Welty. Supporting online problem-solving communities with the Semantic Web. In *Proceedings of the 15th International Conference on World Wide Web*, Edinburgh, Scotland, 2006.

[BHBL09] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data—The Story So Far. *Special Issue on Linked Data, International Journal on Semantic Web and Information Systems (IJSWIS)*, page to appear, 2009.

[Coh00] W. W. Cohen. Data Integration Using Similarity Joins and a Word-Based Information Representation Language. In *ACM TOIS*, pages 288–321, 2000.

[DB08] D. Damljanovic and K. Bontcheva. Enhanced Semantic Access to Software Artefacts. In *5th International Workshop on Semantic Web Enabled Software Enginering (SWESE 2008)*, Karlsruhe, Germany, 2008.

[GGM97] L. Gravano and H. Garcia-Molina. Merging Ranks from Heterogeneous Internet Sources. In *VLDB97, Proceedings of 23rd International Conference on Very Large Data Bases*, pages 196–205, Athens, Greece, 1997.

[Hau09] M. Hausenblas. Linked Data Applications. First Community Draft, Linked Data Research Centre, 2009. `http://linkeddata.deri.ie/tr/2009-ld2webapp`.

[Hea08] T. Heath. How Will We Interact with the Web of Data? *IEEE Internet Computing*, 12(5):88–91, 2008.

[11] `http://www.ict-romulus.eu/`

[IUHT09] A. Iqbal, O. Ureche, M. Hausenblas, and G. Tummarello. LD2SD: Linked Data Driven Software Development. In *21st International Conference on Software Engineering and Knowledge Engineering (SEKE 09)*, Boston, USA, 2009.

[JM02] P. Jackson and I. Moulinier. *Natural Language Processing for Online Applications: Text Retrieval, Extraction and Categorisation.* John Benjamins Publishing Company, Amsterdam, Netherlands, Wolverhampton, United Kingdom, 2002.

[KBT07] C. Kiefer, A. Bernstein, and J. Tappolet. Mining Software Repositories with iSPARQL and a Software Evolution Ontology. In *Proceedings of the ICSE International Workshop on Mining Software Repositories (MSR)*, Minneapolis, MA, 2007.

[SMWH09] A. Sarma, L. Maccherone, P. Wagstrom, and J. Herbsleb. Tesseract: Interactive Visual Exploration of Socio-Technical Relationships in Software Development. In *International Conference on Software Engineering (ICSE 2009)*, Vancouver, Canada, 2009.