LD2SD: Linked Data Driven Software Development

Aftab Iqbal, Oana Ureche, Michael Hausenblas, Giovanni Tummarello
Digital Enterprise Research Institute (DERI),
National University of Ireland, Galway
IDA Business Park, Galway, Ireland
firstname.lastname@deri.org

ABSTRACT

In this paper we introduce *Linked Data Driven Software Development* (LD2SD), a light-weight Semantic Web methodology to turn software artefacts such as data from version control systems, bug tracking tools and source code into linked data. Once available as linked data, the related information from different sources is made explicit, allowing for a uniform query and integration. We show the application of LD2SD using a real-world software project as the reference dataset and discuss the added value of LD2SD compared to existing technologies.

1. MOTIVATION

In the software development process, both humans and so called *software artefacts* are involved (Fig. 1). Human beings such as developers and clients (customers, project managers, etc.) typically interact not only face-to-face or telephone, but also by means of discussion forums, emails, etc.. The software artefacts shown in the lower half of Fig. 1 can be understood as *heterogeneous, interconnected datasets*, conveying information about the software project and the humans involved.

It is worth mentioning that very often these interconnections are not explicit, hence machine-accessible but rather of an implicit nature (a mentioning of a certain Java class in a blog post, for example). Further, some of these datasets, such as the program source code or versioning data are mainly under the control of a *developer*, whilst other datasets are widely "filled" by *clients*. Then, there are datasets that are shared between developer and clients (e.g., a discussion board). In any case, the datasets are closely related and interdependent. A bug report, for example, may lead to a change in the program code and additionally the documentation needs to be updated. This may be reflected in the configuration management system. Further, a feature request may indirectly arise from a discussion on a discussion board, for example.

Nowadays, development takes place mainly in two environments, (i) the developers Integrated Development Environment (IDE), such as Eclipse[1], and (ii) the Web, such as for finding examples and documentation, discussions, etc. as a large. We need hence not only make the links between the software artefacts within a project explicit but also allow connecting to data on the Web. Having such an explicit representation of the connection between the datasets available we will be able to support certain scenarios often found in the software development process:

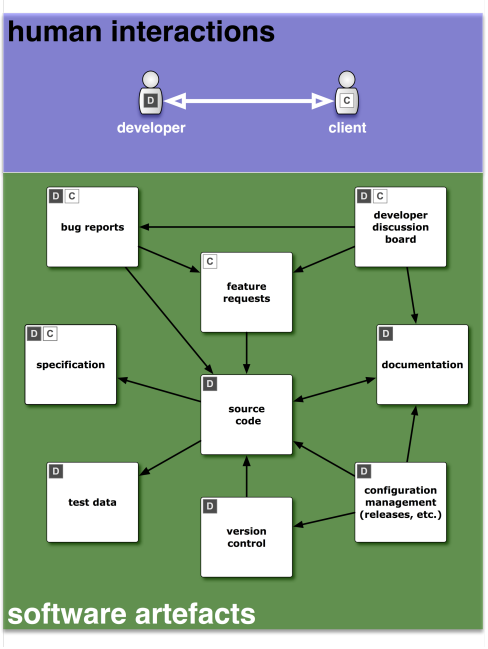

Figure 1: LD2SD Overview.

1. **Synthesis Scenarios**—support the development of new source code:
   - A developer could effectively query colleagues for support (expert finding) and/or being suggested contextualised code fragment(s);
   - A project manager could learn from previous projects and/or metrics.
2. **Analysis Scenarios**—support the project management and maintenance of existing source code:
   - One could perform opinion mining on SIOC [8] based representations of the discussion forums [19] in order to generate reports on a component or extract features requests and/or bug reports. SIOC (Semantically Interlinked Online Communities) is

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

a vocabulary to describe the content and structure of online community sites. It allows to create new connections between discussion channels and posts;

- Given that the documentation is interlinked with the source code, a dynamic FAQs could be provided;
- Developer profiles, based on their commitments to versioning control systems and the source code could be provided.

The contribution of this work is twofold: first, we introduce *Linked Data Driven Software Development* (LD2SD), a light-weight Semantic Web methodology to turn software artefacts such as data from version control systems, bug tracking tools or Java source code into linked data. Further, we show the application of LD2SD on a reference software project.

The paper is structured as follows: in section 2 we present use cases for LD2SD. Then, we introduce the overall LD2SD methodology in section 3 and discuss its characteristics in section 4. We report on exemplary implementation of LD2SD in section 5. In section 6 we review related work and compare it to our approach. Finally, we conclude in section 7 and outline future steps.

2. USE CASES

As motivated above, there are plenty of real-world scenarios one could think of where explicit interlinks between data sources would be desirable. We have detailed out a couple of use cases in the following which we realise in the realm of the software development. The use cases described below have in common that at least two data sets are involved. Note, that in case only *one* data set (such as bug tracking) is targeted, various solutions (cf. section 6) already exist, potentially not justifying the effort to apply LD2SD .

Finding an expert Jim has a long career in software project management. He knows that a task will be solved fast and bug-free only by a developer who is an expert in the required field. Jim now wants to assign a new task which involves Web pages scrapping. He needs to find a member of his team who is an expert in the HTML-Parser Java library.

Issues not fixed in due time Bug tracking systems contain a lot of *issue* entries. These *issues* need to be *fixed* on assigned dates. Harry, a project leader, is very busy, having to travel most of his time. He just came back from a project review and wants to know if all the issues due yesterday have been fixed. Harry additionally wants to know about the breakdown in terms of lines-of-code committed and which packages have been effected.

Find developer replacement Mary, a developer for a software company, has to relocate with her husband in another city. Julie, her supervisor, needs to hire a developer who can replace Mary. Therefore, Julie wants an analysis of her expertise and latest activities: assigned bugs, committed code, mailing list and blog posts, subsequently finding CVs that match Mary's expertise.

Assigning a bug to a developer Bug tracking environments structure *bugs* assignment by projects. John, a user of project X finds a bug and reports it on a *blog* post. Sarah, a developer of project Y, reads the *blog* post. However, she does not know the project X developers and their experience. She needs to find the most active developer in project X and assign the *bug* to him/her.

3. LD2SD FOUNDATIONS

We first introduce linked data, the foundation of *Linked Data Driven Software Development* (LD2SD), and then give an account of the LDSD methodology.

3.1 Linked Data

The basic idea of linked data [6] has first been outlined by Tim Berners-Lee in 2006[2], where he described the linked data principles as follows: (i) all items should be identified using *URIs* [18] and these URIs should be *dereferenceable*, that is, using HTTP URIs allows looking up the an item identified through the URI, further (ii) when looking up an URI (an RDF [13] property is interpreted as a hyperlink), it leads to more data, and (iii) links to URIs in other datasets should be included in order to enable the discovery of more data. In contrast to the full-fledged Semantic Web vision, linked data is mainly about publishing structured data in RDF using URIs rather than focusing on the ontological level or inferencing. This simplification—comparable to what the World Wide Web did for hypertext—fosters a wide-spread adoption [4].

3.2 Methodology

In order to provide a *uniform* and *central* access to the different datasets, one needs to interlink, integrate and align them. Various techniques could potentially be utilised (see also section 6), however, given the arguments regarding linked data above, we decided to realise a linked-data driven approach. In Fig. 2 the overall LD2SD methodology is depicted. This methodology basically covers the layers as described in the following:

1. Assign URIs to all entities in software artefacts and convert to RDF representations based on the linked data principles, yielding LD2SD datasets;
2. Use semantic indexer, such as Sindice [15] to index the LD2SD datasets;
3. Use semantic pipes, such as the DERI Pipes (cf. section 5.2) allowing to integrate, align and filter the LD2SD datasets;
4. Deliver the information to end-users integrated in their preferred environment, such as discussed in section 5.3.

4. LD2SD CHARACTERISTICS

4.1 Scale to the Web

In 2007, the Linking Open Data (LOD) project[3], an open, collaborative effort aiming at bootstrapping the Web of Data

[2] `http://www.w3.org/DesignIssues/LinkedData.html`
[3] `http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData`

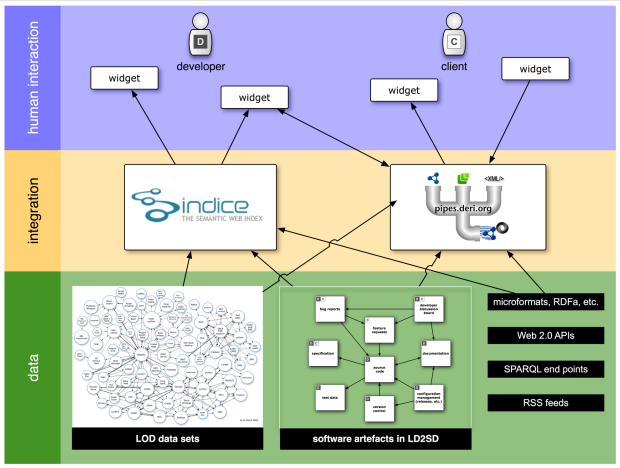

Figure 2: LD2SD Methodology.

by publishing datasets in RDF on the Web and creating interlinks between these datasets, has been launched. With over 50 interlinked dataset offering billions of RDF triples and millions of interlinks, the so called "LOD cloud" enables entire new application areas [11].

We highlight the fact, that by utilising LD2SD, a Web-scale data integration of software development-related information is hence made possible. One can imagine that—as both LD2SD and LOD follow the linked data principles—we are able to connect software artefacts to the LOD datasets, such as DBpedia [3], hence enabling the reuse of existing information in the software development process.

4.2 Read-only?

To this end, LD2SD allows us to integrate, view and filter the data. However, one problem remains unresolved: updating the original software artefact. With a recently launched community project called pushback[4]—aiming at turning the current "read-only" Semantic Web into a read/write Semantic Web—we are confident to adequately address this issue in the near future.

5. IMPLEMENTATION

The methodology described in the previous section will be demonstrated herein. We have divided this section according to Fig. 2: in section 5.1, we describe the reference data set and the interlinking, in section 5.2, we show the querying of the interconnected datasets using DERI Pipes[5], and in section 5.3 we present some earlier work on Semantic Widgets.

5.1 Data Layer

We first present a list of candidate software artefacts to be converted to RDF and then we present a concrete example to realise some use case described in section 2.

To practice what we preach, we have chosen the Sindice software project[6] as the reference software project. In Table 1 the details regarding the respective LD2SD datasets are listed, yielding more than 43,000 (43k) RDF triples in total.

In order to apply the interlinking approach to our software artefacts as listed in the previous section we examine the RDF datasets in the following. An excerpt of an exemplary RDF representation of some Sindice Java source code is shown in listing 1. Further, an example of some

```
@prefix b: <http://baetle.googlecode.com/svn/ns/#> .
@prefix : <urn:java:org.sindice.projects.wp.> .
:WPLinkExtractor a b:Class;
 b:contained <> ;
 b:uses <urn:java:java.awt.Component> ,
        <urn:java:java.io.IOException> .
```

Listing 1: An exemplary Java RDFication.

RDFised Subversion logs is shown in listing 2. From the

```
@prefix b: <http://baetle.googlecode.com/svn/ns/#> .
@prefix : <svn://sindice.com/svn/> .
:bc275 a b:Committing ;
 b:added <svn://sindice/wp/WPLinkExtractor.java> .
 b:author :oanure .
```

Listing 2: An exemplary Subversion RDFication.

listings 1 and 2, we are able to conclude that both RDF fragments are describing the same entity, "WPLinkExtractor.java". We can interlink[7] these two RDF fragments as shown in listing 3 using an `owl:sameAs` property indicating that these URIs actually refer to the same entity.

```
:WPLinkExtractor owl:sameAs
<svn://sindice/wp/WPLinkExtractor.java> .
```

Listing 3: An Interlinking Example.

5.2 Integration Layer

After RDFising and interlinking the software artefacts, the next step is integrating the artefacts and query them.

DERI Pipes [16] are an open source project used to build RDF-based mashups. They allow to fetch RDF documents from different sources (referenced via URIs), merge them and operate on them. In our case at hand, this involves four major steps:

1. Fetch the RDF representation of the Subversion log, JIRA[8] issue tracker, Java source code, etc. using the *RDF Fetch operator*[9];
2. Merge the datasets using a *Simple Mix operator*[10];
3. Query the resulting, integrated dataset with SPARQL[11];
4. Apply XQuery[12] in order to sort and format the data from the previous step.

[4] http://esw.w3.org/topic/PushBackDataToLegacySources
[5] http://pipes.deri.org/
[6] http://sindice.com/
[7] http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/#RDFlinks/
[8] http://www.atlassian.com/software/jira/
[9] http://pipes.deri.org:8080/pipes/doc/#FETCH
[10] http://pipes.deri.org:8080/pipes/doc/#MIX
[11] http://www.w3.org/TR/rdf-sparql-query/
[12] http://www.w3.org/TR/xquery/

Software Artefact	Data Format	RDFizer	Vocabulary	Triples
JIRA Bug Tracker	relational data	D2RQ [7]	BAETLE[a]	12k
Java Source Code	structured data	SIMILE RDFizer[b]	SIMILE Java2RDF	22k
Subversion	relational data	BAETLE RDFizer[c]	BAETLE	7k
Developer's calendar	RFC2445 [10]	iCalendar to RDF[d]	iCalendar [e]	1k
Developer's profile	FOAF/RDF	system specific	FOAF[f]	1k
Developer blog	relational data	SIOC exporter[g]	SIOC [8][h]	not yet implemented
Project Mailing Lists	RFC2822 [17]	SIMILE RDFizer[i]	SIMILE Email2RDF	not yet implemented

Table 1: The Sindice Reference Software Project.

[a] http://baetle.googlecode.com/svn/ns/
[b] http://simile.mit.edu/repository/RDFizers/java2rdf/
[c] http://code.google.com/p/baetle/
[d] http://www.kanzaki.com/courier/ical2rdf
[e] http://www.w3.org/TR/rdfcal/
[f] http://xmlns.com/foaf/spec/
[g] http://sioc-project.org/wordpress/
[h] http://rdfs.org/sioc/spec/
[i] http://simile.mit.edu/repository/RDFizers/email2rdf/

The output of the implemented pipe is then accessible via an URI.

Let's consider the situation described in the *Issues not fixed in due time* use case (cf. section 2). The information we need to process is contained in a JIRA RDF dump[13] describing the issues assigned to a developer, and in the the developers FOAF[14] files. The state of an issue can be *Open*, *Closed* or *Resolved*. We are interested in issues that are *Open* and that were due yesterday. Further, we want to display the issue summary and the author full name. In order to retrieve the information we are interested in, we apply a SPARQL SELECT query (listing 4) .

```
1 PREFIX b: <http://baetle.googlecode.com/svn/ns/#> .
2 PREFIX w3s: <http://www.w3.org/2005/01/wf/flow#> .
3 SELECT ?issue ?author ?summary ?due_date
4 WHERE {
5 ?issue b:assigned_to ?author ;
6        b:due_date ?due_date ;
7        b:summary ?summary ;
8        w3s:state
              <http://ld2sd.deri.org/data/Bugs2RDF/Open>
              .
9 }
```

Listing 4: SPARQL Query to Select Overdue Issues.

As a matter of fact, SPARQL is currently limited to filter only specific dates. However, XQuery allows some basic calculations on top of the resulting SPARQL XML file, as shown in listing 5. In the XQuery box (see Fig. 4) we can specify the `Content-type:xml/html`, allowing us to format the output using HTML and directly display the result in a Web browser. The code snippet from listing 5 calculates the yesterday's date by subtracting one day from the current date (lines 4-5) and renders the *summary*, *author* and *issue* elements as rows in an HTML table (Fig. 3). In a second step, the developer's profiles exposed as FOAF can be integrated. This would for example mean that the URIs in the *Author*-column in Fig. 3 would be replaced by the

[13] http://ld2sd.deri.org/data/Bugs2RDF/RDFDump.rdf
[14] http://www.foaf-project.org/

```
1  for $b in .//*:result
2    where xs:date(xs:dateTime($b/*:binding[@name =
3       "due_date"]/*:literal))
4       = xs:date(xs:date(current-date())-
5       xs:dayTimeDuration("P1DT0H0M"))
6    return
7       <tr style="background: #CBE9C7; color:
            black;">
8          <td> { $b/*:binding[@name =
               "summary"]/*:literal } </td>
9          <td> { $b/*:binding[@name =
               "author"]/*:uri } </td>
10         <td> { $b/*:binding[@name =
               "issue"]/*:uri } </td>
11      </tr>
```

Listing 5: XQuery Filtering by Yesterday's Date.

Summary	Author	Issue URI
Add documentation generation to Maven build	http://swm.deri.org:8080/LD2SD/SVN/gareth	http://swm.deri.org:8080/browse/SINFRA-24
Delete documents based on rules	http://swm.deri.org:8080/LD2SD/SVN/michael	http://swm.deri.org:8080/browse/SMDL-28
IndexReader does not count correctly the number of items	http://swm.deri.org:8080/LD2SD/SVN/andrea	http://swm.deri.org:8080/browse/SMDL-1

Figure 3: Pipe Result in a Web Browser.

respective developer's full name. Further, by integrating the developer's profile data, one can be group developer by team-membership (for example "core", "API", etc.) or render dependencies on other developers.

In the same manner the data from Subversion can be integrated in a further step in order to enable the breakdown in terms of lines-of-code committed or the highlight which packages have been effected by a certain bug-fix. Concluding, the more data sets are integrated, the richer the queries may be.

A screen-shot of a pipe implementing the above example is depicted in Fig. 4.

5.3 Interaction Layer

The interaction layer handles the interaction between the integrated data as described above and the end-users, such as developers. We have shown elsewhere [21] how to utilise

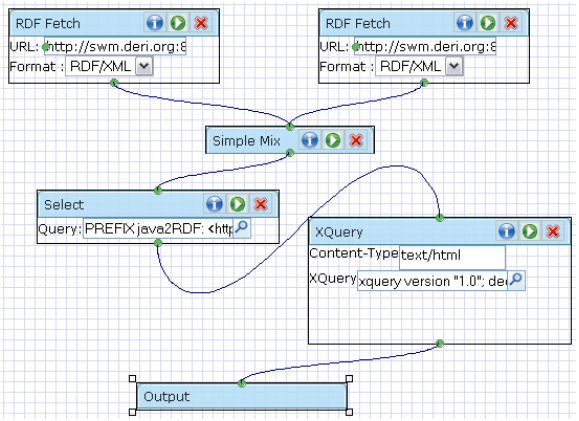

Figure 4: DERI Pipes.

the datasets using Semantic Widgets. With Semantic Widgets, we provide a methodology to enhance existing Web applications and deliver aggregated views of information to end-users. These views are accessed by clicking buttons

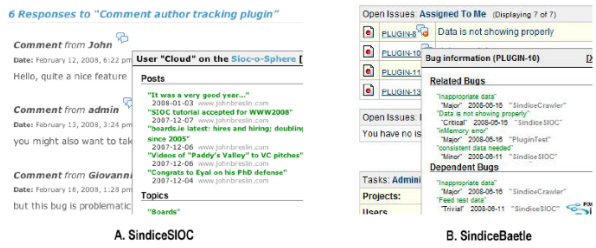

Figure 5: Examples of Semantic Widgets.

which are injected into the DOM of a Web page. For example, next to a bug, related information regarding bugs or dependent bugs is displayed as shown in Fig. 5.

6. RELATED WORK

There are certain technologies in the open source community available to combine software artefacts. Existing work related to combining software artefacts has been described in Dhruv by Ankolekar et.al. [1], a Semantic Web system for open source software (OSS) communities. It provides a semantic interface allowing users to see a detailed description of highlighted terms in the message posted during bug resolution in cross-links pages. Their approach extracts information about software artefacts using information extraction techniques based on noun phrases, code terms and references to other artefacts. Dhruv is specifically designed for OSS bug resolution processes.

Another interesting, closely related work has been described in [2]. There, a relational database is used to store information related to version control and bug tracking data. The source code meta model has been represented using the Rigi Standard Format (RSF) [20], which is a triple based specification language that can be easily customised [2]. The integration of these three artefacts has been done by (i) querying the relational database, and (ii) merging the result with the source model RSF files. In contrast to their approach, we have provided a methodology to integrate the candidate software artefacts by RDFizing and interlinking them using linked-data driven approach.

In [12], Kiefer et.al. have presented EvoOnt [15], a software repository data exchange format based on OWL [16]. EvoOnt includes software code, code repository and bug information. They have used the iSPARQL [17] 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* [9]. Their approach differs from our approach in that we have used DERI pipes to integrate and query different software artefacts.

Existing work related to our use cases discussed in section 2 has been described in [14]. Their approach uses data from change management systems and heuristics are based on but are limited to only two software artefacts, i.e., for software bugs and source code commits. Contrary to their approach, we have added developer's information from blogs, mailing lists and developer's profile to realize our use cases.

In [5], Basili et.al. have presented Experience Factory concept for software development. Experience Factory supports the evolution of processes and other forms of knowledge, based on experiences within the organization [5]. Experiences are captured from FAQs, chat logs, emails and project presentations. In contrast to their approach, we have provided a methodology to capture knowledge from candidate software artefacts and interlink them to find a certain expertise.

Mylyn[18] is a sub-system for the Eclipse IDE allowing multitasking for developer and task management. It provides the means 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. The limitation of Mylyn is that it works only with task repositories such as JIRA, Bugzilla[19].

Further, there are plug-ins[20] which integrates Subversion with bug trackers, for example Bugzilla or JIRA. The plug-in displays all Subversion commit messages related to a specific issue. To the best of our knowledge such Subversion plug-ins are available for a few bug trackers.

Existing work described above somehow try to address the integration or interlinking of different software artefacts but some of them are desktop applications and some are Web based applications and none of the above described approaches address all the candidate software artefacts we described in this paper (see Table 1). Still what is missing is the existence of a generic framework where all software artefacts can be collected and queried that would allow project managers to get an overall view of the software development projects.

[15] http://www.ifi.uzh.ch/ddis/evo/
[16] http://www.w3.org/TR/2004/REC-owl-guide-20040210/
[17] http://www.ifi.uzh.ch/ddis/isparql.html
[18] http://www.eclipse.org/mylyn/
[19] http://www.bugzilla.org/
[20] http://subversion.tigris.org/links.html#misc-utils

7. CONCLUSION & OUTLOOK

We have motivated and introduced Linked Data Driven Software Development (LD2SD) as well as demonstrated its value in a concrete setup in this paper. The basic idea underlying LD2SD is to make the **implicit links between software artefacts** found in software development—such as version control systems, issue trackers, discussion forums, etc.—**explicit** and expose them using RDF. By using LD2SD, one enables a Web-scale integration of data, connecting to the LOD cloud and enabling the vast reuse of information.

We plan to implement further LD2SD use cases to show how one can benefit from it, especially when more than two data sources are involved. We aim to overcome a shortcoming of the current implementation, as it is not 100% linked data conforming; in certain places we use URNs rather than HTTP URIs. Additionally we will improve the interlinking, yielding higher-quality and also more links between the LD2SD datasets.

Acknowledgements

Our work has partly been supported by the European Commission under Grant No. 217031, FP7/ICT-2007.1.2, project Romulus—"Domain Driven Design and Mashup Oriented Development based on Open Source Java Metaframework for Pragmatic, Reliable and Secure Web Development"[21].

8. REFERENCES

[1] 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.

[2] G. Antonio, M. D. Penta, H. Gall, and M. Pinzger. Towards the Integration of Versioning Systems, Bug Reports and Source Code Meta-Models. In *Electronic Notes in Theoretical Computer Science*, pages 87–99, 2005.

[3] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. DBpedia: A Nucleus for a Web of Open Data. In *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007*, pages 722–735, 2007.

[4] D. Ayers. Evolving the Link. *IEEE Internet Computing*, 11(3):94–96, 2007.

[5] V. R. Basili, M. Lindvall, and P. Costa. Implementing the Experience Factory concepts as a set of Experience Bases. In *Proceeedings of the 13th International Conference on Software Engineering and Knowledge Engineering*, pages 102–109, 2001.

[6] C. Bizer, T. Heath, K. Idehen, and T. Berners-Lee. Linked Data on the Web (LDOW2008). In *Linked Data on the Web Workshop (WWW2008)*, 2008.

[7] C. Bizer and A. Seaborne. D2RQ - Treating Non-RDF Databases as Virtual RDF Graphs. In *3rd International Semantic Web Conference*, Hiroshima, Japan, 2004.

[8] U. Bojars, J. Breslin, V. Peristeras, G. Tummarello, and S. Decker. Interlinking the Social Web with Semantics. In *IEEE Intelligent Systems, 23(3): 29-40,*, 2008.

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

[10] F. Dawson and D. Stenerson. Internet Calendaring and Scheduling Core Object Specification (iCalendar), RFC2445. IETF Network Working Group, 1998. `http://www.ietf.org/rfc/rfc2445.txt`.

[11] M. Hausenblas. Exploiting Linked Data For Building Web Applications. *IEEE Internet Computing*, N(N):to appear, 2009.

[12] 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.

[13] G. Klyne, J. J. Carroll, and B. McBride. Resource Description Framework (RDF): Concepts and Abstract Syntax). W3C Recommendation 10 February 2004, RDF Core Working Group, 2004.

[14] A. Mockus and J. Herbsleb. Expertise browser: a quantitative approach to identifying expertise. In *Proceedings of the 24th International Conference on Software Engineering, Orlando*, 2002.

[15] E. Oren, R. Delbru, M. Catasta, R. Cyganiak, H. Stenzhorn, and G. Tummarello. Sindice.com: a document-oriented lookup index for open linked data. *International Journal of Metadata, Semantics and Ontologies*, 3(1):37–52, 2008.

[16] D. L. Phouc, A. Polleres, C. Morbidoni, M. Hauswirth, and G. Tummarello. Rapid Prototyping of Semantic Mash-Ups through Semantic Web Pipes. In *Proceeedings of the 18th International World Wide Web Conference (WWW2009), ACM*, Madrid, Spain, 2009.

[17] P. Resnick. Internet Message Format, RFC2822. IETF Network Working Group, 2001. `http://www.ietf.org/rfc/rfc2822.txt`.

[18] L. Sauermann and R. Cyganiak. Cool URIs for the Semantic Web. W3C Interest Group Note 31 March 2008, W3C Semantic Web Education and Outreach Interest Group, 2008.

[19] S. Softic and M. Hausenblas. Towards Opinion Mining Through Tracing Discussions on the Web. In *Social Data on the Web (SDoW 2008) Workshop at the 7^{th} International Semantic Web Conference*, Karlsruhe, Germany, 2008.

[20] S. R. Tilley, K. Wong, M. A. D. Storey, and H. A. Muller. Programmable Reverse Engineering. In *International Journal of Software Engineering and Knowledge Engineering.*, pages 501–520, 1994.

[21] A. Westerski, A. Iqbal, G. Tummarello, and S. Decker. Sindice Widgets: Lightweight embedding of Semantic Web capabilities into existing user applications. In *Proceedings of the 4th International Workshop on Semantic Web Enabled Software Engineering, ISWC08*, 2008.

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