Fostering Synergies – How Semantic Web Technology could influence Software Repositories

Michael Würsch
wuersch@ifi.uzh.ch

Gerald Reif
reif@ifi.uzh.ch

Serge Demeyer[*]
demeyer@ifi.uzh.ch

Harald C. Gall
gall@ifi.uzh.ch

Department of Informatics, University of Zurich
Zurich, Switzerland

ABSTRACT

The state-of-the-art in mining software repositories stores software artifacts from various sources into monolithic relational databases. This puts a lot of querying power in the hands of the software miners, however it comes at the cost of enclosing the data and hamper cross-application reuse. In this paper we discuss four problem scenarios to illustrate that Semantic Web technology is able to overcome these limitations. However, it requires that the software engineering research community agrees on two prerequisites: (a) a common vocabulary to talk about software repositories – an ontology; (b) a strategy for generating unique and stable references to all software artifacts inside such a repository – a Universal Resource Identifier (URI).

1. INTRODUCTION

Over the last decade, the software engineering community developed various tools which help engineers to specify, develop, test, analyze, and maintain software. Most of these tools use proprietary data formats to store their artifacts. This hampers tool-interoperability and renders querying difficult, especially when you want to query across tool domains. Queries such as "In which release was this bug fixed and which source code modifications where necessary to fix it?", however, involve several domains (*i.e.*, static source code, version control, issue tracking).

The mining software engineering community has tackled this issue by mirroring software artifacts from various sources in a central (relational) database [6]. This additional querying power gave rise to numerous experiments where researchers successfully mined such databases for interesting patterns (see [11] for an overview; specific examples can be found in [3,8,9,16]). Unfortunately, such a central database imposes a universal data schema onto all contributing tools, turning the software repository into a rigid and inflexible monolith. Especially when integrating tools supplied by different research groups, such a software repository is nothing more than the kind of stovepipe systems we all resent.

[*]Prof. Demeyer is also affiliated with the Lab On Reengineering – University of Antwerp

Semantic Web technology has been designed as a solution to such integration problems. In a nutshell, it provides a standardized, well-established framework that allows data to be shared and reused across application, enterprise, and community boundaries. It does so by means of two concepts: (a) ontologies and (b) Universal Resource Identifiers (URIs). The former provides the formal vocabulary that applications can use to exchange semantically rich data, by defining the entities in the domain of discourse and the relationship between them. The latter is a unique and stable reference to all possible entities which enable hyperlinks between semantically-annotated data in one place with data in other places.

In this paper we argue that the use of Semantic Web technology enables the construction of highly interlinked and distributed knowledge bases, which form the basis for flexible queries and data analysis. Using four scenarios, we demonstrate several problems with the current state of the art, *i.e.*, centralized databases. For each of the scenarios we also show how Semantic Web technology may come to the rescue. We conclude the paper with a research agenda which lists the prerequisites that need to be resolved before these scenarios can be realized.

2. THE SEMANTIC WEB IN A NUTSHELL

The Semantic Web was designed to be an extension of the Web as we know it today, enriching it with meta data describing the semantics of Web pages to make their content computer-processable. To describe information on Web pages with meta data accordingly, an *ontology* has to be defined that formally describes the concepts (classes) found in the domain of discourse, the relations between these concepts and the properties used to describe them [10]. These principles are not restricted to Web pages but can be applied to any kind of data. In the software engineering domain, for example, we can define concepts, such as *User*, *Developer*, *Bug*, *Module*; relationships, such as *reports bug*, *fixes bug*, and *is assigned to bug*. Since the Semantic Web describes this information based on formal semantics, data can be exchanged among two applications that support the same ontology, even if they were not meant to interoperate in the first place.

The Resource Description Framework (RDF) [13] is the data-model for representing meta data in the Semantic Web. The RDF data-model formalizes meta data based on *subject – predicate – object* triples, so called RDF statements. RDF triples are used to make a statement about a resource in the universe of discourse. A resource can be almost anything: a bug report, a person, a Web page, a CD, a track on a CD, etc. Every resource in RDF is identified by a Uniform Resource Identifier (URI) [2].

In an RDF statement the subject is the thing (the resource) we want to make a statement about. The predicate defines the kind of information we want to express about the subject. The object defines the value of the predicate. The data-model of RDF is a graph where the subject and object are the nodes and the predicate is a labeled, directed arcs pointing from the subject to the object. The query language SPARQL [15] can be used to query such RDF graphs.

3. SCENARIOS

In the following, we list typical problem scenarios in the context of software analysis and software repository infrastructure. For each problem, we identify the key challenges it brings and elaborate on traditional solutions and their shortcomings, before we outline how the Semantic Web could resolve the issue. In Section 4, we set up a research agenda our community needs to work on.

Scenario 1: A Shared Vocabulary

Description: *Alice has just started a Ph.D. having the working title "On the Influence of the Programming Language on the Occurrence of Bugs in Open Source." In particular, she is curious about whether certain language features are misleading developers to introduce defects. After a thorough discussion with her promoter, she realizes that she needs to build up a query-able knowledge base containing a substantial amount structural information about source code of various open source project, as well as related bug reports. She plans to investigate projects written in Java and C# first, as the languages share many similarities. She reasons about a suitable meta model to represent her data and whether there are already existing parsers that she could adapt, so that she would not need to start from scratch.*

Key Challenges: One of the critical design aspects when building a knowledge base is to define a meta model that describes the data in an adequate level of detail. To share data among different tools and knowledge bases, they need to understand the same vocabulary.

Traditional Approaches: In practice, there are a number of meta models, for example for source code, that define the same concepts, but name them differently. For example, *C++ Data Model* [5] of Chen and the *FAMOOS Information Exchange Model (FAMIX)* of Tichelaar *et al.* [17] can both be used to describe source code written in C++. Although they share many commonalities, tools written to work on FAMIX can not process instances of Chen's model and vice versa. Further, the meta models are often implemented in terms of a relational database schemas. Exchanging schemas among different databases, however, is relatively uncommon, due to vendor-specific implementations of data definition languages. Instead, and despite the advent of specialized exchange formats, such as XMI [14] or GXL [18], data is often serialized into plain XML or a comma separated value (csv) format. These formats are not semantics-preserving and therefore of limited use.

The Semantic Web Approach: The Semantic Web provides a framework to structure data so that it becomes machine processable. The Web ontology language OWL is used to model the vocabulary of a domain [7] by describing properties and classes; for example, relations between classes (*e.g.*, inheritance relationships, disjointness), cardinalities, equality, or characteristics of properties (*e.g.*, inverse, symmetry). These vocabularies are called ontologies and have explicit formal semantics. While it is uncommon to exchange relational database schemas, ontologies were explicitly designed to be shared. They can be serialized using the RDF/XML standard and exchanged without the loss of data semantics.

Scenario 2: Linked Data

Description: *When Alice presents her first workshop paper, she rans into Bruce who is a Ph.D. student too. Bruce is focussing on bug prediction and has also put online a knowledge base, already containing hundreds of systems written in Smalltalk. Alice realizes that the work of Bruce is likely to provide complementary insights on her own research and convinces her promotor to set-up a one week research visit to investigate whether it would be possible to merge the two databases.*

Key Challenges: To interlink repositories on the Web, a flexible data model is needed that allows to expose references to data stored within a repository to the outside. It further should provide means to describe relationships between entities stored in different locations. These references need to remain stable over time to guarantee data consistency across the repositories.

Traditional Approaches: With classical relational database technology alone, synergies between research tools are hard to exploit. For example, we cannot simply establish connections between data stored in EVOLIZER [9] and Sourcerer [1], as it is not possible to set a link from one repository to another – relations are local, not universal. Cross-domain queries spanning multiple repositories are impossible.

The Semantic Web Approach: While the Semantic Web is no panacea, one of its driving forces is the basic assumption that data becomes more useful the more it is interlinked with other data. The simple but powerful concept of statements represented by triples of URIs can be used to build an internet-scale graph of information because it enables us to link and query data that is stored in different locations. Queries on this graph are also possible, thanks to the query language SPARQL.

Let us assume that we build our repositories the way that they generate and expose an URI for every software artifact that they store, and that these URIs are also dereferenceable over the Internet. Then consider a Java class called `Foo` to be stored in one repository – maybe Sourcerer, a search engine for open source code – together with other classes and data about the relationships between them. Assume further that a particular bug report with the number 124 is hosted in a completely independent repository, for example, a Bugzilla or Jira issue tracker, along with other reported bugs, information on their interdependencies, severity and priority, attachments and related discussion threads from bug reporters and developers, and so on. A statement stating that the bug #124 affects the `Foo` class can then be stored in a third repository, possibly our release history database called EVOLIZER. The following s-p-o triple shows how such a statement could look like:

```
http://bugzilla.myProject.org/bugs/nr124
http://myBugOntology.org/affects
http://sourcerer.ics.uci.edu/myProject/Foo.java .
```

Given this triple, client applications, as well as humans, can easily follow the links to access the raw resources and use either the subject, object, or even the predicate as an input for further SPARQL queries.

Scenario 3: Traceability of Results

Description: *Alice is working together with Bruce on her first journal publication describing an empirical study that they have conducted together. To allow other researchers to replicate their results they describe precisely which system releases they used for their study, using the sentence "To demonstrate that our novel approach shows good prediction performance, we selected the five most recent releases from ArgoUML …". Moreover, Alice stores all the data, as well as the detailed results, into a comma separated file, places it on a Web server. In her paper, she lists a link referring to the replication package.*

Key Challenges: The origin of scientific results should be traceable and independently replicable. It has to be made clear what data has been analyzed, *i.e.*, meta data needs to be supplied along with the results. For example, which Java classes were analyzed in particular? Which versions were considered? Was only code on the trunk of a version control system considered, or code in all branches equally? The analysis itself needs to be described in similar detail. What preprocessing steps were applied? Was there any filtering necessary? How did the queries look like exactly?

Traditional Approaches: Publications on empirical studies or machine learning applications in the context of software evolution usually list releases of systems being investigated. In addition, authors often provide input and output files for statical tools, such as R (`http://www.r-project.org/`). On the one hand, these files do not contain any raw, but rather preprocessed – and therefore potentially biased – data. This makes it harder to extend studies of other researchers, because, while preprocessing, subtle differences due to implementation issues can accumulate. On the other hand, the provided input and output files are in the xml or csv format and, as such, do not contain any semantics. This potentially makes them harder to reuse in other tools.

The Semantic Web Approach: In mature scientific fields, authors of publications are requested to provide enough information to allow their findings to be replicated by someone else working independently. Same counts for software engineering research and the Semantic Web may contribute in that respect. RDF statements and URIs describe data under analysis more accurately than sentences like *"To demonstrate that our novel approach shows good prediction performance, we selected the five most recent releases from ArgoUML …"* In addition, authors can make the SPARQL queries public that they have used in their analysis and, since the results are represented again as RDF statements, they can directly be processed in other tools for further investigations, *e.g.*, as input for another query.

Scenario 4: Relations

Description: *Alice has finished her Ph.D. and managed to build an integrated repository to store and query vast information about source code, related bugs, and – thanks to a couple of busy master's students – also requirements documents, such as use case descriptions, and even data about developers,* e.g.*, their code ownership and social network connections. Bruce is more than impressed and starts to explore the repository by querying. In particular, he is interested in finding all kinds of relations between developers and particular Java classes.*

Key Challenges: The quality of a meta model has direct impact on the effectiveness of search interfaces: the meta model (as well as the query language used to query the model) needs to be expressive to allow a broad range of queries and its structure needs to be simple to keep the complexity of common queries, as well as the cost of executing them, low.

Traditional Approaches: There is no consistent way to get the meaning of a relation in relational databases. In fact, a query can join tables by any columns which match by data type – without any check on the semantics. While humans can often guess the meaning of a relation, computers can not. They need to be supplied with additional information. It is therefore necessary to encode a significant amount of implicit knowledge into our applications to make use of the data. To search in an existing repository or to build an own tool on top of it, researchers need to be aware of, and understand this implicit semantics.

The Semantic Web Approach: The SPARQL query language allows, in particular, to query explicitly for relations among resources. Consider the following example query that selects all direct relations between a given developer and a particular Java class:

```
SELECT ?relation
WHERE {
   developer:id101 ?relation javaclass:DBaccess . }
```

The basic graph pattern in the example above consists of a single triple pattern with one variable (?relation). The triple pattern matches all triples where the subject is the developer with the id 101 and the object is a Java class called `DBaccess`, respectively. Each solution gives one way to bind an RDF property to the ?relation variable. The SELECT clause specifies that the ?relation variable is returned as query result. This example demonstrates a SPARQL query which returns the relation between two resources.

These kind of queries are impossible in the relational and in the object-oriented paradigm, unless relationships are explicitly mapped to tables or, in the case of object-orientation, modeled as association classes. The latter, however, can make them difficult to distinguish from "real" classes. Given the high importance of relationships in software engineering, it would be preferable to model them as first class objects – which is exactly what the Semantic Web does.

4. RESEARCH AGENDA

To take advantage of the Semantic Web like envisioned in the scenarios above, the software engineering community should address the following challenges.

Challenge 1: Formalize a common vocabulary to describe software artifacts and their relationships in terms of an ontology. It is relatively uncommon to exchange relational database schemas among different databases but ontologies were explicitly designed to be shared. This also means that one should re-use existing vocabularies whenever possible to enable client applications to process and search in data from many different sources more easily. It is also common practice to mix terms from existing vocabularies. Several ontologies for the domain of software engineering already exist. Some examples are: *Description of a Project (DOAP)* (`http://trac.usefulinc.com/doap`), a vocabulary to describe software projects, and in particular open source. The *Bug And Enhancement Tracking LanguagE (baetle)* (`http://code.google.com/p/baetle/`) describes information kept in bug databases and re-uses, among others, the DOAP ontology. EvoOnt [12] and our *Software Engineering Ontology (SEON)* (see `http://evolizer.org/`) both define a vocabulary to represent information found in version control and issue tracking systems. They also include an ontology for Java source code. This non-exhaustive list of existing ontologies can serve as good a starting point but also needs consolidation and refinement, driven by real scenarios and applications developed by researchers in our domain. In other words, it is a community effort to define an ontology, since it manifests the common, shared data model to represent the data a domain.

Challenge 2: Devise strategies to generate URIs for artifacts of software projects. While redundancy is generally less a problem in the Semantic Web, compared to the relational paradigm, it is crucial that every software artifact can be uniquely identified – and that these URIs can be dereferenced and remain stable over time. Further, a strategy for generating an URI needs to be deterministic and reproducible by other tools. This means that if we parse, for example, the contents of a version control system twice and, in each pass, generate URIs for the source code artifacts found, then the URIs of each pass need to be identical.

5. CONCLUSIONS

In this paper, we have explained how Semantic Web Technology can be used to construct the next generation of software repositories. In our vision, software repositories should blend into a query-able global information space of interlinked data about all possible software engineering artifacts. However, it requires the software engineering community to agree on two issues: (a) an ontology to describe the software artifacts and their relationships; (b) a strategy for generating URIs for such software artifacts.

In the meantime, what should happen to the existing repositories? Indeed, research groups have invested a significant amount of effort in building these software repositories: it is unrealistic – but, fortunately, completely unnecessary – to throw them away. A tutorial on *How to Publish Linked Data on the Web* can be found in [4]. The tutorial also features a section on how to publish existing relational database as *Linked Data* and lists several tools to do so. So migration strategies that allow for an incremental adoption of ontologies and URIs are entirely feasible, and we encourage the research community to make the necessary preparations.

6. REFERENCES

[1] Sushil Bajracharya, Joel Ossher, and Cristina Lopes. Sourcerer: An internet-scale software repository. In *Proc. ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation*, pages 1–4, Washington, DC, USA, 2009. IEEE Computer Society.

[2] Tim Berners-Lee, R. Fielding, and L. Masinter. RFC 2396 - uniform resource identifiers (URI). IETF RFC, Aug. 1998. `http://www.ietf.org/rfc/rfc2396.txt`.

[3] Jennifer Bevan, Jr. E. James Whitehead, Sunghun Kim, and Michael W. Godfrey. Facilitating software evolution research with Kenyon. In *Proc. Joint European Softw. Eng. Conf. and ACM SIGSOFT Symposium on the Foundations of Softw. Eng.*, pages 177–186. ACM, Sept. 2005.

[4] Chris Bizer, Richard Cyganiak, and Tom Heath. How to publish linked data on the web. `http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/`, Last visited Jan. 2010.

[5] Yih-Farn Chen, Emden R. Gansner, and Eleftherios Koutsofios. A c++ data model supporting reachability analysis and dead code detection. *IEEE Trans. Softw. Eng.*, 24(9):682–694, 1998.

[6] Marco D'Ambros, Harald C. Gall, Michele Lanza, and Martin Pinzger. Analyzing software repositories to understand software evolution. In *Software Evolution*. Springer-Verlag, Heidelberg, Germany, 2008.

[7] Mike Dean and Guus Schreiber eds. *OWL Web Ontology Language Reference*. W3C Recommendation, Feb. 2004. `http://www.w3.org/TR/owl-ref/`.

[8] Michael Fischer, Martin Pinzger, and Harald Gall. Populating a release history database from version control and bug tracking systems. In *Proc. Int'l Conf. Software Maintenance*, pages 23–32, Sept. 2003.

[9] Harald C. Gall, Beat Fluri, and Martin Pinzger. Change Analysis with Evolizer and ChangeDistiller. *IEEE Softw.*, 26(1):26–33, Jan./Feb. 2009.

[10] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199 – 220, 1993.

[11] Huzefa Kagdi, Michael L. Collard, and Jonathan I. Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *J. Softw. Maint. Evol.*, 19(2):77–131, 2007.

[12] Christoph Kiefer, Abraham Bernstein, and Jonas Tappolet. Mining software repositories with iSPARQL and a software evolution ontology. In *Proc. Int'l Workshop on Mining Software Repositories*, page 10, Washington, DC, USA, 2007. IEEE Computer Society.

[13] Graham Klyne and Jeremy J. Carroll eds. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation, Feb. 2004. `http://www.w3.org/TR/2004/REC-rdf-schema-20040210/`.

[14] Object Management Group. XML Metadata Interchange (XMI). Technical Report OMG Document ad/98-10-05, Feb. 1998.

[15] Eric Prud'hommeaux and Andy Seaborne eds. SPARQL query language for RDF. W3C Recommendation, 15 Jan. 2008. `http://www.w3.org/TR/rdf-sparql-query/`.

[16] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? In *Proc. Int'l Workshop on Mining Software Repositories*, pages 1–5, New York, NY, USA, 2005. ACM.

[17] Sander Tichelaar, Stéphane Ducasse, and Serge Demeyer. FAMIX and XMI. In *Proc. Working Conf. Reverse Eng.*, page 296, Washington, DC, USA, 2000. IEEE Computer Society.

[18] Andreas Winter, Bernt Kullbach, and Volker Riediger. An overview of the GXL graph exchange language. In *Revised Lectures on Software Visualization, International Seminar*, pages 324–336, London, UK, 2002. Springer-Verlag.