Towards Sharing Source Code Facts Using Linked Data

Iman Keivanloo, Christopher Forbes, Juergen Rilling
Department of Computer Science and Software Engineering
Concordia University
Montreal, Canada
{i_keiv, c_forb, rilling@cse.concordia.ca}

Philippe Charland
System of Systems Section
Defence R&D Canada – Valcartier
Quebec, Canada
philippe.charland@drdc-rddc.gc.ca

ABSTRACT
Linked Data is designed to support interoperability and sharing of open datasets by allowing on the fly inter-linking of data using the basic layers of the Semantic Web and the HTTP protocol. In our research, we focus on providing a Uniform Resource Locator (URL) generation schema and a supporting ontological representation for the inter-linking of data extracted from source code ecosystems. As a result, we created the Source code ECOsystem Linked Data (SECOLD) framework that adheres to the Linked Data publication standard. The framework provides not only source code and facts that are usable by both humans and machines for browsing or querying, but it will also assist the research community at large in sharing and utilizing a standardized source code representation. The dataset has been submitted and registered to ckan.net, under the SECOLD project name, as the first source code Linked Data repository. In order to maintain its relevance to the research community, we plan to update the data set every four months.

Categories and Subject Descriptors
H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval – *Retrieval models;* H.3.5 [**Information Storage and Retrieval**]: Online Information Services – *data sharing, web-based services*

General Terms
Documentation, Design, Experimentation, Standardization

Keywords
Linked Data, source code model, ontology, Semantic Web

1. INTRODUCTION
Mining software repositories and Internet-scale source code search/analysis [8] are two active research areas. A commonality between both domains is that they require major pre-processing steps to allow for the sharing and integration of data to be mined or searched. These steps are time-consuming tasks due to the heterogeneity and constant changes of the data and structures. XML-based exchange formats were developed over recent years [9] to address these shortcomings. While these models work well for smaller and stable datasets, automated integration and sharing of large distributed heterogeneous data are beyond the capability of XML and relational databases (DBs) [11].

ICSE'11, May 21–28, 2011, Waikiki, Honolulu, HI, USA.

Linked Data [1] is a sub-product of the Semantic Web and has been promoted to address these interoperability/sharing issues for open datasets. It enables both humans and machines to interpret the data for mining, searching, and analysis purposes. Linked Data introduces some basic techniques for data publication over distributed environments like the Internet. First, each entity has a unique identifier (i.e. URL). Second, the content must be published using URLs and a common vocabulary. Although Linked Data was defined only recently, it has already been accepted in diverse domains (linkeddata.org) such as health care [3] and mathematics [2]. This popularity is the result of two main factors. First, contrary to the Semantic Web (in general), Linked Data guidelines do not rely on heavy reasoning and complex logic. Second, unlike data exchange mechanisms such as CSV, XML, relational DBs, and web services, it does not require a pre-defined data structure. Instead, it uses a common vocabulary set where it is defined using machine understandable language (contrary to pure XML). The vocabulary set models concepts and relations in the domain of discourse.

In [11], Würsch et al. suggested two essential steps as a research agenda for the software community: (1) design a common vocabulary set (i.e. ontology); (2) create a unique identifier generation schema for software repository entities. In our research, we address both steps by providing a framework to publish Linked Data sets for source code ecosystems. This enables data sharing and integration for the software engineering community at large. We devised a common vocabulary set for source code ecosystems that covers most aspects of source code such as revisions, presentation, syntax, and semantics. Along with the ontological representation, an identifier generation schema was defined. The schema guarantees that each real or abstract entity will have its own unique identifier. Identifiers are constructed by including entity type, revision, local identifier, and some other basic information in their naming convention. The resulting schema works in both distributed and centralized environments. It does not require any synchronization mechanism to guarantee identifier uniqueness. We call these *Reproducible Identifiers*, since by following the schema, the same URL can be generated for an entity at any time, by any given tool.

As part of our research, we developed a Linked Data publication framework for source code ecosystems. This framework provides the largest and first publicly available online Linked Data source code dataset to software engineering researchers and practitioners. The resulting Source code ECOsystem Linked Data (SECOLD) consists of 1.5 billion triples. We have used a source code set [6] as the input for our first release of SECOLD. The resulting dataset provides line-level and statement-level granularity for the presentation and syntax layers respectively. It is available in four forms (1) online HTML (for humans) (2) online RDF/XML (for code search tools) (3) dataset dump files (for research purposes) (4) public query endpoint (for structural query).

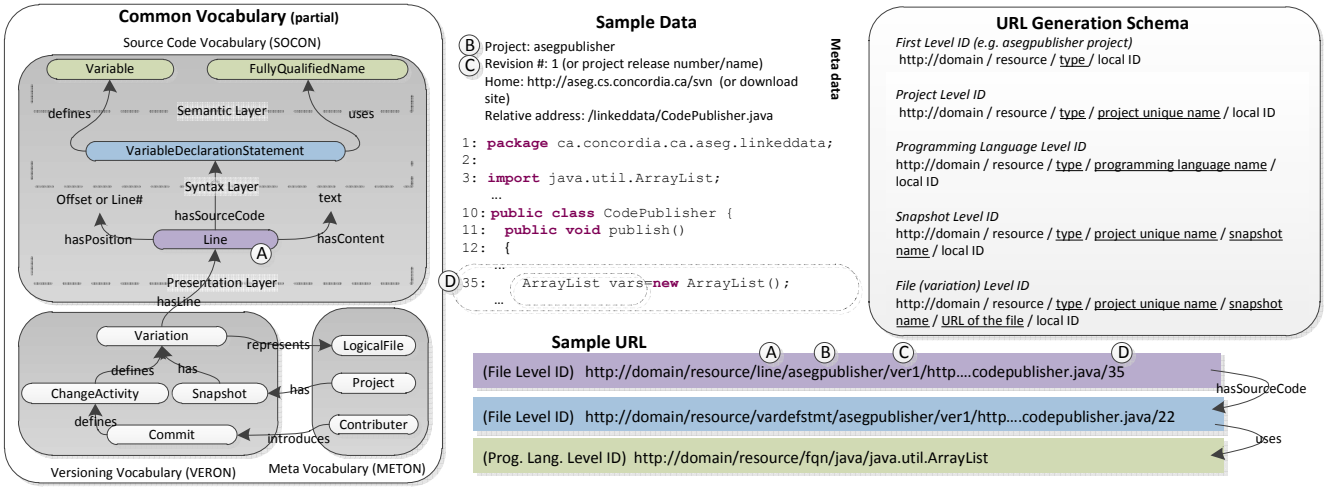

Figure 1. Part of the *Common Vocabulary Set* (top left) and the complete *URL Generation Schema* (top right) are shown. A set of *reproducable* URLs for the given source code is shown. Following the bubbles reveals how they are created.

The ID schemas, common vocabulary, guidelines, data conversion services, and the dataset are available online [10] at http://aseg.cs.concordia.ca/secold.

The remainder of this paper is organized as follows: Section 2 briefly reviews Linked Data. Section 3 and 4 discuss how our approach addresses the research challenges presented in [11]. The Linked Data framework for source code ecosystems and its usage scenarios discussed in Section 5 and 6.

2. LINKED DATA

Linked Data was introduced to ease data sharing and integration in a distributed environment and be superior to XML-based approaches [1, 11]. Each entity in the domain of discourse must have a unique identifier (UID) in the form of a URI (Uniform Resource Identifier). Note that UID, URI, and URL will be used interchangeably throughout the paper. Facts about the *resource* (i.e. entity) are represented using RDF *statements*, with each fact statement being a triple of subject, predicate, and object. To make this information inter-linkable and online, Linked Data mandates that the URLs must be *dereferencable.* That is, clients (i.e. humans and machines) must be able to fetch resource related data via its URL (with http:// prefix). Using a HTTP header, a client specifies the desired output format: HTML or RDF/XML. A response must comprise the following: (1) *Description* and *Backlinks* that contain all triples that have the URL as the subject and object. (2) Equivalent URLs that point to the same entity (i.e. `owl:sameAs`).

3. REPRODUCIBLE IDENTIFIERS

Software analysis research often relies on the use of several tools to analyze the same artifact and gather different types of information into a centralized dataset for final processing. For example, a Javadoc analyzer can be used to extract information related to the available documentation within the code repository and a Java parser may be used for inheritance tree generation. Both tools use the same resource (the source code) as input. The key challenge for the results integration within a centralized dataset is that the results adhere to a common naming and format schema [11]. Inefficiency is highest when each tool produces identifiers using random or tool specific generated result formats, thus, making difficult or impossible to find the equivalent entities in the other dataset. This scenario can be compounded if one uses the same tool to parse a repository several times, which would create several random identifiers for a single entity. An elegant approach is to use the information available in the software ecosystem (e.g. source code, version control, issue tracker) to create stable identifiers. We call these *Reproducible Identifiers*, which are independent from the producer tool's logic and analysis context. This results in a unique identifier for an entity in a centralized or distributed environment. This addresses the second research challenge in [11].

In order to create *Reproducible Identifiers*, we need anchors, which are always available, and context independent. Given such an anchor, a *Reproducible Identifier* is generated by concatenating a set of predefined anchors. The anchor selection itself depends on the entity type and is the most challenging step during the software ecosystem Linked Data population. For example, in Java source code analysis, it is not always possible to specify the type (i.e. class or interface) of an imported type. Failure to choose a stable anchor results in multiple ID for a single entity, which would then violate the premise of *Reproducible Identifiers*. In order to avoid this violation of stability and uniqueness of IDs, a precise examination of the domain of discourse is required.

As part of our research, we introduce a novel ID generation schema for source code ecosystems using available source code (language independent) and versioning information. This schema follows three rules to guarantee *Reproducibility* of the IDs in practice. (1) *Context independency rule*: The anchors must not be selected from the context of an analysis environment. (2) *Right granularity rule*: the anchors must not be either too specific (since this might cause instability) or too general (since this would reduce the effectiveness of the triple repository indices). (3) *Abstraction level dependency rule*: for each entity, there must be some anchor referring to the abstraction or revision notion. This is necessary when there are several levels of abstraction or revisions of an entity. We have identified five patterns based on the above three rules shown in Figure 1. Underlined terms correspond to *anchors*, which vary based on the entity. The *type* is specified using the vocabulary set (ontology). The *Local ID* is generated from the entity itself using such information as line number, project title, etc. Local IDs must also conform to the *Right granularity rule.* For each type within the ecosystem, we have

defined anchors and a Local ID which are available online. It should be noted that, the first three schemas are independent from versioning information. Regardless of the schema, all assigned URLs are fixed. For example, changing a file location does not require altering existing URLs, only starting facts affected by the change will use the new URL.. However, if required, tractability links can be established between the old and the recent URLs.

4. SOURCE CODE ECOSYSTEM MODEL

As stated in [11], another key research challenge in software engineering is the need for a common vocabulary framework. In what follows, we introduce SECON (Source code ECosystem ONtology family) which was designed to address this specific research community requirement. Our model covers source code and versioning concepts as the core artifacts of a source code ecosystem. SECON is built based on a layered model shown (partially) in Figure 1. Due to space limitations, we describe only the underlying rationale of our conceptualization in the domain of discourse. The model and complementary documentation are available online [10]. Our model covers every aspect of source code such as source code presentation (e.g. tokens), syntax (e.g. statements) and semantics (e.g. call graphs), including versioning notion as the baseline. To the best of our knowledge, there is no other (publicly available) comprehensive source code ontology family.

Regardless of the application context of the model, versioning information must be considered during data population to satisfy the *(unique) Reproducible Identifiers* concept. It does not matter if we receive the versioning information from a source control module such as SVN, or a manually maintained revision set, like a downloadable project release. As an example, consider a tool that produces a URL for a Java expression such as `streamHandle==-1`, implemented in one of the `java.lang.Runtime` class revisions. In order to avoid an ambiguous or invalid result, the URL must be unique to the particular statement, implemented in a specific revision. Although one could argue that all revisions of this statement are related to each other at some level, this does not mean that they must have similar URLs in every level of abstraction. In the conceptualization of our domain, we consider revisions of statements as different entities despite the possibility of them being connected to an abstract entity at a higher level of granularity. Therefore, it is possible to include traceability links between corresponding statements in different revisions with no information loss using the proper tools.

4.1 Versioning Ontology Model (VERON)

There are many version control software systems such as CVS, SVN, Git, etc. While providing fairly similar functionalities, they differ significantly in the way they handle the actual versioning task. For example, SVN adds a new record, called a commit, for each set of changes, and assigns the changed files to the committed entity. Alternatively, CVS adds a new commit record for each changed file. Thus, the same container entity does not exist for CVS. Capturing these implementation differences is the main source of challenges during the ontology design. The objective of our Versioning Ontology Model (VERON) was to overcome these difficulties, by providing a design that creates the output data that is uniformly independent of the input (e.g. a set of manually maintained project releases). Existing versioning ontologies [5] do not address the uniformity criterion. We avoided tool-specific terms in VERON to increase its universality.

Revision is the core concept in VERON. It represents a specific temporal instance of a file, which is identified by its physical file address and revision number. A set of *revisions* constitute a *snapshot*. Each *commit* has a set of *change activities,* while each activity introduces a new *revision* to the system. Revisions of a specific file must be connected to each other at a higher abstraction level. The *Logical File* concept is introduced for this purpose in our Meta Data Ontology Model for source code ecosystems (METON). That is, all instances (i.e. revisions) of a specific file belong to the corresponding logical file entity, which can now be identified by its Internet address.

4.2 Source Code Ontology Model (SOCON)

SOCON is our comprehensive model for both code sharing and analysis. In this model, we distinguish three main layers: *Presentation*, *Syntax,* and *Semantics*. The *Presentation* layer models entities such as token, line of code, fragment (a set of lines of code), and the ordering property. Although most of the source code analysis/mining techniques do not require this level of information, we focused on it, since it serves as a connection point for the other layers. The *Syntax* layer models entities such as variable definition statements, etc. Usually, this information is extracted from the Abstract Syntax Tree (AST). Note that the first release of SOCON covers the object-oriented language paradigm. The top abstraction level is the *Semantic* layer. It addresses the modeling of output from static or dynamic code analysis, e.g. call-graph links and code clones. The vocabulary set of this layer is not static to accommodate different analysis techniques. While the two first layers are useful for querying the structural aspects of source code, the *Semantic* layer is used for complex queries (e.g. code clone search).

All entities, regardless of the layer, are connected and traceable. Source code statements are connected to the *Presentation* layer (e.g. line of code) via the *hasSourceCode* property shown in Figure 1. Note that the tail (i.e. range) of the property is not a textual representation of code. It can be a source code statement, expression, or block (derived from the AST). *hasContent* is the property for textual code modeling. The connections to and from the Semantic layer are provided by either *hasSourceCode* property or special relations such as *hasFQN* (Fully Qualified Name).

5. LINKED DATA FRAMEWORK

The two major objectives of our designed and implemented framework for source code Linked Data publication are: (1) To provide a set of public services, including the documentation of the common vocabulary set (SECON), and URL generation schemas. The framework (Figure 2) includes online services for fact extraction from version control systems or manually downloaded code. It also generates the output in the form of N-Triples and RDF formats. All of these functionalities are made publicly available to the software research community [10]. (2) The framework crawls the Web and publishes the Linked Data extracted from open source code. The data is accessible via a web browser (HTML format), query interface, or dump files. Our SECOLD web server is the first framework for Linked Data that not only allows source code data sharing, but also conforms to Linked Data publication rules [1]. The repository contains unique identifiers and their relationships created by analysis modules. It is befitting for client applications because of its RDF output. So far, all mandatory modules (solid borders) have been implemented and optional modules (dashed border) will be implemented at a later stage.

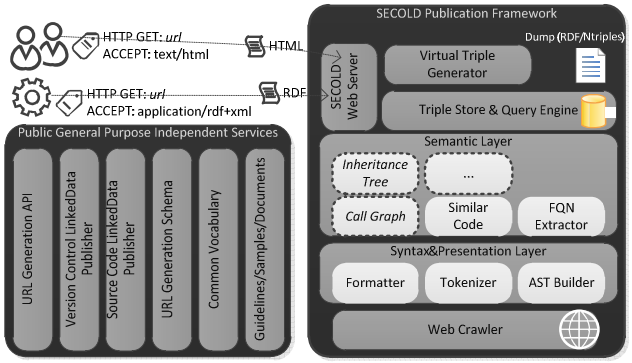

Figure 2. Linked Data framework for source code publication.

Since the number of real triples in the source code ecosystem is huge, we added a special layer called Virtual Triple Generator (VTG). It generates `rdfs:label`, `rdfs:comment`, and `socon:hasContent` literal triples (not source code facts) on the fly. The VTG allows for a reduction of the size of repository.

6. USAGE SCENARIOS

The overall goal of this research is to provide some baselines for universal identifier generation for all entities in a source code ecosystem, such as a contributor, line of code, and call graph link. The objective is to provide the software engineering research community at large with a starting point to facilitate various research activities. In what follows, we discuss three potential application scenarios for SECOLD. They benefit from the facility of Linked Data and its standard data access solutions [1].

Software mining and analysis. Würsch et al. [11] present four potential usage scenarios, which are addressed by our research contributions, to be able to merge two different datasets without the need for ID alignment. Furthermore, if one of the datasets has extended the vocabulary set (ontology), the query engine (i.e. Semantic Web triple store with RDFS reasoning) can handle this extension elegantly. In addition, our online SECOLD can be used as a valuable online resource for the research community to support *clone genealogy* studies, for example. Our vision is to provide an online dataset for the software research community similar to other domains, such as health informatics [3], where everyone share their dataset which are inter-linked on the fly (e.g. we plan regular updates every four months).

Internet-scale code search. SECOLD dataset would be accessible for applications such as Parseweb [8] via HTTP based querying [1]. They can retrieve the extracted code facts by sending a simple HTTP request or SPARQL query. Some interesting queries are (1) All superclasses and subclasses of the given type (supporting transitivity) (2) All similar lines of code. Project, file, class, line, import statement, and code similarity are some of the facts available in the first release. The framework currently supports three types of line similarity detection differing in precision and speed. We have implemented a scalable clone detection tool within the Semantic Layer for the similarity module. To resolve FQNs, we use a technique called *loose unqualified name resolution* [4].

Software maintenance, documentation, and traceability. The last usage scenario is *traceability*. We produce a URL for each piece of source code and extracted facts. This URL would be used to refer to the entity in software documentation and online discussions. It eases the tractability task.

7. CONCLUSIONS

Our presented research has three major contributions. A common vocabulary set (ontology), the URL generation schema, and publication framework (which conforms to the Linked Data guidelines [1]) for source code ecosystems. It addresses the following three objectives: (1) create a source code Linked Data repository (SECOLD) for the software research community which will be updated every four months; (2) provide query services for Internet-scale code search and mining tools; and (3) to provide a set of public services for URL generation and data conversion from source code and version control systems. Our framework and the Linked Data dataset are available online [10]. It is registered to CKAN under the name SECOLD. The first release of SECOLD contains 1.5 billion triples extracted from 1.5 million Java files. It is connected to DBpedia, Freebase, and OpenCyc within the LOD cloud. In the future, we are planning to support additional languages (e.g. C++ and C#), provide API for FAMIX [9] and SourcererDB [7], and include inter-linked issue trackers.

8. ACKNOWLEDGMENTS

This research was partially funded by DRDC Valcartier (contract no. W7701-081745/001/QCV) and supported by Franz Inc. by providing AllegroGraph.

9. REFERENCES

[1] Berners Lee, T. Linked Data, *http://www.w3.org/DesignIssues/LinkedData.html*. Last visited Jan. 2011.

[2] David, C., Kohlhase, M., Lange, C., Rabe, F., Zhiltsov, N., and Zholudev, V. Publishing Math Lecture Notes as Linked Data. *Lec. Notes in Comp. Science* (6089/2010), 370-375.

[3] Jentzsch, A., et al. 2009. Enabling tailored therapeutics with linked data. In *Proc. 2nd Workshop Linked Data on the Web.*

[4] Keivanloo, I., Roostapour, L., Schugerl, P., Rilling, J. 2010. Semantic web-based source code search. In *Proc. 6th Intl. Workshop on Semantic Web Enabled Software Engineering.*

[5] Kiefer, C., Bernstein, A., and Tappolet, J. 2007. Analyzing software with iSPARQL. In *Proc. of the 3rd Intl. Workshop on Semantic Web Enabled Software Engineering*.

[6] Lopes, C., Bajracharya, S., Ossher, J., Baldi, P. 2010. UCI Source Code Data Sets. *http://www.ics.uci.edu/~lopes/datasets*. Irvine, University of California.

[7] Ossher, J., Bajracharya, S., Linstead, E., Baldi, P., Lopes, C. 2009. SourcererDB : An Aggregated Repository of Statically Analyzed and Cross-Linked Open Source Java Projects. In *Proc. Working Conf. on Mining Software Repositories*.

[8] Thummalapenta, S., and Xie, T. 2007. Parseweb: a programmer assistant for reusing open source code on the web. In *Proc. of 22th Intl. Conf. on Automated Soft. Eng*.

[9] Tichelaar, S., Ducasse, D., and Demeyer, S. 2000. FAMIX and XMI. In *Proc. Working Conf. Reverse Eng*. DC, USA.

[10] Keivanloo, I., Forbes, C., Rilling, J. Source Code Ecosystem Linked Data (SECOLD), *http://aseg.cs.concordia.ca/secold.*

[11] Würsch, M., Reif, G., Demeyer, S., and Gall, H.C. 2010. Fostering synergies: how semantic web technology could influence software repositories. In *Proc. on Search-driven Dev: Users, Infrastructure, Tools and Eval.* (SUITE '10).