2016 IEEE Tenth International Conference on Semantic Computing

# SPARQL Queries over Source Code

Mattia Setzu
Dept. of Mathematics and Computer Science
University of Cagliari
Via Ospedale 72, 09124 Cagliari (Italy)
Email: ma.setzu2@studenti.unica.it

Maurizio Atzori
Dept. of Mathematics and Computer Science
University of Cagliari
Via Ospedale 72, 09124 Cagliari (Italy)
Email: atzori@unica.it

***Abstract*—We introduce a framework to extract and parse Java source code, serialize it into RDF triples by applying an appropriate ontology and then analyze the resulting structured code information by using standard SPARQL queries. We present our experiments on a sample of 134 Java repositories collected from Github, obtaining 17 Million triples about methods, input and output types, comments, and other source code information. Experiments also address the scalability of the framework. We finally provide examples of the level of expressivity that can be achieved with SPARQL by using our proposed ontology and semantic technologies.**

## I. Introduction

Recent research in software engineering is focusing on improving the user experience of computer programmers by proposing tools that make finding and reusing existing software code very easy. For instance, Microsoft Research is developing a search tool called *Bing Code Search/Developer Assistant* [1] that extracts contextualized C# code from different HTML sources, such as StackOverflow[1], based on text questions like "how to read file line by line". Other research focused on searching and automatically recalling code based on type information, such as JavaSketch/Prospector [2], with an ad-hoc simplified query language, or a rule-based approach to code transformation [3]. These approaches seem to either focus on textual queries (sometimes keyword based such as in [4]) or more structured queries that do not take textual information into account. Other work, such as the *Web of Functions*, focused on reusing code by sharing it over the internet and making use of it from SPARQL through remote procedure calls [5], [6].

In this paper we propose to leverage the existing open-source codebases available on massive repositories such as Github to extract structured information (such as class hierarchies, methods, types, etc.) and semistructured and unstructured information (such as textual comments) in order to generate a large set of RDF triples that can be queried for different purposes, including code completion, automatic code writing, code mining, etc. We apply well-known semantic web techniques such as automatic entity annotation algorithms to enrich the content of otherwise unstructured text comments found in the source code. This way we obtain 3 main results: *(i)* we provide great expressivity by using an existing standard query language such as SPARQL, *(ii)* we instantly get benefit of millions of triples extracted from existing codebases that can be then queried, and *(iii)* we make both structured and unstructured information usable at the same level, enriching textual information with semantic web annotations.

We believe that our proposal shows the benefits that Semantic Web technologies can provide in research areas such as software engineering and computer-aided programming, also contributing with tools and the resulting dataset to further investigation.

With respect to software development, semantic search may allow developers to substantially reduce time and costs by introducing enhanced ways to code reuse, focusing on existing code bases that have been already tested and in production. Existing text-based search engines are not appropriate for this task as they fail identifying the code scope, its dependencies and the most abstract and implicit relations. Here a few examples of limitations of keyword-search approach: given a keyword, it is unlikely to find a snippet of code whose functionality is described using that specific word; a retrieved snippet of code cannot be seamlessly executed as it is, due to lack of dependencies or entities the code refers to, but not declared in the snippet itself; classes, packages and variables referenced would not be self-evident on a text-only basis; given two snippets of code it is not possible to seamlessly merge them for the same reasons, nor is possible to know when two code elements in two different snippets refer to the same entity.

We propose a semantic search layer on top of a full-text search that enables structured queries (later discussed in section III) and provides a transitive layer for natural language queries.

## II. The Framework

The framework is based on a workflow that follows four sequential steps described in this section. Some parts, such as the ontology we propose and the use of SPARQL for querying the resulting dataset, are detailed later in this paper.

*Extraction.* The first step is aimed at collecting a set of repositories from Github and their mandatory dependencies. Since most of recently developed Java code is built and distributed through automated build tools, our extraction tool supports the two most popular, namely *Gradle* and *Maven*. Both allow the programmers to specify the program structure, its dependencies in name and version and other minor details. The very first step is therefore to identify the build tool used in each repository, and automatically and recursively download all dependencies in order to get a valid classpath. The tools avoid multiple downloads of the same package.

[1] http://stackoverflow.com/

978-1-5090-0662-5/16 $31.00 © 2016 IEEE
DOI 10.1109/ICSC.2016.65



IEEE computer society

Authorized licensed use limited to: University of Pisa. Downloaded on July 02,2023 at 16:13:45 UTC from IEEE Xplore. Restrictions apply.

*Parsing.* The parsing and error checking is managed by *Spoon*, a Java source code parser and analyzer. All downloaded software modules get checked for errors; then *Spoon* creates an enhanced Abstract Syntax Tree (AST) of the source code. We developed a Spoon *Processor* that recursively collects: method's names, parameters and return types, classes, modifiers, javadocs, raw source code snippets, and references to other entities (e.g., classes and interfaces) in the source code. They are internally saved by the crawler using a pseudo-RDF representation.

*Serialization.* The serialization phase maps the internal representation into RDF triples using our proposed ontology described later in the next section. In order to do this we take advantage of the *Apache Jena* libraries, producing a dataset in the N-Triples format.

*Semantic Enrichment.* We use state of the art entity recognition services for text analysis [7], [8], based on Wikipedia data, to provide an high-level tagging of the Javadoc comments. A script to fetch data for the REST service has been created. Javadoc comments are therefore tagged, creating an extra N-Triples dataset with their semantic annotations.

## III. An Ontology to Describe Source Code

The ontology we propose to model the Java language, created with the *Protégé* semantic tool, is inspired by previous work in [9] as it provides definitions for most of the same entities, and is built on three layers, *core*, *object-oriented* and *Java* to enhance flexibility. For our purposes, we extended the original proposals with a number of definitions. For instance, *declared_by*, *has_parameter*, *parameter_position*, *source_code* have been added to allow source graph walks in both directions, to keep track of the dependencies of each method and entity, and retrieve every element's source code. In Fig. 1, an excerpt of the `is-a` hierarchy is show. Our current ontology consists of 74 entities (e.g., `woc:Method`) and 217 triples (e.g., the property `woc:has_parameter` for the entity `woc:Method`).

### A. Extensibility

While the ontology is focused on the Java 1.8 syntax specification, it provides several definitions in the *core* and *object-oriented* layers which are common to other paradigms, thus being redundant but more expressive. As a consequence, while our work focused on Java only, languages sharing most design guidelines with Java can be described in the ontology with only a minimum set of additional definitions.

For instance, let us use Scala and its higher order functions. The Java layer defines both methods and parameters as *datum*, defined as a self-existent code element. Trivially, we'll be able to define a function as a *datum* and a *parameter* as either a *type* or a *function* itself. Scala defines types for functions: we will have triples similar to these ones: *function a type*, *function a datum*, *parameter a type*, *parameter a datum*, without affecting the *core* or the *object-oriented* layers. As a worst-case scenario let us use a language relatively recent, with a highly specific target, that is *GO* and its *goroutines*. As a concurrent-specific feature, such definitions do not belong to the *core* layer, nor to any of the ones defined in the ontology. The ontology developed is oriented towards languages as formal implementations of Turing-complete implementations of languages and shall not define specific paradigm or hardware-related concepts in order to preserve a tiny core and great flexibility.

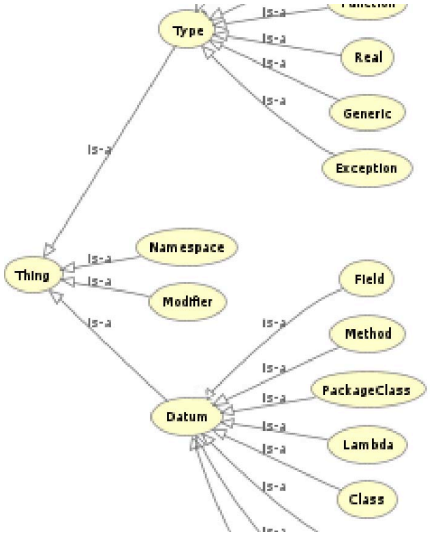


Fig. 1. An excerpt of the `is-a` ontology property.

## IV. Experiments

In our experiments we considered a sample of 134 Github repositories, selected among the most starred in the platform using either *Gradle* or *Maven*. The tool can also run on repository with others or none build system, provided a valid classpath is given. In Table I we show a subset of them, with quantitative information about the data we extracted, including overall information obtained by summing up data from all 134 repositories.

TABLE I. Repositories

| Repository | LLOC (logical code lines) | Triples | Individuals | Classes | Methods |
|---|---|---|---|---|---|
| Activemq | 577072 | 4520027 | 25633 | 2397 | 7215 |
| FB-Presto | 333194 | 429280 | 25227 | 4481 | 20255 |
| Junit | 37975 | 50955 | 2006 | 1129 | 2125 |
| A.Jena Core | 195561 | 179194 | 19174 | 1469 | 6552 |
| OrientDB | 389038 | 172473 | 4722 | 316 | 1603 |
| Clojure | 58067 | 59117 | 7222 | 357 | 2247 |
| *Overall* | *15209817* | *17115353* | *738444* | *121847* | *414723* |

We also provide response time on the *Overall* dataset in I, obtained by looking for all triples, classes, etc. against our Fuseki 2.13 server on a Compaq Presario CQ58 (1.1GHz x86_62 AMD 20, 4GB DDR3 RAM, 500GB 5400rpm HD):

TABLE II. Query times

| *Query* | Triples | Classes | Methods | Interfaces |
|---|---|---|---|---|
| *Time(ms)* | 395538 | 2958 | 14928 | 704 |

## V. Querying Source Code using SPARQL

In the demo we will show the effectiveness of the framework and the expressivity of SPARQL against our dataset and ontology. In the following we now show two examples of possible queries that can be handled



Authorized licensed use limited to: University of Pisa. Downloaded on July 02,2023 at 16:13:45 UTC from IEEE Xplore. Restrictions apply.

with our approach. Notice that the prefix `woc` stands for `<http://rdf.webofcode.org/woc/>`. Further information can be found at the project website: http://atzori.webofcode.org/projects/SPARQL4coding/

*Get the source code of all methods which take no parameters in input and returning an* int

```
SELECT ?code WHERE {
    ?method rdf:type woc:Method .
    ?method woc:returns woc:int .
    ?method woc:source_code ?code
    FILTER NOT EXISTS {
        ?method woc:has_parameter ?par
    }
}
```

*Get all classes referenced by any boolean-returning method of* org.junit.assert.Assert

```
SELECT ?class WHERE {
    ?method rdf:type woc:Method .
    ?method woc:declared_by
        woc:org.junit.assert.Assert .
    ?method woc:returns woc:boolean .
    ?method woc:request ?method_invoked
    ?method_invoked woc:declared_by ?class
    FILTER NOT EXISTS {
        ?method woc:has_parameter ?par1 .
        ?par1 woc:parameter_position 1
    }
}
```

*Get all methods that are recursive*

```
SELECT ?method WHERE {
    ?method rdf:type woc:Method .
    ?method woc:request ?method
}
```

In the following query we show an advanced query that makes use of semantic annotations of comments of our framework plus the property paths feature of SPARQL 1.1.

*Get methods for writing and managing PNG images, with required libraries/classes code*

```
SELECT ?method WHERE {
    ?method rdf:type woc:Method .
    ?method rdfs:about
        dbpedia:Portable_Network_Graphics .
    ?method rdfs:about dbpedia:Computer_file .
    ?method rdfs:about dbpedia:Writing .

    # source code
    ?method woc:source_code ?source .
    # declaring type
    ?method woc:declared_by ?c .
    # package
    ?pack woc:package ?c
    # return type
    ?method woc:returns ?return .
    # parameters
    ?method woc:has_parameter ?par .
    ?par woc:type ?par_type .

    # source code for required elements
    ?method
        (woc:requests+ /
        (<http://rdf.webofcode.org/woc/>)*
        / woc:source_code) ?code .

}
```

This query returns the source code methods, plus the source code for the types referenced by the method itself recursively. For instance, if a method *m()* returns a new object of type *T*, then *T* is included in the query results; if then *T* has a field of type *T'* also *T'* is included in the results and so on. We are so able to get the scope for the methods we searched for.

## VI. Conclusions

The paper briefly presented a framework that allows to take advantages of petabytes of existing source code, transforming it into useful structured data through a general ontology for object-oriented languages, and adding semantic annotations to comments. The result is a layer queryable by SPARQL that can be exploited to create different advanced developer tool. Future work shall aim to make the framework multi-thread and add support for other build tools and their configuration files. Another future work may focus on developing the ontology further, in order to support high-level features such as an improved representation of source code dependency.

## Acknowledgments

This work was supported in part by a *Google Faculty Research Award* (Winter 2015) and by MIUR PRIN 2010-11 project *Security Horizons*. The authors wish to thank Massimo Bartoletti, Livio Pompianu and Carlo Zaniolo for the insightful discussions on related topics.

## References

[1] S. G. Y. H. Yi Wei, Nirupama Chandrasekaran, "Building bing developer assistant. MSR-TR-2015-36," Microsoft Research, Tech. Rep., 2015, tool available at http://codesnippet.research.microsoft.com/.

[2] D. Mandelin, L. Xu, R. Bodík, and D. Kimelman, "Jungloid mining: helping to navigate the API jungle," in *ACM SIGPLAN 2005 Conf. on Programming Language Design and Implementation*, 2005, pp. 48–61.

[3] S. P. Reiss, "Semantics-based code search," in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 243–253. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2009.5070525

[4] G. Little and R. C. Miller, "Keyword programming in Java," *Autom. Softw. Eng.*, vol. 16, no. 1, pp. 37–71, 2009.

[5] M. Atzori, "Toward the web of functions: Interoperable higher-order functions in SPARQL," in *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference*, 2014, pp. 406–421.

[6] ——, "call: A nucleus for a web of open functions," in *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference*, 2014, pp. 17–20.

[7] P. N. Mendes, M. Jakob, A. García-Silva, and C. Bizer, "Dbpedia spotlight: shedding light on the web of documents," in *I-SEMANTICS 2011, Graz, Austria, September 7-9, 2011*, 2011, pp. 1–8.

[8] P. Ferragina and U. Scaiella, "Fast and accurate annotation of short texts with wikipedia pages," *IEEE Software*, vol. 29, no. 1, pp. 70–75, 2012.

[9] R. Dabrowski, K. Stencel, and G. Timoszuk, "Software is a directed multigraph," in *ECSA 2011, Essen, Germany*, 2011, pp. 360–369.



Authorized licensed use limited to: University of Pisa. Downloaded on July 02,2023 at 16:13:45 UTC from IEEE Xplore. Restrictions apply.