CodeOntology: RDF-ization of Source Code

Mattia Atzeni and Maurizio Atzori(✉)

Math/CS Department, University of Cagliari,
Via Ospedale 72, 09124 Cagliari (CA), Italy
ma.atzeni12@studenti.unica.it, atzori@unica.it

Abstract. In this paper, we leverage advances in the Semantic Web area, including data modeling (RDF), data management and querying (JENA and SPARQL), to develop *CodeOntology*, a community-shared software framework supporting expressive queries over source code. The project consists of two main contributions: an ontology that provides a formal representation of object-oriented programming languages, and a parser that is able to analyze Java source code and serialize it into RDF triples. The parser has been successfully applied to the source code of OpenJDK 8, gathering a structured dataset consisting of more than 2 million RDF triples. CodeOntology allows to generate Linked Data from any Java project, thereby enabling the execution of highly expressive queries over source code, by means of a powerful language like SPARQL.

Keywords: Ontology · SPARQL · RDF · OWL · Programming languages

1 Introduction

Nowadays, the online availability of an increasingly large amount of source code is dramatically changing the way programmers approach the development of large software systems. The possibility of reusing existing code allows developers to focus on the added value of their products, speed up the development process and easily explore new possibilities and solutions, while keeping high quality components at the foundations of the system. Several studies have been conducted to understand the general attitude of developers towards the comprehension of large code bases. For instance, in [1] questions asked by programmers during software evolution tasks are analyzed and classified in 44 different categories. The study also highlights the lack of specific methods to answer these questions. Hence, in this paper, we introduce *CodeOntology*, as a resource aimed at supporting the adoption of Semantic Web technologies, in the domain of software development and software engineering. The project has been conceived as an approach to leverage the Semantic Web technology stack and the impressive amount of code available online, to extract structured information from source code, thereby allowing to publish it on the Web in the form of Linked Open Data, as well as enabling the execution of highly expressive queries over source code by means of a powerful language like SPARQL. Thus, CodeOntology is of

C. d'Amato et al. (Eds.): ISWC 2017, Part II, LNCS 10588, pp. 20–28, 2017.
DOI: 10.1007/978-3-319-68204-4_2

particular interest not only to the Semantic Web community, but also to software developers and engineers.

CodeOntology consists of two contributions: an ontology to represent the domain of programming languages and a parser that allows to parse Java source code or bytecode, to serialize it into RDF triples. The ontology is mainly focused towards the Java programming language, but it has been designed with flexibility in mind, thereby being suitable to be reused to represent more languages. On the other hand, the parser is able to extract structural information common to all object-oriented programming languages, like class hierarchy, methods and constructors. Optionally, it can also serialize into RDF triples all the statements and expressions, thereby providing a complete RDF-ization of source code. We then apply semantic techniques like Named Entity Disambiguation, to analyze the comments available within the code and link entities extracted from source code to specific DBpedia [2] resources. This way, it is possible to take advantage of SPARQL to run semantic queries over source code for different purposes, including computer-aided programming, static code analysis, component search and reuse, question answering over source code.

2 Related Work

Querying source code is a critical task in software engineering. Most of the research in software engineering, indeed, is focused towards the development of tools to enhance the maintenance and understanding of extremely large and old software systems. This need has underpinned the development of several source code querying systems, such as OMEGA [3] and CIA (The C Information Abstraction System) [4], which are based on the relational model. More powerful systems, such as Software Refinery [5] are based on graphs and abstract syntax trees. These systems are more sophisticated than tools based on the relational model, but they lack a well-defined query language.

More recently, the online availability of large amounts of open source code has motivated the development of tools to enable programmers to take advantage of this otherwise unstructured information. As an example, Sourcerer [6] allows to collect source code from open repositories and automatically leverage structural information extracted from arbitrary Java projects. The data used by this system, however, are not published on the Web as Linked Open Data, with obvious limitations. Such limitations are partially addressed in [7], where a system to automatically generate an ontology from source code is introduced. This system does not use a unique ontology to describe the entities belonging the programming languages domain, but it generates a different ontology for each input project.

An approach that is more similar to CodeOntology is represented by SCRO (Source Code Representation Ontology) [8]. However, SCRO does not allow to represent some features of modern object-oriented programming languages, such as parameterized types and exceptions handling. Furthermore, the project lacks a system to serialize source code into RDF triples. CodeOntology, unlike

other state-of-the-art systems, makes full use of all the resources made available by the Semantic Web technology stack. Data are extracted from source code according to an appropriate ontology and are published using the RDF data model. The collected information is then available to be queried by means of a highly expressive language like SPARQL. Furthermore, CodeOntology allows to analyze documentation comments and link entities extracted from source code to appropriate DBpedia resources that are semantically associated with these entities.

3 The Ontology

The ontology is written in OWL 2 and has been designed using the Protégé tool [9], according to the design principles of clarity, coherence, extensibility, minimum encoding bias and minimum ontological commitment, introduced in [10]. It consists of 65 classes, 86 object properties and 11 data properties and it has been checked for satisfiability, incoherence and inconsistencies using the HermiT reasoner [11]. The modelling process underlying the creation of the ontology has been guided by common competency questions that usually arise during software process and has been inspired by a re-engineering of the Java abstract syntax, as specified in [12]. However, the ontology is sufficiently general to be extended in order to meet future requirements. For instance, it is possible to reuse the ontology to better represent other programming languages, apart from Java. The IRI associated with the ontology is http://rdf.webofcode.org/woc/, abbreviated as `woc`. The ontology represents structural entities common to all object-oriented programming languages, such as classes, methods, variables, statements and expressions, in a hierarchy of disjoint classes. The root of this hierarchy is the *CodeElement* class, that is the common superclass of all the elements extracted from source code. Since the parser is also able to serialize into RDF triples the structure of Java projects and to analyze libraries such as JAR files, two other classes, namely *Project* and *Library*, have been defined to represent these entities.

The design of the ontology has been conducted according to well-known Ontology Design Patterns, best practices and naming conventions. As an example, the domain of object-oriented programming languages involves large part-whole relations. For instance, a statement may be part of a method, which in turn is part of a class, that is contained in a specified package. In order to represent this partitive relations, the ontology employs a common Content OP and reuses the XKOS vocabulary [13], more precisely the terms `xkos:hasPart` and `xkos:isPartOf`. According to the Transitive Reduction pattern, only the most general property is transitive. Thus, transitivity is delegated to the XKOS vocabulary, which in turn gets transitivity from SKOS[1] and DCMI Metadata Terms[2]. We make use of XKOS because it allows to represent both partitive

[1] https://www.w3.org/TR/skos-reference/.
[2] http://dublincore.org/documents/dcmi-terms/.

(part-whole) and generic (generic-specific) relations. The domain of programming languages, indeed, includes also generic relations between entities. For instance, inheritance in object-oriented programming turns into generic-specific relations between classes. CodeOntology also makes use of other common Ontology Design Patterns and best practices, such as the N-ary relation pattern[3] and the SV (Specified Values) pattern[4] originally introduced by the W3C SWBPD (Semantic Web Best Practices and Deployment) Working Group. They are used in the ontology to model both access modifiers and primitive data types.

The ontology is available at http://doi.org/10.5281/zenodo.577939, under CC BY 4.0 license. Each entity in the ontology has been annotated by means of the `rdfs:comment` and `rdfs:label` properties. A documentation of the ontology, generated using Parrot [14], is available at http://codeontology.org.

4 The Parser

The RDF triple extraction process is managed by the parser, that is the module of CodeOntology that analyzes and parses Java source code to serialize it into RDF triples. As shown in Fig. 1, the RDF serialization of a Java project acts in three steps: first the project is analyzed to download all of its dependencies and load them in class path, then an abstract syntax tree of the source code and its dependencies is built and processed to extract a set of RDF triples.

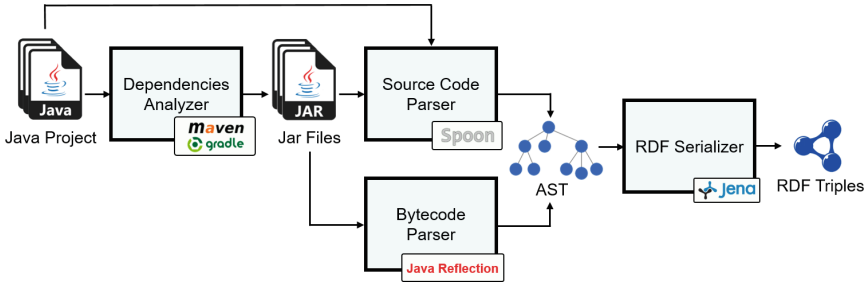

Fig. 1. The RDF serialization process.

CodeOntology currently supports both Maven and Gradle projects. When analyzing a project, the parser first looks at its structure to recognize whether it is built with Maven or Gradle and download the dependencies of the project. JAR files downloaded in this step can optionally be processed and serialized into RDF triples, as well.

Next, the parser builds the abstract syntax tree of the whole input project. This step is handled by Spoon [15], an AST-based source code analysis and

[3] https://www.w3.org/TR/swbp-n-aryRelations/.
[4] https://www.w3.org/TR/swbp-specified-values/.

transformation library, that provides a Java metamodel designed to be easy to understand, query and manipulate. This library is used by CodeOntology to build a model containing information about packages, classes, interfaces, methods, as well as statements, expressions, comments and so on. Spoon allows to define processors to be launched over the abstract syntax tree. The RDF triple extraction is managed by a Spoon processor invoked for every package in the input project. From a particular package, the control flow moves to the types contained in that package, such as classes and interfaces, up to the fields, constructors and methods declared within a specified class. CodeOntology looks then inside the body of each method, to take note of all the referenced types, fields, constructors, methods and variables. The RDF serialization process is handled using Apache Jena[5] and it can optionally involve also all the statements and expressions. The parser also allows to keep track of unstructured information such as comments. We then use TagMe [16] to analyze these comments and automatically link entities extracted from source code to pertinent DBpedia resources.

Beside the processor aimed at walking the abstract syntax tree created by Spoon, CodeOntology actually has three more processors. One of these processors is used to analyze the structure of the input project and serialize it into RDF triples. The second one is used to parse comments and detect Javadoc tags, to extract useful information about parameters and method return values. The last processor is used to analyze JAR files, thereby enabling CodeOntology to run not only on Java source code, but also on bytecode. Given a JAR file, this processor makes use of Java reflection to create an abstract syntax tree that is compliant with the Java metamodel defined by Spoon. The resulting tree is then processed as described above, by means of the main Spoon processor. The parser, along with a tutorial on how to use it to extract a knowledge base from any Java project, is available on GitHub under the GPLv3 license: https://github.com/codeontology/parser.

5 Experiments

The parser has been successfully applied to extract a knowledge base from the OpenJDK 8 source code[6]. These data can be queried through a remote SPARQL endpoint at: http://codeontology.org/sparql. Moreover, the dataset is available at https://doi.org/10.5281/zenodo.818116 and on figshare [17]. The analysis has been conducted on about 1.5 million lines of code, retrieving a total of almost 2M RDF triples falling into 4 categories: structural information on source code (1.9M triples), DBpedia links (309k triples), actual source code as literals (134k triples) and literal comments (105k triples). Quality assessments have been conducted on a sample of methods and classes. Figure 2 shows a small subset of the triples produced by the parser from a simple "hello world" class. This representation allows to run expressive queries over source code, some of which are shown in

[5] https://jena.apache.org/.
[6] http://openjdk.java.net/.

```
package org.codeontology;

public class Example {
  /** Prints a "hello world" message to the standard output */
  public static void main(String[] args) {
    System.out.println("Hello CodeOntology!");
  }
}
```

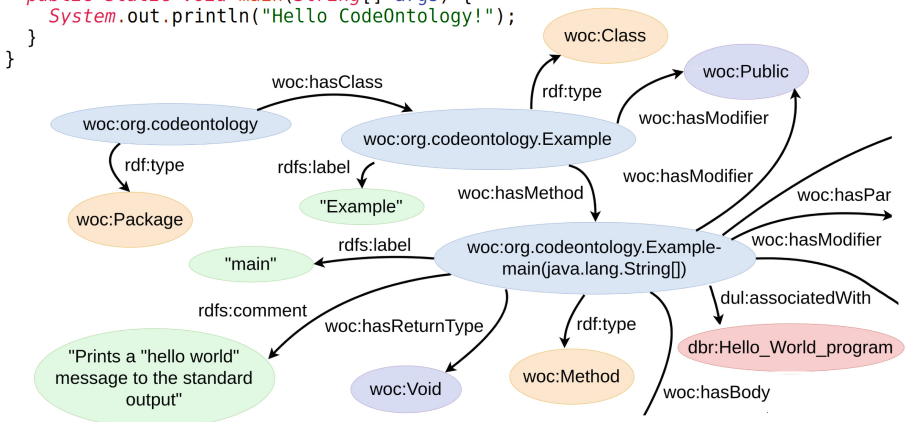

Fig. 2. An excerpt of the RDF serialization produced for a simple "hello world" code.

[18]. As an example, since the output of the parser is a graph, we can easily apply to software not only metrics specifically designed for software engineering tasks, but also metrics borrowed from other fields, such as Social Network Analysis. Suppose we want to rank classes in OpenJDK, to select only the most important ones, according to a specified metric. For instance, we can select the three classes that turn out to be the most referenced by the methods of the other classes. When a method m references a specific class c, then the parser is able to serialize this information into a triple of the form: m *woc:references* c. Thus, we can rank the classes in OpenJDK according to this metric and retrieve the most referenced ones, by means of a simple SPARQL query[7]. Unsurprisingly, the most referenced class in OpenJDK is the `java.lang.String` class, followed by the classes `java.lang.Object` and `java.io.IOException`. In order to compute such a metric efficiently, a graph-based representation of software systems is needed.

Moreover, the extracted DBpedia links can be used to run highly expressive semantic queries over source code. For instance, we can retrieve all the methods for computing the cube root of a real number, by selecting the resources associated with the entity `dbpedia:Cube_root`.

Besides OpenJDK, the system has also been tested on a sample of 20 Java repositories randomly collected from GitHub. Table 1 shows the execution times required to download the dependencies, in the form of JAR files, analyze the source code and process the JAR files previously downloaded. All the times are expressed in seconds. Table 1 also shows the total number of RDF triples extracted from each project.

[7] see http://codeontology.org/examples.

Table 1. Execution times for processing a sample of 20 Java projects and number of RDF triples extracted.

Download	Source code	JAR	Total time	RDF triples
18.5	3.0	0.1	21.6	4336
30.3	–	4.2	34.4	544744
20.2	–	2.5	22.7	344465
15.3	–	0.2	15.5	2496
129.3	21.8	6.0	157.2	626607
94.6	38.3	2.4	135.3	212258
18.2	–	0.1	18.3	2598
0.1	–	1.4	1.6	90071
78.4	–	0.1	78.4	2597
95.6	–	5.3	100.9	505262
258.1	9.9	162.9	430.9	9152328
2950.5	99.7	216.7	3267.1	17059138
171.8	–	0.2	172.0	2499
53.8	–	0.2	54.0	2496
47.0	–	6.9	54.0	561580
121.1	–	2.4	123.4	95376
140.8	93.9	3.5	238.1	171267
78.3	–	0.1	78.6	4992
34.2	–	10.9	45.2	1101273
26.4	–	1.2	27.6	26212
4382.5	266.6	427.3	5076.8	30512595

In some cases, it was not possible to analyze source code because Spoon failed building the Abstract Syntax Tree for different reasons, such as missing dependencies that were not automatically downloaded. However, the parser has been able to extract a knowledge base consisting of more than 30.5 million RDF triples, from only 20 repositories.

6 Conclusions and Future Work

CodeOntology is a project that consists of two contributions: an ontology describing structural entities common to all object-oriented programming languages and a parser capable of serializing Java source code and bytecode into RDF triples. In this paper, we have described the core ideas underlying the design of the ontology and we have analyzed the architecture of the parser. Furthermore, CodeOntology allows to analyze Java comments, in order to link entities extracted from source code to DBpedia resources. This way, it is possible to precisely search specific

software components using expressive semantic queries. In the future, we plan to develop a Question Answering system to hide the complexity of SPARQL queries and allow retrieving software components by means of questions in natural language. Moreover, it will be possible to dereference and execute the source code of the methods in the datasets, using the *Web of Functions* technology [19].

Acknowledgments. This work was supported in part by a 2015 Google Faculty Research Award and Sardegna Ricerche (*project OKgraph*, CRP 120). The authors wish to thank the anonymous reviewers for their insightful comments.

References

1. Sillito, J., Murphy, G.C., De Volder, K.: Questions programmers ask during software evolution tasks. In: Proceedings of the 14th ACM SIGSOFT 2006/FSE-14, pp. 23–34. ACM, New York (2006)
2. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., Bizer, C.: DBpedia - a large-scale, multilingual knowledge base extracted from Wikipedia. Semant. Web J. **6**(2), 167–195 (2015)
3. Linton, M.A.: Implementing relational views of programs. SIGSOFT Softw. Eng. Notes **9**(3), 132–140 (1984)
4. Chen, Y.F., Nishimoto, M.Y., Ramamoorthy, C.V.: The C information abstraction system. IEEE Trans. Softw. Eng. **16**(3), 325–334 (1990)
5. Reasoning Systems: Refine user's guide (1992)
6. Bajracharya, S., Ossher, J., Lopes, C.: Sourcerer: an infrastructure for large-scale collection and analysis of open-source code. Sci. Comput. Program. **79**, 241–259 (2014)
7. Ganapathy, G., Sagayaraj, S.: To generate the ontology from Java source code OWL creation. Int. J. Adv. Comput. Sci. Appl. **2**(2), 111–116 (2011)
8. Alnusair, A., Zhao, T.: Component search and reuse: an ontology-based approach. In: IRI, IEEE Systems, Man, and Cybernetics Society, pp. 258–261 (2010)
9. Musen, M.A.: The protégé project: a look back and a look forward. AI Matters **1**(4), 4–12 (2015)
10. Gruber, T.R.: Toward principles for the design of ontologies used for knowledge sharing. Int. J. Hum.-Comput. Stud. **43**(5–6), 907–928 (1995)
11. Shearer, R., Motik, B., Horrocks, I.: Hermit: a highly-efficient OWL reasoner. In: OWLED, CEUR Workshop Proceedings, vol. 432, CEUR-WS.org (2008)
12. Gosling, J., Joy, B., Steele, G., Bracha, G., Buckley, A.: The Java Language Specification, Java SE 8 edn. Oracle (2015)
13. Gillman, D., Cotton, F., Jaques, Y.: XKOS: extending SKOS for describing statistical classifications. In: The 12th International Semantic Web Conference (2013)
14. Tejo-Alonso, C., Berrueta, D., Polo, L., Fernández, S.: Metadata for web ontologies and rules: current practices and perspectives. In: García-Barriocanal, E., Cebeci, Z., Okur, M.C., Öztürk, A. (eds.) MTSR 2011. CCIS, vol. 240, pp. 56–67. Springer, Heidelberg (2011). doi:10.1007/978-3-642-24731-6_6
15. Pawlak, R., Monperrus, M., Petitprez, N., Noguera, C., Seinturier, L.: SPOON: a library for implementing analyses and transformations of Java source code. Softw.: Pract. Exp. **46**, 1155–1179 (2016). http://onlinelibrary.wiley.com/doi/10.1002/spe.2346/full

16. Ferragina, P., Scaiella, U.: Fast and accurate annotation of short texts with Wikipedia pages. IEEE Softw. **29**(1), 70–75 (2012)
17. Atzeni, M., Atzori, M.: CodeOntology OpenJDK8 dataset. figshare (2017). https://doi.org/10.6084/m9.figshare.5234878
18. Atzeni, M., Atzori, M.: CodeOntology: querying source code in a semantic framework. In: Proceedings of the ISWC 2017 Posters & Demonstrations Track co-located with 16th International Semantic Web Conference (ISWC 2017)
19. Atzori, M.: Toward the web of functions: interoperable higher-order functions in SPARQL. In: Mika, P., et al. (eds.) ISWC 2014. LNCS, vol. 8797, pp. 406–421. Springer, Cham (2014). doi:10.1007/978-3-319-11915-1_26

http://www.springer.com/978-3-319-68203-7

The Semantic Web – ISWC 2017
16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part II
d'Amato, C.; Fernández, M.; Tamma, V.; Lecue, F.; Cudré-Mauroux, P.; Sequeda, J.; Lange, C.; Heflin, J. (Eds.)
2017, XLVI, 388 p. 103 illus., Softcover
ISBN: 978-3-319-68203-7