CodeOntology
RDF-ization of Source Code

Mattia Atzeni and Maurizio Atzori

University of Cagliari
Introduction

• Code reuse, code querying and computer-aided programming are some of the main research challenges in software engineering.

• Recent research is focusing towards **graph-based approaches** to model both software and software process architecture.
Introduction

Every day 6000 new developers push code on GitHub.

More than 2.8M users created their first repo within the last year.

S. Keskkula, “What is this GitHub you speak of?”
Introduction

* The online availability of an increasingly **large amount of open source code** is dramatically changing the way programmers approach the development of large software systems.
The Semantic Web technology stack provides expressive standards to represent and query structured information.
CodeOntology consists of two main contributions:

• an **ontology** providing a formal representation of object-oriented programming languages;

• a **parser** capable of analyzing both Java source code and bytecode to generate RDF triples.
CodeOntology allows to run highly expressive **SPARQL queries** over source code for different purposes, such as:

- Static code analysis
- Component search
- Question Answering over source code
• The ontology is available on Zenodo under CC BY 4.0 license.
• The parser is available on GitHub under GPLv3 license.
• A RDF dataset extracted from OpenJDK 8 is available on Zenodo under CC BY 4.0 license.
The ontology is mainly focused towards the Java programming language. It is written in OWL 2 and has been designed using the protégé resource.
The Ontology

• The modeling process underlying the creation of the ontology has been inspired by:
  • common **competency questions** that usually arise during software evolutions tasks
  • a **reengineering of Java abstract syntax**.
### The Ontology

<table>
<thead>
<tr>
<th>Category</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>65</td>
</tr>
<tr>
<td>Object properties</td>
<td>86</td>
</tr>
<tr>
<td>Data properties</td>
<td>11</td>
</tr>
<tr>
<td>Axioms</td>
<td>1097</td>
</tr>
</tbody>
</table>
The ontology represents structural information common to all object oriented programming languages in a strict hierarchy of disjoint classes.

The root of this hierarchy is the `CodeElement` class.
The Ontology
The Ontology

• Part-whole relations are represented by employing a common Content OP and by reusing the XKOS vocabulary, in particular the terms \texttt{xkos:hasPart} and \texttt{xkos:isPartOf}.

• According to the Transitive Reduction pattern, transitivity is delegated to XKOS, which in turn gets transitivity from SKOS and DCMI Metadata Terms.
The Ontology

CodeOntology reuses also other common ontology design patterns such as:

• N-ary Relation Pattern
• SV (Specified Values) Pattern, originally introduced by the W3C SWBPD (Semantic Web Best Practices and Deployment) Working Group.
Classes


**Access Modifier (OWL Class)**

An access modifier

**Definition**

The URI of this class is [http://rdf.webofcode.org/woc/AccessModifier](http://rdf.webofcode.org/woc/AccessModifier)
The RDF serialization of a Java project acts in three steps:

1. The project is analyzed to download its dependencies
2. An abstract syntax tree of the project is built
3. The AST is processed and serialized into RDF triples
The Parser

Download of the dependencies of the project

We currently support both Maven and Gradle projects. During this step, all of the dependencies of the project are downloaded in form of JAR files and loaded in classpath.

maven  gradle
Abstract Syntax Tree

Once all the dependencies of the project have been loaded, an abstract syntax tree of the project is built by using Spoon, a source code analysis and transformation library.
The abstract syntax tree generated by spoon is processed and serialized into RDF triples. This step is handled using Apache Jena.
The parser can also exploit **Java Reflection** to process JAR files, thereby enabling CodeOntology to run not only on source code but also on **bytecode**.
• CodeOntology makes use of TagMe to perform Named Entity Disambiguation on the comments provided within the code and link entities extracted from source code to pertinent DBpedia resources.
The Parser

- CodeOntology allows also to extract structured information from Javadoc comments.

```java
/**
 * Tests if this string ends with the specified suffix.
 * @param suffix the suffix.
 */
public boolean endsWith(String suffix) {
    return startsWith(suffix, value.length - suffix.length);
}
```
The Parser

Java Project → Dependencies Analyzer → Jar Files → Source Code Parser → Bytecode Parser → RDF Serializer → RDF Triples
Example

```java
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");
    }
}
```
Experiments

- CodeOntology has been applied on OpenJDK 8, gathering a structured dataset consisting of more than 2.5M RDF triples.

<table>
<thead>
<tr>
<th>Category</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structural information</td>
<td>1981108 triples</td>
</tr>
<tr>
<td>DBpedia links</td>
<td>309688 triples</td>
</tr>
<tr>
<td>Source code as literals</td>
<td>134757 triples</td>
</tr>
<tr>
<td>Literal Comments</td>
<td>105881 triples</td>
</tr>
</tbody>
</table>

- Structural Information: 78.26%
- DBpedia Links: 12.23%
- Source Code as Literals: 5.32%
- Comments: 4.18%
<table>
<thead>
<tr>
<th>Repository</th>
<th>Download (s)</th>
<th>Spoon (s)</th>
<th>Source code (s)</th>
<th>Jar files (s)</th>
<th>Total time (s)</th>
<th>Triples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18,524</td>
<td>2,091</td>
<td>0,926</td>
<td>0,091</td>
<td>21,632</td>
<td>4336</td>
</tr>
<tr>
<td>2</td>
<td>30,289</td>
<td>-</td>
<td>-</td>
<td>4,181</td>
<td>34,47</td>
<td>544744</td>
</tr>
<tr>
<td>3</td>
<td>20,154</td>
<td>-</td>
<td>-</td>
<td>2,546</td>
<td>22,7</td>
<td>344465</td>
</tr>
<tr>
<td>4</td>
<td>15,326</td>
<td>-</td>
<td>-</td>
<td>0,188</td>
<td>15,514</td>
<td>2496</td>
</tr>
<tr>
<td>5</td>
<td>129,321</td>
<td>4,917</td>
<td>16,915</td>
<td>6,048</td>
<td>157,201</td>
<td>626607</td>
</tr>
<tr>
<td>6</td>
<td>94,562</td>
<td>4,023</td>
<td>34,236</td>
<td>2,454</td>
<td>135,275</td>
<td>212258</td>
</tr>
<tr>
<td>7</td>
<td>18,236</td>
<td>-</td>
<td>-</td>
<td>0,128</td>
<td>18,364</td>
<td>2598</td>
</tr>
<tr>
<td>8</td>
<td>0,136</td>
<td>-</td>
<td>-</td>
<td>1,426</td>
<td>1,562</td>
<td>90071</td>
</tr>
<tr>
<td>9</td>
<td>78,365</td>
<td>-</td>
<td>-</td>
<td>0,123</td>
<td>78,488</td>
<td>2597</td>
</tr>
<tr>
<td>10</td>
<td>95,567</td>
<td>-</td>
<td>-</td>
<td>5,329</td>
<td>100,896</td>
<td>505262</td>
</tr>
<tr>
<td>11</td>
<td>258,127</td>
<td>3,903</td>
<td>5,981</td>
<td>162,936</td>
<td>430,947</td>
<td>9152328</td>
</tr>
<tr>
<td>12</td>
<td>2950,463</td>
<td>5,362</td>
<td>94,613</td>
<td>216,653</td>
<td>3267,091</td>
<td>17059138</td>
</tr>
<tr>
<td>13</td>
<td>171,782</td>
<td>-</td>
<td>-</td>
<td>0,193</td>
<td>171,975</td>
<td>2499</td>
</tr>
<tr>
<td>14</td>
<td>53,836</td>
<td>-</td>
<td>-</td>
<td>0,205</td>
<td>54,041</td>
<td>2496</td>
</tr>
<tr>
<td>15</td>
<td>47,007</td>
<td>-</td>
<td>-</td>
<td>6,947</td>
<td>53,984</td>
<td>561580</td>
</tr>
<tr>
<td>16</td>
<td>121,072</td>
<td>-</td>
<td>-</td>
<td>2,365</td>
<td>123,437</td>
<td>95376</td>
</tr>
<tr>
<td>17</td>
<td>140,788</td>
<td>5,216</td>
<td>88,672</td>
<td>3,462</td>
<td>238,138</td>
<td>171267</td>
</tr>
<tr>
<td>18</td>
<td>78,383</td>
<td>-</td>
<td>-</td>
<td>0,133</td>
<td>78,516</td>
<td>4992</td>
</tr>
<tr>
<td>19</td>
<td>34,247</td>
<td>-</td>
<td>-</td>
<td>10,908</td>
<td>45,155</td>
<td>1101273</td>
</tr>
<tr>
<td>20</td>
<td>26,406</td>
<td>-</td>
<td>-</td>
<td>1,208</td>
<td>27,614</td>
<td>26212</td>
</tr>
<tr>
<td><strong>Sum</strong></td>
<td><strong>4382,591</strong></td>
<td><strong>25,512</strong></td>
<td><strong>241,343</strong></td>
<td><strong>427,524</strong></td>
<td><strong>5076,97</strong></td>
<td><strong>30512595</strong></td>
</tr>
<tr>
<td><strong>Avg</strong></td>
<td><strong>219,12955</strong></td>
<td><strong>4,252</strong></td>
<td><strong>40,22383333</strong></td>
<td><strong>21,3762</strong></td>
<td><strong>253,8485</strong></td>
<td><strong>1525629,75</strong></td>
</tr>
<tr>
<td><strong>Std. Dev.</strong></td>
<td><strong>646,1135323</strong></td>
<td><strong>1,220474006</strong></td>
<td><strong>41,47638727</strong></td>
<td><strong>58,3236317</strong></td>
<td><strong>716,322511</strong></td>
<td><strong>4173300,06</strong></td>
</tr>
</tbody>
</table>
• Select the most referenced classes

```
SELECT ?class
  (COUNT(DISTINCT ?anotherClass) AS ?count)
WHERE {
  ?class a woc:Class .
  ?method a woc:Method ;
  woc:isDeclaredBy ?anotherClass ;
  woc:references ?class .
  FILTER (?class != ?anotherClass)
}
GROUP BY ?class
ORDER BY DESC(?count)
LIMIT 3
```

<table>
<thead>
<tr>
<th>?class</th>
<th>?count</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String</td>
<td>2407</td>
</tr>
<tr>
<td>java.lang.Object</td>
<td>1458</td>
</tr>
<tr>
<td>java.io.IOException</td>
<td>735</td>
</tr>
</tbody>
</table>
• Select the methods computing the cube root of a real number

```sparql
SELECT ?method
WHERE {
  ?method a woc:Method ;
  woc:hasParameter/woc:hasType woc:Double ;
  dul:associatedWith db:Cube_root .
}

?method
java.lang.Math.cbrt(double)
java.lang.StrictMath.cbrt(double)
```
Query examples

Select recursive methods

```
SELECT ?method
WHERE {
  ?method a woc:Method ;
  woc:references+ ?method .
}
LIMIT 20
```

Get the most referenced classes

```
SELECT ?class (COUNT(DISTINCT ?anotherClass) as ?count)
WHERE {
  ?class a woc:Class .
  ?method a woc:Method ;
  woc:isDeclaredBy ?anotherClass ;
  woc:references ?class .
  FILTER (?class != ?anotherClass)
}
GROUP BY ?class
ORDER BY DESC(?count)
LIMIT 3
```

Sort classes by the number of subclasses

```
SELECT ?superClass (COUNT(DISTINCT ?class) as ?count)
WHERE {
  ?class a woc:Class;
  woc:extends* ?superClass .
}
GROUP BY ?superClass
ORDER BY DESC(?count)
LIMIT 3
```

Singletons

```
SELECT DISTINCT ?class
WHERE {
  ?class woc:hasConstructor ?constructor ;
  ?class a woc:Singleton .
}
LIMIT 10
```
Run a query in this box.  
Check the documentation and the examples for more details.

```
1. PREFIX owl:<http://www.w3.org/2002/07/owl#>
2. PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3. PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
4. PREFIX woc:<http://rdfs.webofcode.org/woc/> 
5. PREFIX dbr:<http://dbpedia.org/resource/>
6. PREFIX dul:<http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#>

9. # Get all methods in OpenJDK & computing the cube root of a parameter of type double
   10. SELECT ?method
   11.   WHERE {
   12.     ?method a woc:Method ;
   13.       woc:hasParameter woc:hasType woc:Double ;
   15. }
```

Query  Clear
Future Work

As future work, we plan to:

• extend the project to other languages, such as Python or JavaScript

• develop a question answering system to hide the complexity of SPARQL queries and allow the user to retrieve software components by means of questions in natural language
Conclusions

CodeOntology allows to run expressive SPARQL queries over source code. The project has been tested on OpenJDK 8, gathering a dataset consisting of more than 2.5M triples. The dataset and the ontology are available on Zenodo, while the parser is available on GitHub.

For more details see http://codeontology.org.
Questions
Thank You

Mattia Atzeni and Maurizio Atzori