Source Code Plagiarism Detection Method Using Protégé Built Ontologies

Ion SMEUREANU, Bogdan IANCU
The Bucharest University of Economic Studies
ion.smeureanu@csie.ase.ro, bogdan.iancu@ymail.com

Software plagiarism is a growing and serious problem that affects computer science universities in particular and the quality of education in general. More and more students tend to copy their thesis's software from older theses or internet databases. Checking source codes manually, to detect if they are similar or the same, is a laborious and time consuming job, maybe even impossible due to existence of large digital repositories. Ontology is a way of describing a document's semantic, so it can be easily used for source code files too. OWL Web Ontology Language could find its applicability in describing both vocabulary and taxonomy of a programming language source code. SPARQL is a query language based on SQL that extracts saved or deducted information from ontologies. Our paper proposes a source code plagiarism detection method, based on ontologies created using Protégé editor, which can be applied in scanning students' theses' software source code.
Keywords: *Ontology, OWL, SPARQL, Plagiarism, Protégé*

1 Introduction

In our days we have a huge volume of digital information, thing that can be very useful on one side, but a disadvantage on the other. The useful part is that we can find any needed information more quickly (at a click of a button as we usually say) than in the past by taking advantage of the digital repositories. The disadvantage is that finding similar or duplicated documents is very difficult now, especially when this job is made manually. That is why we try to find alternative solutions in the field of plagiarism detection systems [1].

The term "ontology" is inherited from philosophy where it refers to existence and the things that exist. In computer science those things are represented by data and the ontology generally describes the semantic of terms used in a specific domain (in our case programming), providing a vocabulary for that domain as well as a computerized specification of the meaning of terms used in the vocabulary. Ontologies range from taxonomies and classifications, database schemas, to fully axiomatized theories. In recent years, ontologies have been adopted in many business and scientific communities as a way to share, reuse and process domain knowledge. Ontologies are now central to many applications such as scientific knowledge portals, information management and integration systems, electronic commerce, and semantic web services [2]. In our work we will use ontologies for building the knowledge graph specific to each source code that we suspect of plagiarism.

OWL Web Ontology Language is a specification by the World Wide Web Consortium (W3C) and serves as a fundamental component of the Semantic Web initiative. OWL is based upon the Extensible Markup Language (XML), XML Schema [3], the Resource Description Framework (RDF) and RDF Schema (RDF-S) [4]. It is composed from three sublanguages OWL-Lite, OWL-DL and OWL-Full, from those OWL-DL being the one most often used because it provides maximum expressiveness.

The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web. It is particularly intended for representing metadata about web resources, such as the title, author, and modification date of a web page, copyright and licensing information about a web document, or the availability schedule for some shared resource [4]. However, by generalizing the concept of a web resource, RDF can also be used to represent information about things that can

DOI: 10.12948/issn14531305/17.3.2013.07

be identified on the web, even when they cannot be directly retrieved on the web.
RDF is intended for situations in which this information needs to be processed by applications, rather than being only displayed to people. RDF provides a common framework for expressing this information so it can be exchanged between applications without loss of meaning. Since it is a common framework, application designers can leverage the availability of common RDF parsers and processing tools. The ability to exchange information between different applications means that the information may be made available to applications other than those for which it was originally created.
We will use RDF and OWL in our method as standards and formats for saving the ontologies created via the Protégé editor. We prefer this approach because they are W3C standards and in this way we can provide interoperability between our work and other future related works.
Protégé is a free, open source ontology editor and knowledge-base framework that provides a suite of tools to construct domain models and knowledge-based applications with ontologies. At its core, Protégé implements a rich set of knowledge-modeling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats. Protégé can be customized to provide domain-friendly support for creating knowledge models and entering data [2].
SPARQL for RDF [5] is a query language that can be used to retrieve information across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. SPARQL also supports extensible value testing and constraining queries by source RDF graph. The results of SPARQL queries can be results sets or RDF graphs.

2 Proposed Method

Our method is a step by step algorithm build with the help of the Protégé editor, version 4.3.0.
The Protégé platform supports two main ways of modeling ontologies:

- the *Protégé-Frames* editor enables users to build and populate ontologies that are frame-based, in accordance with the Open Knowledge Base Connectivity protocol (OKBC). In this model, an ontology consists of a set of classes organized in a subsumption hierarchy to represent a domain's salient concepts, a set of slots associated to classes to describe their properties and relationships, and a set of instances of those classes - individual exemplars of the concepts that hold specific values for their properties;
- the *Protégé-OWL* editor enables users to build ontologies for the *Semantic Web*, in particular in the W3C's Web Ontology Language (OWL). An OWL ontology may include descriptions of classes, properties and their instances. Given such an ontology, the OWL formal semantics specifies how to derive its logical consequences, i.e. facts not literally present in the ontology, but entailed by the semantics. These entailments may be based on a single document or multiple distributed documents that have been combined using defined OWL mechanisms.

As we have already stated, we will choose the second way of modeling ontologies provided by Protégé and we will create W3C's OWL based ontologies.
The first step in the development of the source code plagiarism detection system is building the needed OWL Classes [6]. This approach is similar to the OOP paradigm [7]. We will implement classes like *Variable, Constant, DataType, ProgrammingStructure, Comment, SystemFunction and Operator*. The classes created within the editor are visible in Figure 1.

DOI: 10.12948/issn14531305/17.3.2013.07

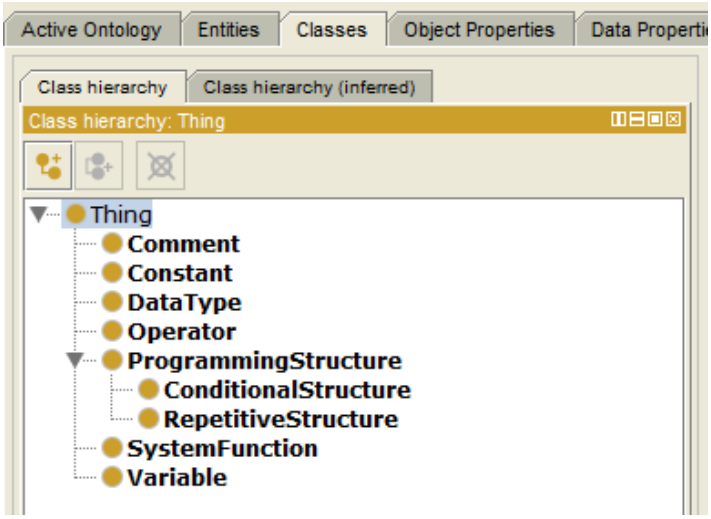

Fig. 1. OWL Classes

The OWL syntax specific to this classes is:

```
<owl:Class rdf:about="&untitled-ontology-4;Comment"/>
<owl:Class rdf:about="&untitled-ontology-4;Constant"/>
<owl:Class rdf:about="&untitled-ontology-4;DataType"/>
<owl:Class rdf:about="&untitled-ontology-4;Operator"/>
<owl:Class rdf:about="&untitled-ontology-4;ProgrammingStructure"/>
<owl:Class rdf:about="&untitled-ontology-4;SystemFunction"/>
<owl:Class rdf:about="&untitled-ontology-4;Variable"/>
```

We can also define specialized concepts that can therefore be used to build taxonomies. This is the case of *RepetitiveStructure* and *ConditionalStructure* from Figure 1. They are defined as special programming structures (subclasses of *ProgrammingStructure*).
The correspondent OWL syntax for this is:

```
<owl:Class rdf:about="&untitled-ontology-4;ConditionalStructure">
    <rdfs:subClassOf rdf:resource="&untitled-ontology-4;ProgrammingStructure"/>
</owl:Class>
<owl:Class rdf:about="&untitled-ontology-4;RepetitiveStructure">
    <rdfs:subClassOf rdf:resource="&untitled-ontology-4;ProgrammingStructure"/>
</owl:Class>
```

To define relations between the modeled concepts we use *ObjectProperty.* These relations can be marked as transitive, symmetrical or functional. Two relations can be marked as inverse to each other. Furthermore relations can be specialized by using *subPropertyOf* in analogy to *subClassOf* for concepts. The following example, shown in Figure 2, defines the relation *conditions* with the concept *ConditionalStructure* as domain and the concept *RepetitiveStructure* as range. It is inverse to another relation called *has_condition*.

DOI: 10.12948/issn14531305/17.3.2013.07

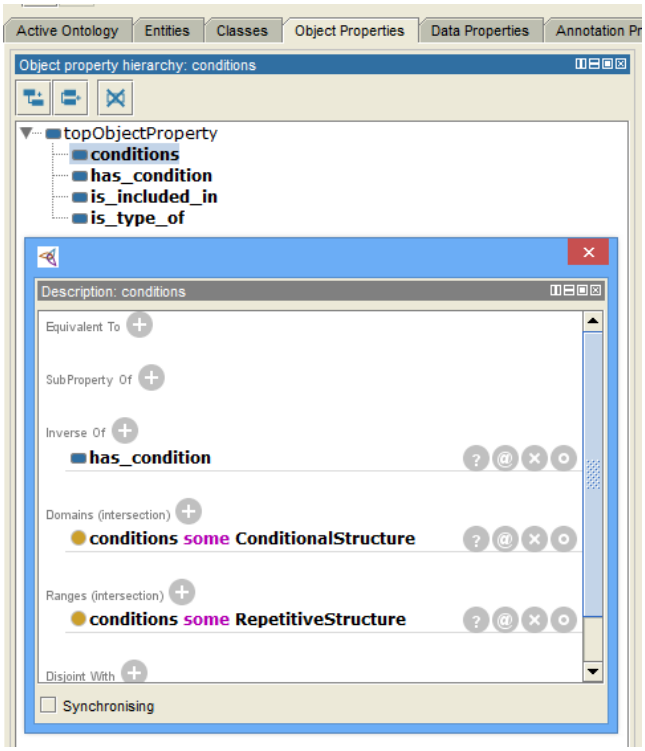

Fig. 2. Object Properties

OWL syntax for the new concepts is:

```
    <owl:ObjectProperty rdf:about="&untitled-ontology-4;conditions">
        <rdfs:domain>
            <owl:Restriction>
                <owl:onProperty rdf:resource="&untitled-ontology-4;conditions"/>
                <owl:someValuesFrom rdf:resource="&untitled-ontology-
4;ConditionalStructure"/>
            </owl:Restriction>
        </rdfs:domain>
        <rdfs:range>
            <owl:Restriction>
                <owl:onProperty rdf:resource="&untitled-ontology-4;conditions"/>
                <owl:someValuesFrom rdf:resource="&untitled-ontology-
4;RepetitiveStructure"/>
            </owl:Restriction>
        </rdfs:range>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:about="&untitled-ontology-4;has_condition">
        <owl:inverseOf rdf:resource="&untitled-ontology-4;conditions"/>
    </owl:ObjectProperty>
```

Other defined relations in our ontology are *is_included_in* (which is marked as transitive) and *is_type_of*. We could limit their domain and range as well, to *ProgrammingStructure* or *Variable* and *DataType*. The correspondent OWL syntax for them is:

```
    <owl:ObjectProperty rdf:about="&untitled-ontology-4;is_included_in">
        <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:about="&untitled-ontology-4;is_type_of"/>
```

The next step is to define facts upon the previously defined concepts, attributes and relations by instancing them. These instances are called individuals (similar to the OOP

DOI: 10.12948/issn14531305/17.3.2013.07

concept of object [7]). The following example (Figure 3) states that *For* and *While* are repetitive structures.

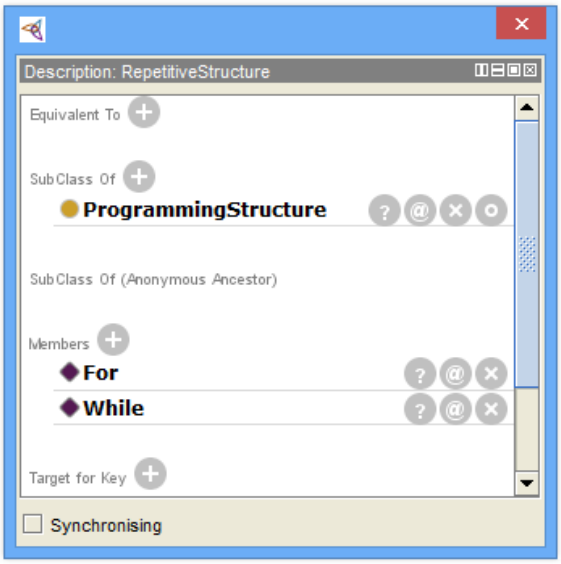

Fig. 3. Individuals

The specific OWL syntax for individuals is:

```
<owl:NamedIndividual rdf:about="&untitled-ontology-4;While">
    <rdf:type rdf:resource="&untitled-ontology-4;RepetitiveStructure"/>
</owl:NamedIndividual>
<owl:NamedIndividual rdf:about="&untitled-ontology-4;For">
    <rdf:type rdf:resource="&untitled-ontology-4;RepetitiveStructure"/>
</owl:NamedIndividual>
```

We will do the same for the rest of individuals found in the investigated source code. In this way we create an ontology for each source code that is suspect of plagiarism. We do this thing manually using Protégé just for demonstration purposes only. This process can be made automatically by building a crawler that reads the source code and builds the OWL file correspondent to it [8]. The crawler will receive as input the raw source code and will return as output the OWL file corresponding to the built ontology. In this way we will have an ontology file for each source code no matter of the programming language in which it is written.

We choose as an example the following source code written in C:

```
int option = 0;
int i;
int numbers[3];
while (option!=3)
{
	printf("Please choose an option and press enter:\n");
	printf("1. Read 3 numbers\n 2. Print the max\n 3.Exit\n");
	scanf("%i",&option);
	if (option==1)
	{
		for (i=0; i<3; i++)
		{
			printf("\nnumbers[%i]=",i+1);scanf("%i",&numbers[i]);
		}
	}
	if (option==2)
	{
```

DOI: 10.12948/issn14531305/17.3.2013.07

```
                int max = 0;
                for (i=0; i<3; i++)
                {
                        if(numbers[i] > max)
                        {
                                max = numbers[i];
                        }
                }
                printf("\nMax=%i",max);
        }
}
```

The code displays a menu which has three items: *Read 3 numbers*, *Print the max* and *Exit*. Based upon the value that the user provides the program reads three integer numbers, computes the maximum from the three of them or interrupts its execution.

The process of creating the ontology for the current example (process that could be made automatically by a crawler) is explained in the following paragraph.

The crawler will read the code line by line from top to bottom and will create the specific individuals for each line of code. For example, for the first three lines of code we have three individuals of type *Variable* that will have their object property *is_type_of* set to individuals of type *DataType* called *int* and *array*. On the following lines we have an individual of type *ProgrammingStructure* with three individuals of type *SystemFunctions* included in it (object property *is_included_in*). The same rules apply to the next lines of code until we finish parsing all the source code. To keep this example as simple as possible we will not use properties for each condition of the conditional or repetitive programming structures and we will name the individuals based on their pseudocode name [9] and the number of apparitions (e.g. *for, for2, if, if2, while,* etc).

And in comparison we will take the following code written in Javascript that do the same thing. The differences are caused only by the different syntaxes of the two languages:

```
var option = 0;
var i=0;
var numbers=new Array();
while(option!=3)
{
        document.write("Please choose an option and press enter:\n");
        document.write("1. Read 3 numbers\n 2. Print the max\n 3.Exit\n");
        option = prompt("Option");
                if (option==1)
                {
                        for (i=0; i<3; i++)
                        {
                                numbers[i] = prompt("numbers[" + (i+1) + "]");
                        }
                }
                if (option==2)
                {
                        var max = 0;
                        for (i=0; i<3; i++)
                        {
                                if(numbers[i] > max)
                                {
                                        max = numbers[i];
                                }
                        }
                        document.write("\nMax=" + max + "\n");
                }
}
```

Similar to what we have done before we create an ontology for the Javascript source

DOI: 10.12948/issn14531305/17.3.2013.07

code too.
The next step, and the final one, in our proposed method is to find a way of comparing the two ontologies obtained from the presented process.
A solution is to take advantage of the fact that OWL ontologies are based on RDF and build different SPARQL queries for comparing the source codes. The queries will depend on the algorithms that we want to test. For example we can choose some metrics that will be measured using SPARQL and then compared to see the plagiarism degree.

SPARQL is the standard query language for accessing RDF data [10], where the basic access pattern is called the triple pattern. A triple pattern has the same form as an RDF triple, but with variables. Like the counterpart of select-project-join queries in SQL, the SPARQL query supports both conjunctions and disjunctions of the triple patterns. Furthermore, the predicates in the SPARQL query can also be variables, which allow "predicate-agnostic" queries.
Protégé editor provides us an user interface where we can run SPARQL queries (as shown in Figure 4), but limits our output to results sets.

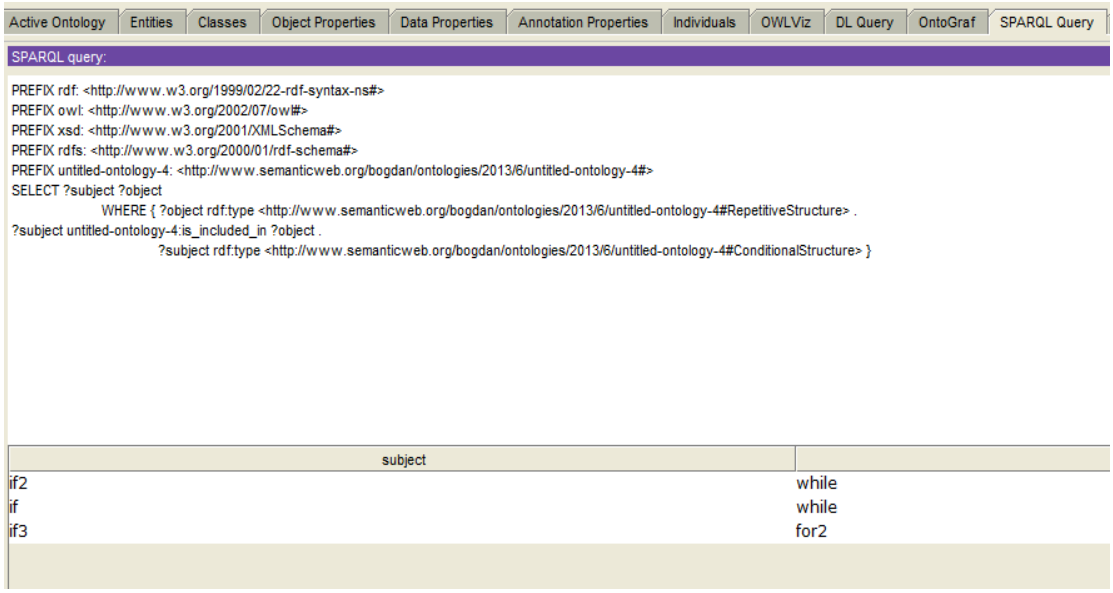

Fig. 4. Protégé SPARQL Query Editor

We will define ten metrics [11] (presented in Table 1) that will be measured for each ontology apart. Based on these metrics we will compute a plagiarism degree.

Table 1. Metrics

No	Metric	SPARQL Query	Result on C code[1]	Result on Javascript code[2]	Percentage of similarity (min(R[1], R[2]) /max(R[1], R[2]) * 100)
1.	Total number of conditional structures	`SELECT ?subject WHERE { ?subject rdf:type <http://ontology_uri#ConditionalStructure>}`	3	3	100%
2.	Total number of repetitive structures	`SELECT ?subject WHERE { ?subject rdf:type <http://ontology_uri#RepetitiveStructure>}`	3	3	100%

DOI: 10.12948/issn14531305/17.3.2013.07

3.	Total number of variables	`SELECT ?subject WHERE { ?subject rdf:type <http://ontology_uri#Variable>}`	4	4	100%
4.	Total number of conditional structures included in repetitive structures	`SELECT ?subject ?object WHERE { ?object rdf:type <http://ontology_uri #RepetitiveStructure> . ?subject untitled-ontology-4:is_included_in ?object . ?subject rdf:type <http:// ontology_uri#ConditionalStructure> }`	3	3	100%
5.	Total number of repetitive structures included in repetitive structures	`SELECT ?subject ?object WHERE { ?object rdf:type <http://ontology_uri #RepetitiveStructure> . ?subject untitled-ontology-4:is_included_in ?object . ?subject rdf:type <http://ontology_uri #RepetitiveStructure> }`	0	0	100%
6.	Total number of system functions called	`SELECT ?subject WHERE { ?subject rdf:type <http://ontology_uri#SystemFunction>}`	5	4	80%
7.	Total number of system functions called in conditional structures	`SELECT ?subject ?object WHERE { ?object rdf:type <http://ontology_uri#ConditionalStructure> . ?subject untitled-ontology-4:is_included_in ?object . ?subject rdf:type <http://ontology_uri#SystemFunction> }`	1	1	100%
8.	Total number of system functions called in repetitve structures	`SELECT ?subject ?object WHERE { ?object rdf:type <http://ontology_uri#RepetitiveStructure> . ?subject untitled-ontology-4:is_included_in ?object . ?subject rdf:type <http://ontology_uri#SystemFunction> }`	4	3	75%
9.	Total number of data types used	`SELECT ?subject WHERE { ?subject rdf:type <http://ontology_uri#DataType>}`	2	2	100%
10.	Total number of variable of type array	`SELECT ?subject WHERE { ?subject rdf:type <http://ontology_uri#Variable> .`	1	1	100%

DOI: 10.12948/issn14531305/17.3.2013.07

		`?subject untitled-ontology-4:is_type_of <http://ontology_uri#array> }`			
Total plagiarism degree (arithmetic mean of metrics)					**95.5 %**

As we can see this method is precise enough for determining the plagiarism degree, but it depends very much on the chosen metrics. So if we choose to make a software application for this, it is a better approach if the final user will have the possibility to choose the interest metrics and how they influence the final result (in our case we consider them equal in influencing the final result). Because this method is not as accurate as we wish we have searched for alternative solutions in the field of ontologies to confirm the result obtained in this way.

So we found that another method of comparing two source codes ontologies is by using the graphical representation of the semantic networks. The semantic network (called in some cases concept network) is a graph, where the nodes represent concepts and the arcs represent the relations between the concepts [12].

Most semantic networks are cognitively based. They also consist of arcs and nodes which can be organized into a taxonomic hierarchy. Semantic networks contributed ideas of spreading activation, inheritance, and nodes as proto-objects. They are intractable for large domains.

Some properties are not easily expressed using a semantic network, e.g., negation, disjunction, and general non-taxonomic knowledge. Expressing these relationships requires workarounds, such as having complementary predicates and using specialized procedures to check for them, but this was not a problem in our method.

A particular case of a semantic network representation is the topic map. The Topic Maps family of standards is designed to facilitate the gathering of all the information about a subject at a single location. The information about a subject includes its relationships to other subjects; such relationships may also be treated as subjects (subject-centric) [13].

These visual representations of ontologies can help in our method. Topic models (which can be viewed as the Bayesian version of latent semantic analysis) are useful for extracting semantic content from any type of collections. After topic modeling, the topic representation is projected onto two dimensions to create the topic map visualization [14].

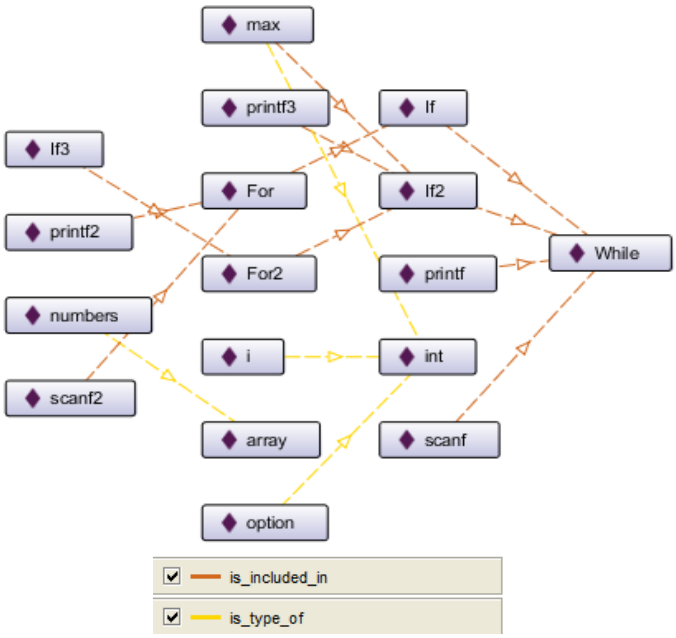

Fig. 5. C Ontology - OntoGraph representation of Individuals

DOI: 10.12948/issn14531305/17.3.2013.07

In our proposed algorithm we will use OntoGraph representation of Protégé to visually compare the two ontologies. In this representation the nodes are individuals (represented by a rectangle with a violet diamond) and the arcs are relations between them (with orange the *is_included_in* relation and with yellow the *is_type_of* relation). The arcs are oriented and show the direction of the relation. We preferred the tree horizontal view because our *is_included_in* and *is_type_of* relations are hierarchical.

The first OntoGraph (shown in Figure 5) is the representation of the C ontology with its specific individuals. The second one (shown in Figure 6) is the Javascript ontology. We can see that this one has another set of individuals.

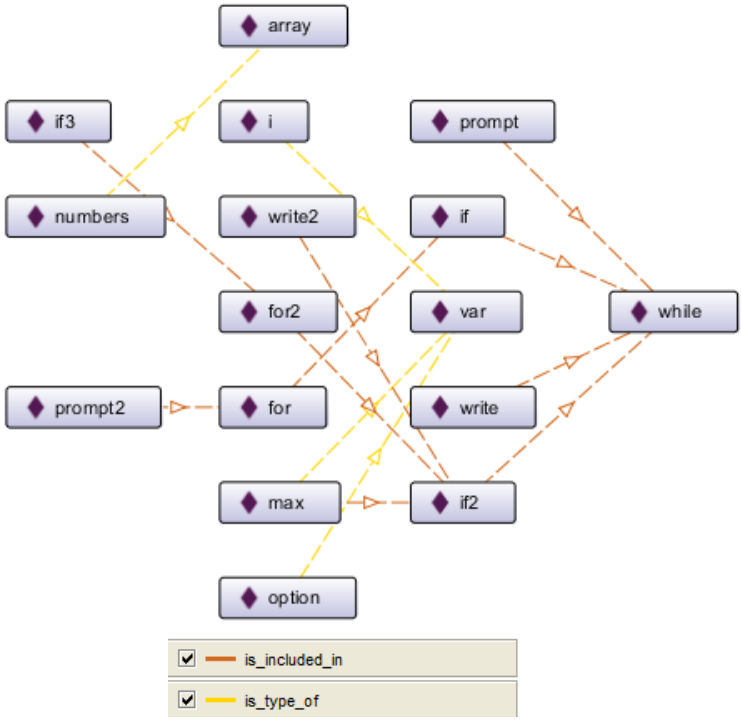

Fig. 6. Javascript Ontology - OntoGraph representation of Individuals

To compare the two Topic Maps we can look at them separately or we can create a new topic map (eventually a plotted one) that represents both source code ontologies in a single graph, by using an existing visualization tool or by creating a specific one [15].

In our example the similarities between the two source codes are obvious even in the separated representations, so we can conclude that the tested source codes are copied one from another.

Combined, the two forms of plagiarism detection solutions based on ontologies (metrics measured with SPARQL and Topic Maps) can be a very powerful and useful way of determining if two source codes are similar and in which percentage.

In Figure 7 we describe the architecture of the method proposed by us with its necessary steps:

1. Build the OWL ontology based on the source code;
2. Query the RDF graph of the OWL based ontology using SPARQL;
3. Measure metrics based on result sets;
4. Represent ontology's topic map;
5. Determine the final plagiarism degree by comparing these results with the results obtained from another ontology.

DOI: 10.12948/issn14531305/17.3.2013.07

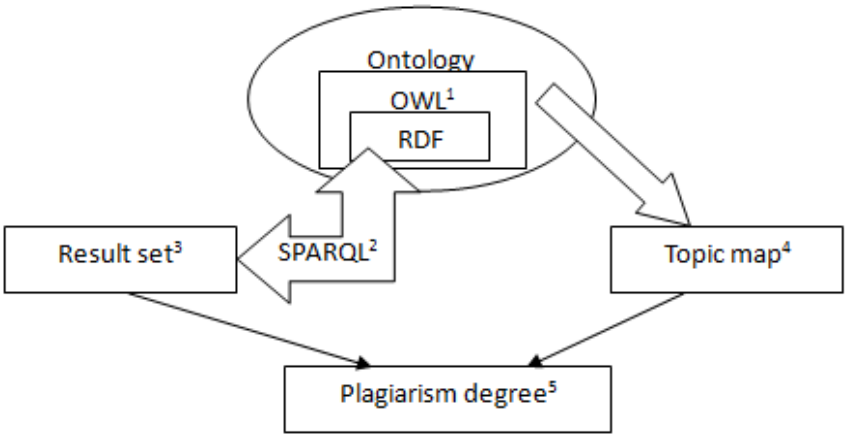

Fig. 7. Architecture of the plagiarism detection method

3 Future Developments

To have a reliable detection system based on this method, all the steps of the presented architecture must be made automatically. So to create such a system we will need a crawler that will parse the code and extract the OWL ontology, a set of defined metrics, each one with its own dynamically generated SPARQL query, and a custom representation of all the involved topic maps. These components will be created in our future work.

4 Conclusions

In this paper it was shown that ontologies can be used in detecting source code plagiarism. By using the OWL Web Ontology Language which is based on RDF Resource Description Framework and the SPARQL RDF based query language we can extract the needed information from our ontology that was built based on the vocabulary and taxonomy of a programming language source code. We saw that a way of constructing this kind of ontology is Protégé, a free open source ontology editor and that beside the metrics that can be measured using SPARQL we can see the graphic representation of the ontology by using a topic map.

However, the real benefit of using ontologies for complex software plagiarism detection systems is that all the detection process can be made automatically and in this way we can improve the quality of students' theses in particular and the quality of education in general. The introduced approaches are a good starting point for the future work to establish a fully automatically system for source code plagiarism detection.

References

[1] M. K. Shenoy, K. C. Shet and U. D. Acharya. (2012, May). Semantic Plagiarism Detection System Using Ontology Mapping. Advanced Computing: An International Journal [Online]. 3(3). Available: http://airccse.org/journal/acij/papers/0512acij06.pdf

[2] The Protégé Ontology Editor and Knowledge Acquisition System. Internet: http://protege.stanford.edu/ (2013, July 1).

[3] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler and F. Yergeau. (2004, February 4). Extensible Markup Language (XML) 1.0. W3C Recommendation [Online]. Third Edition. Available: http://www.w3.org/TR/2004/REC-xml-20040204/

[4] F. Manola and E. Miller (2004, February 10). RDF Primer. W3C Recommendation [Online]. Available: http://www.w3.org/TR/2004/REC-rdf-primer-20040210/

[5] S. Harris, A. Seaborne. (2013, March 21). SPARQL 1.1 Query Language. W3C Recommendation [Online]. Available: http://www.w3.org/TR/2013/REC-sparql11-query-20130321/

[6] J. Bao, D. Calvanese, B. C. Grau, et al.

DOI: 10.12948/issn14531305/17.3.2013.07

(2012, December 11). OWL 2 Web Ontology Language. W3C Recommendation [Online]. Second Edition. Available: http://www.w3.org/TR/owl2-overview/

[7] E Akin, Object Oriented Programming, Houston: Rice University Publishing House, 2001, pp. 33-34.

[8] S. Kara, O. Alan and O. Sabuncu, "An ontology-based retrieval system using semantic indexing", Information Systems, vol. 37, no. 4, pp. 294–305, June 2012.

[9] Pseudocode Standards, California Polytechnic State University Website, Internet: http://users.csc.calpoly.edu/~jdalbey/SWE/pdl_std.html (2013, July 1).

[10] C. Liu, H. Wang, Y. Yu and L. Xu, "Towards Efficient SPARQL Query Processing on RDF Data", Tsinghua Science & Technology, vol. 15, no. 6, pp. 613–622, December 2010.

[11] I. Ivan and C. Boja, Metode Statistice in analiza software. Bucharest: ASE Publishing House, 2004, pp. 218-224.

[12] S. Russel and P. Norving, Artificial Intelligence: A Modern Approach (2nd edition). New Jersey: Pearson Education Inc., 2003, pp. 350-352.

[13] P. Durusau, S. Newcomb and R. Barta (2007, November). Topic Maps Reference Model. International Organization for Standardization [Online]. Available: http://www.isotopicmaps.org/TMRM/TMRM-7.0/tmrm7.pdf

[14] D. Newman, T. Baldwin, L. Cavedon and E. Huang, "Visualizing search results and document collections using topic maps", Web Semantics: Science, Services and Agents on the World Wide Web, vol. 8, no. 2-3, pp 169–175, July 2010.

[15] A. Hatzigaidas, A. Papastergiou, G. Tryfon and D. Maritsa, "Topic Map Existing Tools: A Brief Review", in Proc. The International Conference on Theory and Applications of Mathematics and Informatics, Thessaloniki, Greece, 2004, pp 185-201

Ion SMEUREANU has graduated the Planning and Economic Cybernetics faculty in 1980, as promotion leader. He holds a PhD diploma in "Economic Cybernetics" from 1992 and has a remarkable didactic activity since 1984 when he joined the staff of Bucharest Academy of Economic Studies. Currently, he is a full Professor of Economic Informatics within the Department of Economic Informatics and the dean of the Faculty of Cybernetics, Statistics and Economic Informatics. He is the author of more than 16 books and an impressive number of articles. He was also project director or member in many important research projects. He was awarded the General Romanian Economist Association Excellence Diploma 2010, the Virgil Madgearu Excellence Award 2006, the award for the entire research activity offered by the Romanian Statistics Society in 2007 and many others.

Bogdan IANCU has graduated The Faculty of Cybernetics, Statistics and Economic Informatics from The Bucharest University of Economic Studies in 2010. He has a master in Economic Informatics (2012) and he is a PhD Candidate in Economic Informatics starting from 2012 in the field of Ontologies and eLearning. He is a teaching assistant in The Department of Economic Informatics of The Bucharest University of Economic Studies. His current work focuses on the analysis of semantic web and ontologies innovations. Other fields of interest include data mining, multimedia, object oriented programming in C++, windows applications programming in C# and Business Intelligence tools.

DOI: 10.12948/issn14531305/17.3.2013.07