Received: July 30, 2019

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Metric Method for Long Life Semantic Applications

Eman K. Elsayed[1] **Naglaa E. Ghannam**[1]*

[1]*Faculty of Science, Al-Azhar University (Girls Branch), Cairo, Egypt*
* Corresponding author's Email: Naglaasaeed@azhar.edu.eg

Abstract: Ontology is the core of the semantic applications, so the quality of it has a direct proportion with Ontology. It impacts directly on the long life of the semantic applications. There are different negative effects in the design of Ontology' classes as Blob, Lazy Class, Large Class, and Singleton. These negative effects called bad smells. However, detecting smells is not supported in any Ontology editors. This paper proposes a metric method called ONTOPYTHO. It can detect classes' smells automatically from Ontology models even if it is a Big Ontology. We programed ONTOPYTHO via Python and SPARQL languages. We evaluated the proposed method ONTOPYTHO by applying it on twelve publicly OWL Ontology projects. The detected smells appeared 117495 times in the twelve projects. The results showed that both the size and the number of classes of OWL Ontology has no effect in the presence of the smells. The results also showed that 69.24% of the classes are lazy classes. This means that big OWL Ontologies are not big in their nature, but because of theses lazy classes. The proposed method is the first method that detects Lazy class smell in the design of big OWL Ontology. In our random sample of big Ontologies, Lazy class smell appears approximately 99.8% of the smells.

Keywords: Semantic applications, Design smells, Ontology, Python.

1. Introduction

Over the last few years, semantic advances got to be vital in numerous areas of computer science. Semantic technologies have a wide range of applications, including model transformations, cloud security engineering, decision support, search and semantic integration [1] - [5]. The most important technology is the semantic web technology which makes web content comprehensible for both humans and machines. All semantic applications are based on Ontology which presents the semantic structure of the applications' domains.

Ontology is the cornerstone of the semantic web, so the quality of the Ontology impacts directly on semantic web applications' quality. The success of any Ontology design depends on the availability of the quality elements such as maintainability, manageability, testability, and comprehensibility. These elements are adversely affected by smells. Ontological smells are those structures that reflect analysed problems of OWL Ontologies causing inconsistencies, bad reasoning performance and many other quality problems [6]. While according to [7], they defined Ontology smells as patterns that appear evident but have no effect or not optimal in practice. Smells have many other terms as design defects, design flaws, pitfalls, and anti-patterns [8].

In this paper, we are interested in smells in Ontologies' classes. The class is the most vital component of Ontology components. Without it, there is no structure. There are different cases in design Ontology' classes, which have negative effects as Blob, Lazy Class, Large Class, and Singleton. These cases called bad smells on Ontology. The Blob case is assigned when the class has a great number of both attributes and operations and almost control the system causing low cohesion and a great probability for errors. Low cohesion implies incompatible design and high complexity. The Lazy class case is assigned when the class is leaf and has no functionalities. It has a little number or does not has any operations, it may be just a data store. While large class indicates the case when the class has a great number of operations or

functionalities but not all the functionalities of the system and has a little number of attributes. The singleton case when the class has only one instance and operation for retrieving this instance.

In large Ontologies systems, we need to measure system components to control its complexity. On the other hand, reasoners cannot deal with large Ontologies or cannot detect all the smells as shown by [7]. They cannot handle even all the errors resulting from a timeout. According to [9], they showed that reasoners of the Ontology verify the design quality to get consistency systems. However, reasoners verification against some structural smells is not found. The proposed smell detection method can detect structural smells without using the reasoners and whatever the Ontology size.

Generally, Ontology is not big but sometimes there are big Ontologies as GO (Gene Ontology). The proposed approach can evaluate both small and big Ontologies. According to [10], protégé needs at least 4G of RAM to manipulate the GO Ontology. Also, according to [11], they showed that using Gowinda to detect unbiased enrichment in gene sets from big datasets will cause Memory consumption. As GO needs about 1.2 GB of RAM and maybe increased according to the number of SNPs when be analyzed using "Gowinda". On the other hand, according to [12], they found that there is a correlation between the existence of smells and memory problems.

According to [13], they classified the detection strategies into three types which are software metrics, design patterns, and predefined rules. The proposed approach uses the software metrics strategy. Software metrics play a vital role in software development [14]. The quality of the software is measured according to how much the final software system matches its specification [14]. In software metrics techniques, the smells are detected using metrics and their thresholds. The proposed approach calculates the software design metrics using simple protocol and RDF query language (SPARQL) and Python programming language. The greater the number of attributes and operations the greater the design complexity and greater low quality. Earlier studies [15] showed that for avoiding software complexity and low quality, we should measure and control system components. The detection process of the design smell usually involves finding the fragments of design which violate these software metrics. The proposed approach detects the smells in an automatic way and the correction will be manual. Motivated by the research, the major contributions of this paper are five-fold:

- Assessing the correlation between smell's existence and memory consumption.
- Proposing an Ontology-Python (ONTOPYTHO) approach for detecting smells on OWL-light and big Ontologies.
- Describing the experimental evaluation of the proposed approach in 12 OWL projects. Showing how it detects 4 design smells which appeared 117495 times on the OWL Ontologies.
- Showing the benefits of using the Python programming language for smell detection in OWL Ontologies.
- Analyzing the correlation between the detected smells and showing how Blob and Large-class smells have a direct correlation.

The rest of this paper is organized as follows. In the next section, we present the related work. The details of the Ontology smells models are then introduced in section 3. Section 4 presents the proposed method of ONTOPYTHO. Section 5 presents the experimental evaluation of the proposed approach aimed to detect the design smells on OWL Ontologies. While section 6 presents the results and discussion. And, finally, the concluding remarks and future work are given along with scope for future work in section 7.

2. Related works

Several techniques were proposed in the literature for detecting and defining smells types in software systems generally and some in Ontologies. The automatic detection of smells is a good way to keep the maintenance, easy the evolution tasks and improves usability and software quality. There is a fact said that the smells detection at the design level reduces many codes of smells and maintenance costs which is more general. According to [16] and [17], they proposed using of Bayesian Belief Networks to detect 3 smells which are Blob, Functional Decomposition, and Spaghetti Code using machine learning techniques. In [16] and [17], they can work with missing data and provide an abstract definition as a detection algorithm for every smell. They detect smells but not in OWL ontologies. The proposed method detects Lazy Class, Large Class, and Blob in OWL Ontology systems. On the other hand, reference [18] analyzed 30 releases of three different open-source systems which are ArgoUML, Hibernate, and ANT. They detected 29 smells, 13 of them are design's smells and 16 are lexical smells. They showed that lexical smells can make, in some cases, classes with design smells more fault-prone when both occur in classes

of object-oriented systems. They proved that classes containing design' smells only are more change- and fault-prone than classes with lexical smells only. Reference [19] introduced the "Arcan" tool which is written in Java 8 to detect architectural smells. "Arcan" is used to support the automatic analysis of software architecture through a graph representation of data, providing support during the software development and maintenance processes. They detected 4 smells which are Cyclic Dependency in classes, Cyclic Dependency in packages, Unstable Dependency, Hub Like Dependency. In reference [20], they introduced multi-objective genetic programming (MOGP) to find the best metrics that increase the detection of smells examples. They evaluated their proposal on seven large open-source systems and found that, on average, most of the different three code-smells types were detected with an average of 86% of precision and 91% of recall. On the other hand, [21] proposed DÉCOR which contains a consistent vocabulary about code-smell to specify Smells. The descriptions of the smells are then converted to detection rules. This approach has detected only four design code-smells which are the Blob, functional decomposition, spaghetti code, and Swiss-army knife. They validated the detection algorithms on XERCES v2.7.0, and discuss the precision of these algorithms on 11 open source systems.

Our proposed method detects "Blob, Lazy Class, Large Class, and Singleton" design smells in big and light OWL ontology.

Several techniques for detecting Ontology smells can be divided into two main branches: Code smells detection and design smells detection. According to [22], they presented EvoOnt, a software repository data exchange format which is based on the Web Ontology Language (OWL). EvoOnt includes software, release, and bug-related information. They also introduced iSPARQL which is a SPARQL-based

Semantic Web query engine containing similarity joins. Together with EvoOnt, iSPARQL can accomplish a sizable number of tasks sought in software repository mining projects framework to detect four smells in the code of the OWL Ontologies. The detected smells were Alien spider, God Class, Orphan method query and long parameter list smells. Reference [23] Presented OCEAN to detect code smells from the source-code of the Ontology models and the production of Ontological individuals that represent code smells. OCEAN detected two code smells which are God Class and Brain method. They also introduced a tool called RESYS that makes the refactoring process easily and linked semantically to their code smells. While according to [24], they presented OntoUml to detect semantic design smells. Their approach focused on the design smells that cannot be detected as modeling errors. They detected seven semantic smells in the design of the OWL Ontologies. According to [25], they detected nine smells in the design of Ontology. They classified them into three groups which are Logical smells, Non-logical smells and Guidelines smells. But the detected smells are existing in the inconsistent ontologies.

Recently, Ontologies have become a promising way of building intelligent systems. Ontology and Python have a wide range of applications. In the semantic web, [26] used Ontology-Python for making data integration and information discovery for linked open data. They used graph theory to enhance the features of the semantic data in Ontologies and used the Python network package to generate the graph objects. On the other hand, [27] proposed architecture for linking the contents of the media outlets semantically. This architecture was implemented in Python using AllegroGraph which is designed for storing RDF Triples and has a client interface for Python. According to [28], they handled Gene Ontology resources in Python to retrieve the annotations of the GO.

Also, to perform gene enrichment analyses, and to compute the semantic similarity between GO terms. On the other hand, [29] presented research on how the tweets are retrieved from tweeter through a tweet script crawler which was built with Python using the Ontology to classify the tweets. According to [30], they used Python to link the Ontology to various technologies for NLP tasks. They used the "Owlready" which is a new Python OWL API library to be used as a verbalizer. This verbalizer will run on the Ontology to be used as domain experts and linguists. While according to [31], they created some Ontologies using the Crop as a domain for these Ontologies. They implemented the presented schema in Python using the "Owlready" library to generate all Ontology components such as classes and properties. They used Python also to implement similarities techniques for finding the alignments between the generated components of the Ontologies. [32] used the "Gensim" Python library and two-layer neural network algorithm word2vec to extract the word's semantic concept from the Ontologies. They proposed a technique to map the end user's words which presented in the natural languages to the corresponding concepts in the Ontology. [33] Presented "OntoSenticNet" which is a common-sense Ontology for sentiment analysis. The "OntoSenticNet" is based on

"SenticNet"; a semantic network of 100,000 concepts. They showed that using Python is better more than RDF/XML as it provides easier support for the integration of "OntoSenticNet" into real-world applications.

In this paper the proposed approach uses Python for detecting Ontology design smells automatically through the ONTOPYTHO method.

3. Ontology smells models

This section presents the different Ontology models of Class smells as Blob, Large class, Lazy class, and Singleton. The simulation visualizes the models using Protégé platform.

3.1 Blob smell model

Blob smell is characterized by a class that has a great number of both attributes and operations and has the most functionalities. Fig. 1 shows the OWL Ontology model for presenting the Blob smell. We can see class1 which has a great number of both object properties (operations) and "datatype" properties (attributes). Other classes just have a low number of operations and all the range of their operations is class1.

Figure. 1 The OWL ontology model for blob smell

3.2 Lazy class smell model

The Lazy class smell is a leaf class which has no subclasses and has a little number of operations may be zero. We can consider it a data class. Fig. 2 shows the OWL Ontology model for presenting the Lazy class smell. It contains four classes (Class1, Class2, Class3, and Class4). All the classes have operations (object properties) while Class1 does not has any operations, in addition to that, it is a leaf class, so it is classified as a lazy class.

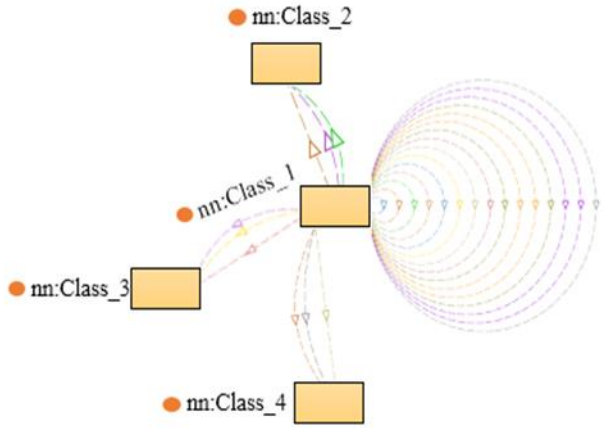

Figure. 1 The OWL ontology model for blob smell

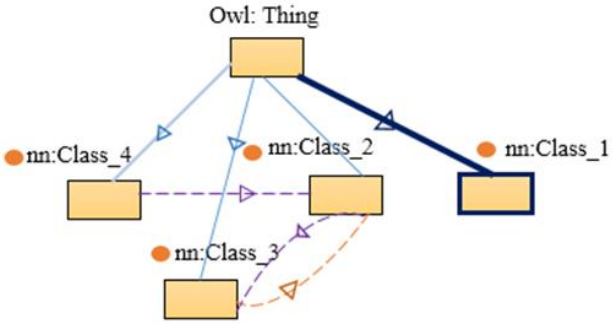

Figure. 2 The OWL ontology representing the lazy class smell

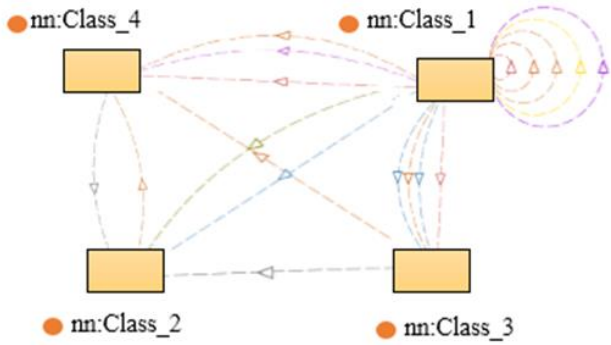

Figure. 3 The OWL ontology representing the large class smell

3.3 Large class smell model

Large class smell characterized by a great number of operations or functionalities and has a little number of attributes or does not have any attributes. Fig. 3 shows the OWL Ontology model of the large class smell. The model contains 4 classes; all of them have object properties. So, none of them is a data class which implying that Blob smell is not satisfied here. For that, the class here is classified as a large class.

3.4 Singleton smell

Singleton smell is a design smell that restricts the number of instances of the class to only one instance. The class has only one instance and operation for retrieving this instance. Fig. 4 shows the OWL Ontology for presenting singleton class smell. It contains (Class1) which has an object property (nn:get_instance) and only one instance(nn:Instance). The domain and the range of the object property are the same which implies that the operation will get the instances from the same class (Class1) as in (1). While (2) implies that the class has only one instance and (3) implies that the operation will retrieve that instance.

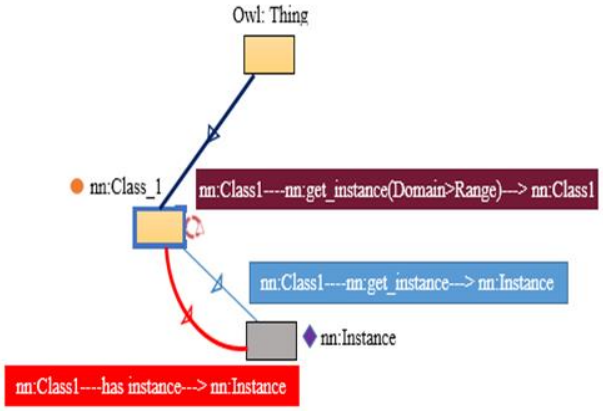

Figure. 4 The OWL ontology representing the singleton smell

nn:Class1--nn:get_instance(Domain>Range)→ nn:Class1 (1)

nn:Class1---has instance --- → nn: Instance (2)

nn:Class1---nn:get_instance--→nn:Instance (3)

4. The proposed detection method "ONTOPYTHO"

This section presents the pseudo code of "ONTOPYTHO" Algorithm 1; the proposed Ontology smells detection Algorithm. ONTOPYTHO is used to detect classes' smells in large OWL Ontologies. The proposed method is based on the merging of Python programming language and RDF query language (SPARQL). Python is a high-level programming language which is considered as an interpreter with dynamic semantics. It reduces the cost of the program maintenance, supports modules and packages, which implies code reuse [34].

The proposed approach deals with the Ontology' design, not the code or the converted Ontology. That is to guarantee no missing information or parts of the Ontology' structure according to the conversion or the code generation. Also, Python has "rdflib" library to handle big Ontology as GO and give the proposed approach the ability to evaluate it without needing to manipulate it with Ontology editors. Also, adding the proposed approach to any Python library can improve the quality of the Ontologies before using in the semantic web.

ONTOPYTHO (Ontology smells detection Algorithm) uses SPARQL queries to calculate Ontology metrics and to produce a report of the detected smells and model analysis.

Algorithm 1. Class Smells detection' algorithm

Input: OWL Ontology O, Threshold for attributes x and operations y.

Output: a list of detected smells

```
1   Insert Ontology O,
2   Counter n is a number of classes in O;
3   For (C > =1 && C < = n):
4       A  ←getNumOfAttributes;
5       P  getNumOfOperations;
                ←
6       I  ←getNumOfInstances;
7       ð  ←getOperationValue;
8       run Blob query Q1;
9           if Blob classes > =1:
10                  print "Blob detected";
11          else print (" ");
12      run Lazy Class query Q2;
13          if Lazy Classes > =1:
14                  print " Lazy class detected";
15          else print (" ");
16      run Large Class query Q3;
17          if Large Classes > =1:
18                  print ("large class detected");
19          else print (" ");
20      run Singleton query Q4;
21          if Singleton classes > =1:
22                  print (" singleton detected");
23          else print (" ");
24          End.
```

The proposed approach pre-process is analyzing and parsing OWL Ontology model. This process uses Python libraries as “rdflib” library. Then synchronously, the four SPARQL queries run. The four SPARQL queries are:

- Blob detection SPARQL Query Q1.
- Lazy class detection SPARQL Query Q2.
- Large class detection SPARQL Query Q3.
- Singleton detection SPARQL Query Q4.

Finally, the visualization process displays the report of the detected Smells in the OWL model.

4.1 Blob detection SPARQL query

Using SPARQL query Q1, we detect Blob smell and retrieve Blob classes from the OWL Ontology. The query code as shown in Algorithm 2. This query returns the Blob classes by counting their number of operations and their number of attributes. This detection process has two restrictions. The first restriction indicates that the number of operations must excessed than "x". While the second restriction indicates that the number of attributes must excessed than y. Both "x" and "y" are the accepted threshold which class should have. This query could be general. So, according to developers’ accepted thresholds, they can detect Blob classes.

Algorithm 2. (Q1. Blob detection SPARQL Query)

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT ?Blob_classes (count(?d) as ?NO_Operations) (count(?A) as ?No_Attributes)
where{{?d a owl:ObjectProperty.
?d rdfs:domain ?Blob_classes} UNION
{?A a owl:DatatypeProperty. ?A rdfs:domain ?Blob_classes} }
group by ?Blob_classes having(?No_Operations>?x && ?No_Attributes>?y)
```

4.2 Lazy class detection SPARQL query

Using SPARQL query Q2, we detect the lazy class smell as shown in Algorithm 3. The query retrieves the Lazy classes defined by two restrictions. The Filter part of the query presents the restrictions. The restrictions indicate that Lazy classes have not any operations and they have leaf classes.

Algorithm 3. (Q2: Lazy Class detection SPARQL Query)

```
SELECT distinct ?LazyClasses
WHERE {?LazyClasses a owl:Class.
Filter NOT EXISTS{{?h a
owl:ObjectProperty.
?h rdfs:domain ?LazyClasses.} UNION{?subject rdfs:subClassOf ?LazyClasses.} }}
group by ?LazyClasses
```

4.3 Large class detection SPARQL query

Using SPARQL query Q3, we detect the large class smells as shown in Algorithm 4. The query retrieves the large classes. That is by counting process of the number of operations “?No_Op”. The restriction implies exceeding the number of operations more than “y”. Where “y” is the accepted threshold for the developers. Synchronously, the Large classes might have datatype properties or not. We concerned with the number of object properties.

Algorithm 4. (Q3. Large class detection SPARQL Query)

```
SELECT ?Large_classes (count(?d) as ?No_Op)
where{?d a owl:ObjectProperty.
?d rdfs:domain ?Large_classes.}
group by ?Large_classes having(?No_Op>?y)
```

4.4 Singleton detection SPARQL query

The SPARQL query Q4 for detecting the Singleton smell as shown in Algorithm 5. The query Q4 retrieves the singleton classes under three restrictions. We have two Filters. The first restriction in the first Filter for indicating that both the domain and the range of the operation are the same. The second Filter indicates that the retrieved instance is an instance of the same class that contains the operation. The last restriction is for indicating that the class has only one instance.

Algorithm 5. (Q4 Singleton detection SPARQL Query)

```
SELECT ?Sing_classes
where{?d a owl:ObjectProperty.
?d rdfs:domain ?Sing_classes.
?r rdfs:range ?Sing_classes.
filter  (?d=?r).
?in rdf:type ?Sing_classes.
filter exists {?o ?p ?d}.
?A a owl:DatatypeProperty.
?A rdfs:domain ?Sing_classes.}
group by ?Sing_classes
having(count(?A)=1)
```

5. Experimental evaluation

In this section, we assess how well the proposed approach ONTOPYTHO can predict the quality of OWL Ontology design. We assess our approach by applying it on twelve popular OWL Ontologies which are considered as big Ontologies. We downloaded them from Github, [35], and [36]. The size of the Ontologies includes all Ontology components. Table 1 shows the OWL Ontologies features. The random sample of twelve Ontologies in different fields as lexicons, Chemistry, Biology, Anatomy, Industry, and bioinformatics. The research study includes the identification and repeating of the smells across different domains and different sizes.

Table 1. Description of OWL ontologies under analysis

OWL Ontology Projects	Size MB	Classes	Specification
dbpedia_2015-10.owl	2.32	739	The DBpedia datasets, each release of this Ontology corresponds to a new release of the DBpedia data set
dbpedia_2016-04.owl	2.36	754	
dbpedia_2016-10.owl	2.37	760	
Gene Ontology (go.owl)	171	61714	An Ontology for describing the function of genes and gene products
IFC2X3_Final.owl	3.18	1149	Industry Foundation Classes IFC Ontology
Edam.owl	3.01	3379	EDAM is an Ontology of bioinformatics types of data.
Gaz.owl	590	10144	Gazetteer, a gazetteer constructed on Ontological principles
chebi.owl	471	32288	Chemical Entities of Biological Interest. A structured classification of molecular entities of biological interest.
Foodon.owl	7.21	6556	The core repository for the food Ontology project
hp-full.owl	95.3	47015	human-phenotype-Ontology is for the description of human clinical features
xao.owl	4.26	1735	Xenopus Anatomy Ontology. Anatomy and development of the African clawed frog.
Zfa.owl	8.27	3200	Zebrafish anatomy and development Ontology. A structured controlled vocabulary of the anatomy and development of the Zebrafish.

For explaining the proposed approach, we display a part of the result of applied ONTOPYTHO on the ‘dbpedia_2015-10.owl ‘in Fig. 5.

```
(u'http://dbpedia.org/ontology/GrandPrix'), 'Large class detected')
(u'http://dbpedia.org/ontology/Place'), 'Large class detected')
(u'http://dbpedia.org/ontology/Athlete'), 'Large class detected')
(u'http://dbpedia.org/ontology/GolfPlayer'), 'Large class detected')
(u'http://dbpedia.org/ontology/Artist'), 'Large class detected')
(u'http://dbpedia.org/ontology/River'), 'Large class detected')
(u'http://dbpedia.org/ontology/Species'), 'Large class detected')
(u'http://dbpedia.org/ontology/Organisation'), 'Large class detected')
(u'http://dbpedia.org/ontology/Work'), 'Large class detected')
```

Figure. 5 The result of running ONTOPYTHO on ‘’dbpedia_2015-10’’ Ontology

6. Results and discussion

After applying ONTOPYTHO on the OWL Ontologies in Table 1, we detected the smells as presented in Table 2. The proposed approach detected 3 smells which were appeared 117495 times. We can note that lazy class smell is the most detected smell while Singleton smell has not been detected. Both Blob and large class smells have a little appearance in the Dbpedia versions. They decreased in the new versions. The Lazy class smell is the most appeared one. It increased more and more in the new versions as in Fig. 6.

Table 3 summarizes the results we obtained for the detection of 3 smells from our input 12 OWL Ontologies. The integer value represents the number of occurrences of the smells in the OWL Ontology. The percentage is the ratio of this value according to the total number of smells. We noted that the same class can be affected by more than one smell. For example, a class can be a large class and a blob class simultaneously.

From Table 3, we can note that Lazy class smell has the maximum ratio of smells which is 99.865% while Blob smell has the minimum ratio which is 0.0374%. But is there any relation between the three smells?

The analysis of the relation between class' smells types means the determination of the relation between Ontology's concepts. So, we used a statistical analysis package as SPSS (Statistical Package for the Social Sciences) to analyze the correlation (Pearson correlation coefficient) between the three detected smells. We found that the highest correlation is between Blob and Large Class' smells as the Pearson correlation coefficient between them equals 0.889 i.e. there is a strong correlation between them. This means that the presence of one means the presence of the other by a large percentage. While the correlation between Lazy class smell and the Blob smell equals 0.276, and between Lazy class and Large-class equals 0.221;

Table 2. Design smells per OWL ontologies

OWL Ontologies	Smells			
	Blob	Lazy Classes	Large Classes	Singleton
dbpedia_2015-10.owl	8	490	17	0
dbpedia_2016-04.owl	7	502	14	0
dbpedia_2016-10.owl	7	506	14	0
Gene Ontology (go.owl)	0	44155	0	0
IFC2X3_Final.owl	2	393	9	0
EDAM.owl	0	2742	10	0
gaz.owl	3	10023	7	0
chebi.owl	8	12052	18	0
foodon.owl	5	5500	13	0
hp-full.owl	4	36735	8	0
xao.owl	0	1515	4	0
zfa.owl	0	2724	0	0

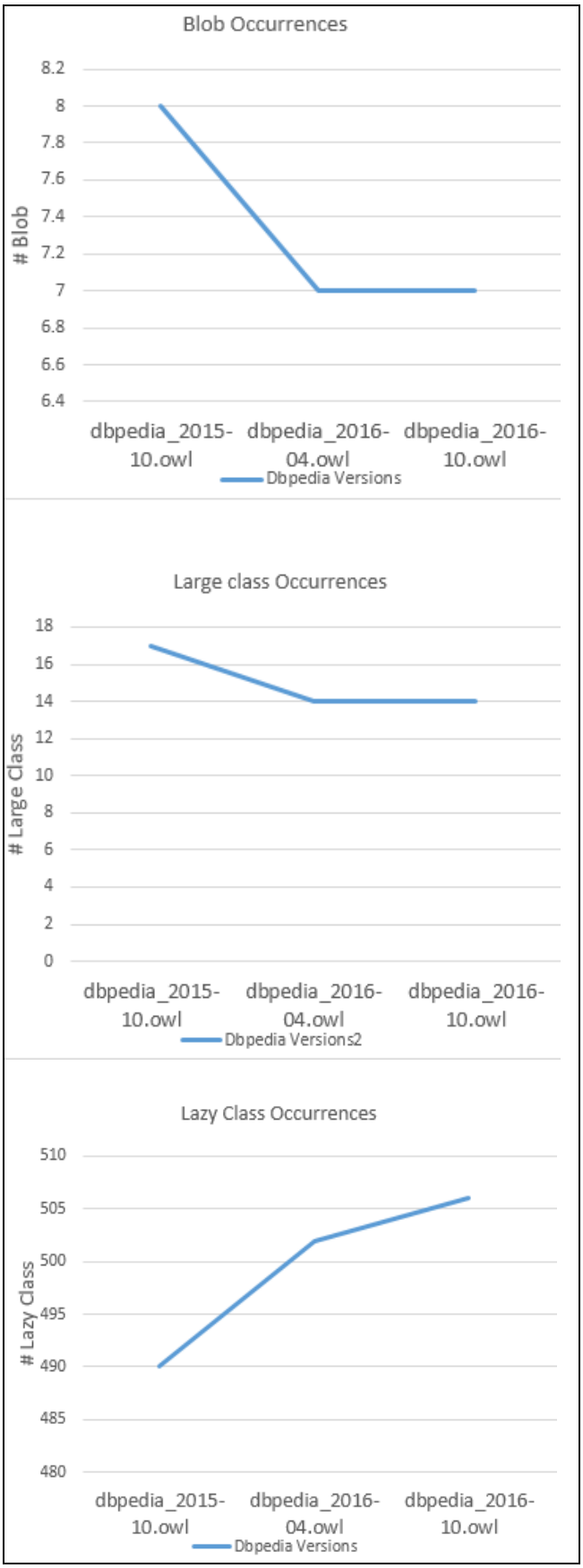

Figure. 6 The appearance of the detected smells in the Dbpedia OWL Ontologies versions

Table 3. Ontopytho results for the detection of 3 smells in 12 OWL ontologies

	Class Smells		
	Blob	Lazy Class	Large Class
Total	44	117337	114
Ratio	0.0374%	99.865%	0.0970%

Table 4. The SPSS Pearson correlation coefficient matrix between the class' smells

		Blob	Lazy Class	Large Class
Blob	Pearson correlation	1	0.276	0.889
	Sig. (2-taild)		0.385	0
Lazy Class	Pearson correlation	0.276	1	0.221
	Sig. (2-taild)	0.385		0.489
Large Class	Pearson correlation	0.889	0.221	1
	Sig. (2-taild)	0	0.489	

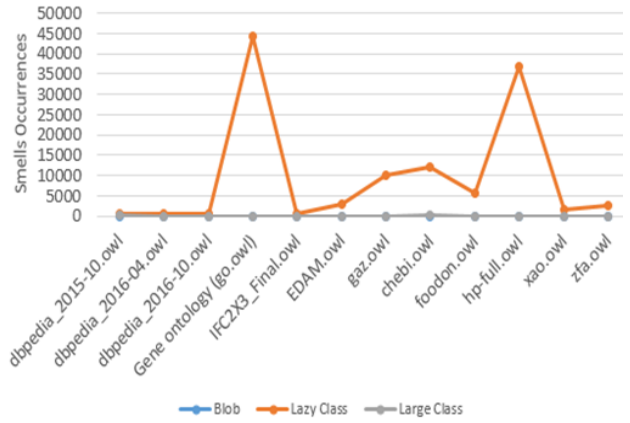

Figure. 7 The similarity between the means of "Blob and Large-class" smells and differences between them and "Lazy class" smell

which means that there is a low correlation between them. This means that the presence of one does not imply the presence of the other. The SPSS Pearson correlation coefficient matrix is shown in Table 4.

Fig. 7 assessed the similarity between both the means of Blob and Large-class smells like the result of the correlation.

Now, we need to address if there is a relation between the number of classes and the presence of the smells. Table 5 illustrates the ratio between the number of classes and smells in every OWL Ontology.

From Table 5, although "gaz.owl" does not has the maximum number of classes, but it has the maximum ratio of smells 98.90% in the classes of it. This leads us to the fact that the number of classes in OWL Ontologies has no effect on smells presence. Now we will ask, is there a relation between the size of the Ontology and smells existence?

Table 5. The ratio of smells according to the classes of OWL ontologies

OWL Ontologies	Classes	Smells	ratio
dbpedia_2015-10.owl	739	515	69.68%
dbpedia_2016-04.owl	754	523	69.36%
dbpedia_2016-10.owl	760	527	69.34%
Gene Ontology (go.owl)	61714	44155	71.54%
IFC2X3_Final.owl	1149	404	35.16%
EDAM.owl	3379	2752	81.44%
gaz.owl	10144	10033	98.90%
chebi.owl	32288	12078	37.40%
foodon.owl	6556	5518	84.16%
hp-full.owl	47015	36747	78.16%
xao.owl	1735	1519	87.55%
zfa.owl	3200	2724	85.12%

Table 6. The SPSS Pearson correlation coefficient matrix between the size of the ontologies and the smells

		Size	Lazy Class
Size	Pearson Correlation	1	0.283
	Sig. (2-taild)		0.373
Smells	Pearson Correlation	0.283	1
	Sig. (2-taild)	0.373	

To answer this question, we analyzed the correlation between them using SPSS as in Table 6. We found that there is no relation between them as the correlation coefficient is 0.283. So, the size of the Ontologies has no effect on smells presence.

Comparing the proposed method "ONTOPYTHO" to other techniques in section of related works [16, 17, 19-25]. The results of comparison are in Table 7. The related references detected smells generally in software projects and some of them in OWL Ontologies. References [22], [24], and [25] just detected smells in OWL ontologies. The comparison included the number of evaluated software projects, the number of detected smells, the software systems weather they were OWL ontologies or not, and the number of occurrences of smells.

Table 7. Comparison between the "ONTOPYTHO" and other techniques

The techniques	# software projects	# Detected Smells	Software Systems	# occurrences of smells
Bayesian Belief Networks [16]	2	1	-	19
BDTEX [17]	2	3	-	46
Arcan [19]	8	4	-	1025
MOGP [20]	7	3	-	725
DÉCOR [21]	10	4	-	513
EvoOnt [22]	1	5	OWL	0
OCEAN[23]	4	3	-	445
OntoUml[24]	1	7	OWL	3612
OWL catalogue [25]	0	11	OWL	0
ONTOPYTHO Proposed method	**12**	**4**	**OWL**	**117495**

Table 8. The shared detected smells using proposed method and other techniques

The techniques	# Blob	# LazyClass
Bayesian Belief Networks [16]	19	-
BDTEX [17]	13	-
DÉCOR [21]	44	-
EvoOnt [22]	9	-
OCEAN [23]	134	21
ONTOPYTHO Proposed method	44	117337

We can note that the proposed method has the greatest number of the evaluated projects and the greatest number of occurrences of smells which are 12 projects and 117495 times. The proposed method detected smells "Blob, Lazy Class, and Large Class" which are not detected by any other techniques. But there are some cases in the other techniques which detected just Blob as in [16, 17, 21-23]. Lazy class was detected only in one case in reference [23] as in Table 8. The proposed method has the maximum number of Lazy Class smell which is 117337 smells and of ratio 99.865%.

Strengths of the proposed method which does not exist in the other techniques:

(1) The proposed method detected the smells Blob, Lazy Class, and Large Class which are not detected in OWL Ontologies using the other techniques.
(2) The proposed method detects smells in big and light OWL Ontologies.
(3) The proposed method improves semantic web projects which are developed using Python programing language.
(4) The proposed method is the first method detected Lazy Class smell in OWL Ontology design with a high percentage.

Finally, we recommend treating the detected smells by using classification for large classes or Blob and distributing the operations between the other classes and delete the lazy classes. That leads to minimize the Ontology' size to be suitable to manipulate it in any Ontology editors, save time, and minimize storage.

7. Conclusion

Ontologies are assuming the crucial role in Semantic Web vision. Improving the quality of Ontology design implies high-quality semantic web applications. We presented the ONTOPYTHO approach to detect smells on OWL Ontologies. The proposed approach is based on the metric method via the Semantic Web query language SPARQL and Python programming language. The proposed approach provides the ability to improve the quality of software systems represented in the OWL Ontology format. We applied ONTOPYHO on twelve OWL Ontologies. The proposed approach presented the method for detecting four design smells which are Blob, Large class, Lazy class, and Singleton smells. We detected three design smells which appeared 117495 times in the OWL Ontologies. We found that there is a direct relation between Blob and Large-class smells while there is a reverse relation between them and the Lazy class smell. Also, we found that both the size and the number of classes of the OWL- Ontology has no effect in the presence of the smells. The results showed that 69.24% of the classes are lazy classes. This means that big OWL Ontologies are not big in their nature, but because of the existence of theses lazy classes. So, detecting Lazy classes using ONTOPYTHO approach and deleting them will reduce the size of the ontologies. This consequently will allow reasoner check, manipulation using any ontology editor like protégé and solves the difficulty of using SPARQL queries directly according to the Ontology editor plugin. Also, when ONTOPYTHO detects large Ontology' classes, users can divide these classes into subclasses. The proposed approach provides the ability to measure the size and complexity of the OWL Ontologies. Finally, the strong correlation between large class and Blob smells leads to solve Blob when solving large-class.

In the Future works, we will insert the ONTOPYTHO into Python as a library. Also, we will insert all the detected semantic smells to Ontology Catalog. Finally, we will analyze the reasons of the relation between Blob and Large-class smells in Ontology and how can we avoid smells happen.

References

[1] S. A. Aljawarneh, A. Alawneh, and R. Jaradat, "Cloud security engineering: Early stages of SDLC", *Future Generation Computer Systems*, Vol.74, No.C, pp.385-392, 2017.

[2] W. Bartussek, T. Weiland, S. Meese, M. O. Schurr, M. Leenen, A. Uciteli, S. Kropf, H. Herre, C. Goller, P. Blohm. And W. Lauer, "Ontology-based search for risk-relevant PMS data", In: *Proc. of the 3rd Biennial South African on Biomedical Engineering Conf. (SAIBMEC)*, pp.1-4, 2018.

[3] G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati, "Using Ontologies for semantic data integration", In: *Flesca S., Greco S., Masciari E., Saccà D. (eds) A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years. Studies in Big Data*, Vol.31, pp.187-202, 2018.

[4] G. Kappel, E. Kapsammer, H. Kargl, G. Kramler, T. Reiter, W. Retschitzegger, W. Schwinger, and M. Wimmer, "Lifting metamodels to Ontologies: A step to the semantic integration of modeling languages", In: *Proc. of International Conf. on Model Driven Engineering Languages and Systems*, pp.528-542, 2006.

[5] P. Maurice, F. Dhombres, E. Blondiaux, S. Friszer, L. Guilbaud, N. Lelong, B. Khoshnood, J. Charlet, N. Perrot, E. Jauniaux, and D. Jurkovic, "Towards Ontology-based decision support systems for complex ultrasound diagnosis in obstetrics and gynecology", *Journal of Gynecology Obstetrics and Human Reproduction*, Vol.46, No.5, pp.423-429, 2017.

[6] O. Corcho, C. Roussey, O. Šváb-Zamazal, and F. Scharffe, "SPARQL-DL queries for Antipattern Detection", In: *Proc. of the 3rd International Conf. on Ontology Patterns WOP'12, Eva Blomqvist, Aldo Gangemi, Karl Hammar, and Mari Carmen Suárez-Figueroa (Eds.)*, CEUR-WS.org, Vol.929, pp.85-96, 2012.

[7] O. Corcho, C. Roussey, O. Šváb-Zamazal, F. Scharffe, and S. Bernard, "Antipattern detection in web Ontologies: an experiment using SPARQL queries", In: *Proc. of* the *12th International Francophone Conf. on Extraction and Knowledge Management*, pp.263-268, 2012.

[8] K. Alkharabsheh, Y. Crespo, E. Manso, and J. A. Taboada, "Software Design Smell Detection: a systematic mapping study", *Software Quality Journal*, Vol.26, No.4, pp.1-80, 2018.

[9] S. Staab and R. Studer, "Handbook on Ontologies", (2nd ed.), *Springer Science & Business Media*, Springer-Verlag, Berlin, Heidelberg, pp. 21-43, 2010.

[10] Protege setup for GO Eds, http://wiki.geneOntology.org/index.php/Protege_setup_for_GO_Eds (Accessed February 2019).

[11] R. Kofler and C. Schlötterer, "Gowinda: unbiased analysis of gene set enrichment for genome-wide association studies", *Bioinformatics*, Vol.28, No.15, pp.2084-2085, 2012.

[12] K. Jezek and R. Lipka, "Smells causing memory bloat: A case study", In: *Proc. of the 24th International Conf. on Software Analysis, Evolution and Reengineering (SANER)*, pp.306-315, 2017.

[13] M. Misbhauddin and M. Alshayeb, "UML model refactoring: a systematic literature review", *Empirical Software Engineering*, Vol.20, No.1, pp.206-251, 2015.

[14] M. K. Gopal, "Design Quality Metrics on the Package Maintainability and Reliability of Open Source Software", *International Journal of Intelligent Engineering and Systems*, Vol.9, No.4, pp. 195-204, 2016.

[15] M. Lanza and R Marinescu, "Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems", (1st ed.), *Springer Publishing Company, Incorporated, Springer-Verlag, Berlin, Heidelberg*, pp. 45-72, 2007.

[16] F. Khomh, S. Vaucher, Y. G. Guéhéneuc, and H. Sahraoui, "A bayesian approach for the detection of code and design smells", In: *Proc. of the Ninth International Conf. on Quality Software*, pp.305-314, 2009.

[17] F. Khomh, S. Vaucher, Y.G. Guéhéneuc, and H. Sahraoui, "BDTEX: A GQM-based Bayesian approach for the detection of Smells", *Journal of Systems and Software*, Vol.84, No.4, pp.559-572, 2011.

[18] L. Guerrouj, Z. Kermansaravi, V. Arnaoudova, B. C. Fung, F. Khomh, G. Antoniol, and Y. G. Guéhéneuc, "Investigating the relation between lexical smells and change-and fault-proneness:

an empirical study", *Software Quality Journal*, Vol.25, No.3, pp.641-670, 2017.

[19] F. A. Fontana, I. Pigazzini, R. Roveda, D. Tamburri, M. Zanoni, and E. Di Nitto, "Arcan: a tool for architectural smells detection", In: *Proc. of International Conf. on Software Architecture Workshops (ICSAW)*, pp. 282-285, 2017.

[20] U. Mansoor, M. Kessentini, S. Bechikh, and K. Deb, "Code-smells detection using good and bad software design examples", *Technical Report*, COIN Report Number 2014009, 2013.

[21] N. Moha, Y.G. Gueheneuc, L. Duchien, and A. F. Le Meur, "Decor: A method for the specification and detection of code and design smells", *IEEE Transactions on Software Engineering*, Vol.36, No.1, pp.20-36, 2010.

[22] C. Kiefer, A. Bernstein, and J. Tappolet, "Mining software repositories with isparol and a software evolution Ontology" In: *Proc. of the Fourth International Workshop on Mining Software Repositories*, pp.10-10. 2007.

[23] L. P. da Silva Carvalho, R. L. Novais, L. do Nascimento Salvador, and M. G. de Mendonça Neto, "An Approach for Semantically-Enriched Recommendation of Refactorings Based on the Incidence of Code Smells", In: *Proc. of International Conf. on Enterprise Information Systems*, pp.313-335, 2017.

[24] G. Guizzardi and T. P. Sales, "Detection, simulation and elimination of semantic Smells in Ontology-driven conceptual models", In: *Proc. of International Conf. on Conceptual Modeling*, pp.363-376, 2014.

[25] C. Roussey, O. Corcho, and L. M. Vilches-Blázquez, "A catalogue of OWL Ontology Antipatterns", In: *Proc. of the fifth international Conf. on Knowledge Capture*, pp.205-206, 2009.

[26] A. Akgün and S. Ayvaz, "An Approach for Information Discovery Using Ontology In Semantic Web Content", In: *Proc. of International Conf. on Information Science and System*, pp.250-255, 2018.

[27] P. Bubna, S. Sharma, and S.K. Malik, "Linking Online News Semantically Using NLP and Semantic Web Technologies", *International Journal of Computer Sciences and Engineering*. Vol.6, No.7, pp.589-598, 2018.

[28] C. Dessimoz and N. Škunca, "The Gene Ontology Handbook", (1st ed.), *Humana Press*. Vol.1446, pp.15-24, 2017.

[29] S. F. Huang, C. J. Su, and M. B. V. Saballos, "Social media big data analysis for global sourcing realization", In: *Bhatia S., Mishra K., Tiwari S., Singh V. (eds) Advances in Computer and Computational Sciences. Advances in Intelligent Systems and Computing*, Vol.554, pp.251-256, 2018.

[30] C. M. Keet, M. Xakaza, and L. Khumalo, "Verbalising OWL Ontologies in isiZulu with Python", In: *Proc. of the 14th European Semantic Web Conf. ESWC*, pp.59-64. 2017.

[31] N. Chatterjee, N. Kaushik, D. Gupta, and R. Bhatia, "Ontology merging: A practical perspective", In: *Proc. of International Conf. on Information and Communication Technology for Intelligent Systems*, pp.136-145, 2017.

[32] B. Zarei, S. Heil, and M. Gaedke, "Natural-Language-Enabled End-User Tool Endowed with Ontology-Based Development", In: *Proc. of International Conf. on Web Engineering*, pp.473-476, 2018.

[33] M. Dragoni, S. Poria, and E. Cambria, "OntoSenticNet: A commonsense Ontology for sentiment analysis", *IEEE Intelligent Systems*, Vol.33, No.3, pp.77-85, 2018.

[34] What is Python? Executive Summary. https://www.Python.org/doc/essays/blurb (Accessed March 2019).

[35] Ontologies & RDF datasets. The OBO Foundry http://www.obofoundry.org/ (Accessed February 2019).

[36] The DBpedia. https://wiki.dbpedia.org/Datasets (Accessed February 2019).