Generating Visualizations of Ontologies in the Logic Graphs Language

Vasiliy Kuryshev, Ildar Baimuratov, Vladislav Shmatkov, Dmitry Mouromtsev
ITMO University
Saint Petersburg, Russia
{vk937937, baimuratov.i, shmatkovvlad, d.muromtsev}@gmail.com

***Abstract*—We analyze the existing ontology visualization tools. Our analysis shows that there is no tool that represents the semantics of logical operations in ontologies, as Ch. S. Pierce's existential graphs do for first order logic. Previously, we developed Logic Graphs – a semantic-oriented visual language for ontologies, complete with respect to OWL DL. The goal of this research is to develop a tool for generating ontology visualizations in Logic Graphs. This tool parses an ontology to extract classes and their properties, converts ontology classes to their descriptions in DOT format, and builds graphical representations of ontology classes from DOT descriptions. Ontology visualization in Logic Graphs facilitates the users perception of ontologies, and provides the possibility of diagrammatic reasoning.**

I. Introduction

Visualization helps to work with ontologies and to study their features, because representation of an ontology as a set of graphic primitives is perceived much easier than textual information. Currently, there is a large number of ontology visualization services [1]–[3], However, they are mainly focused on the expressiveness and readability of graphical representation [4], [5].

Compare VOWL (Visual Notation for OWL Language) [6] and OnotoGraf [7] or any other most popular ontology visualization services such as with Peirce's existential graphs [8]. The existing ontology visualization services only *denote* logical operations using arbitrary graphical primitives, which, in fact, is just replacing one language with another, while existential graphs reflect the semantics of operations. The latter characteristics of existential graphs, firstly, actually facilitates the perception of ontologies by users, and secondly, provides the possibility of diagrammatic reasoning.

However, existential graphs are defined for predicate logic, while ontologies are based on description logics. In this regard, based on existential graphs, we have developed a new visual language for ontologies, and called it Logic Graphs [9], [10]. It is complete with respect to the OWL DL ontology description language and reflects the semantics of logical operations in ontologies. Until now, there has been no tools for visualization of ontologies in the Logic Graphs language. This article presents a method for generating visual representations of ontologies in Logic Graphs and an application that implements this method.

As a result, unlike the existing ontology visualization tools, our solution allows i) visualizing complex logical expressions, and ii) representing semantics of logical relations. Due to correct and semantical visualization of logical expressions, our solution facilitates the usage of ontologies both by non experts, because perceiving semantics of logical relations requires no specific knowledge, and by ontology experts, because it provides the possibility of diagrammatic reasoning over ontologies. Our solution is especially relevant for ontologies that are rich in complex axioms. Such ontologies are used, for example, in requirements modelling.

II. Related work

A. Existing ontology visualization tools

There are numerous ontology visualization services. They are reviewed in [1]–[3]. For each tool, we examined its ability to represent the semantics of logical operations. Consider several most popular services to demonstrate the analysis performed.

One of the most popular ontology visualization languages is VOWL [6]. According to its authors, VOWL is a well-specified visual language for a user-oriented representation of ontologies. It defines graphical depictions for most elements of OWL language that are combined to a force-directed graph layout. VOWL aims for an intuitive and comprehensive representation that is understandable for users that are not ontology experts. VOWL is implemented as two tools: ProtégéVOWL [11] – a plugin for the ontology editor Protégé [12], and WebVOWL [13] – a standalone web-application.

Consider an axiom (1), which states that a vegan and not a vegetarian is a person that eats only plants and not a person that eats only plants or dairy. Fig. 1 provides its visualization in WebVOWL. As can be seen, WebVOWL does not seem to represent this axiom correctly. First, it is not full, as it does not represent the elements of the intersection of the axiom except for the first one, second, it does not represent the semantics of this intersection itself, as the intersection is represented only with a pictogram of a Venn diagram for intersection. Only a person that is familiar with Venn diagrams can understand there is an intersection, but even this kind of person can not define which classes are intersected.

Another visualization service, OntoGraf [7], provides various automated layouts for the structure of ontology visualization, supports a number of relationships: subclass, individual, domain/range of object properties, and equivalence. It is implemented as a Protégé plugin. Fig 2 provides a visualization of the axiom (1) in OntoGraf. Again, it does not represent the axiom correctly, because it does not depict the intersection. Moreover, it misleadingly claims that a vegan and not a vegetarian is equivalent to a plant.

$$Vegan\ and\ not\ vegetarian \equiv Person \sqcap \forall eats.Plant \sqcap \neg(Person \sqcap \forall eats.(Plant \sqcup Dairy)), \qquad (1)$$

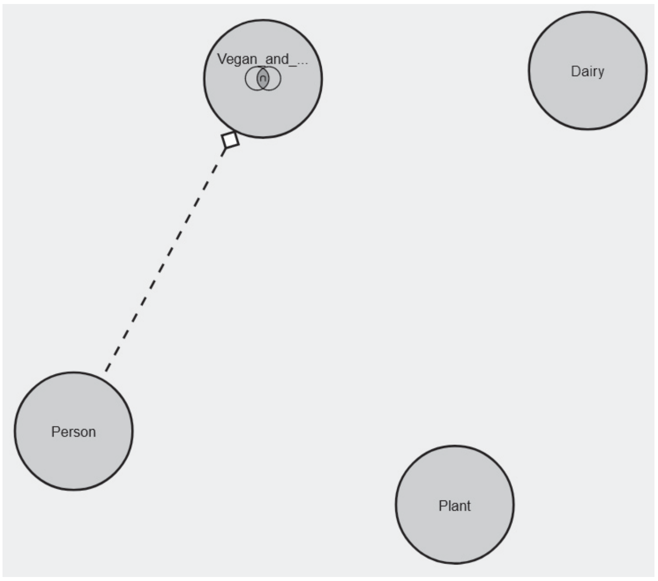


Fig. 1. Axiom (1) in WebVOWL

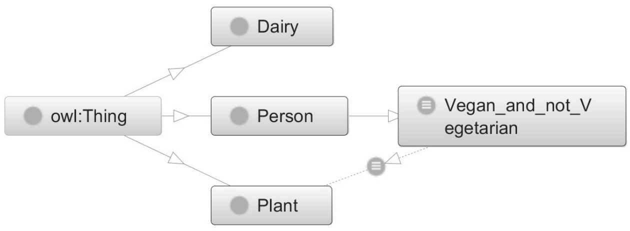


Fig. 2. Axiom (1) in OntoGraf

We are not going to represent the whole survey, as it is outside the scope of this research. The result is, no ontology visualization tools have been found that would properly represent logical operations.

B. *Semantic-oriented visual languages*

Unlike existing ontology visualization tools, there are semantic-oriented visual languages for representing logical expressions. As an example, consider Ch. S. Pierce's existential graphs (EG) [8]. In an EG, there are propositions placed on a sheet of assertion. A proposition can be negated by placing it into a shaded area. If two propositions are on the sheet of assertion, they are both true, therefore, their conjunction is also true. Having a negation and a conjunction makes it possible to express any complex proposition. Thus, the graphic primitives of existential graphs represent the semantics of propositional operations. As an example, consider an existential graph for the expression "If it rains, it is cold" in Fig. 3. It states that the situation when it rains, but it is not cold is impossible.

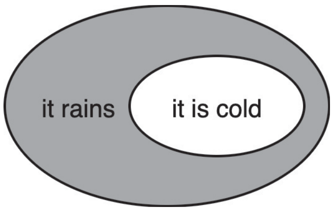


Fig. 3. Existential graph for "If it rains, it is cold"

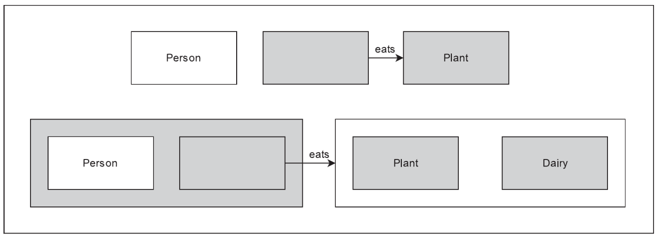


Fig. 4. Axiom (1) in Logic Graphs

Based on existential graphs, we developed a visual language for ontologies, and called it Logic Graphs (LG) [9], [10]. It is semantic-oriented, similar to existential graphs, and complete with respect to OWL DL language. The syntax of LG for a fragment of OWL language corresponding to ALC description logic is provided in Table I, where C, C_1, C_2 are concepts, R – a role, and a and b are individuals.

Similar to the existential graphs, the space on which the graph is located is called the assertion space, it denotes the universe of objects. A rectangle denotes a concept, that is, a set of objects of the universe that have a certain property. Shading denotes negation – a set of objects that do not have the specified property. Concepts can be nested within each other. The outer rectangle denotes a set of objects that have the properties of all inner rectangles. For details about representation of other operations see [10].

As an example, consider representation of the axiom (1) in Logic Graph in Fig 4. It reflects that a vegan and not a vegetarian is an intersection of a person, of a nothing that eats anything except for plants, and of a nothing that is a person and does not eat anything, which is not plant, or dairy.

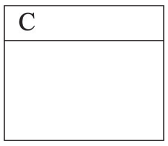
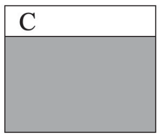
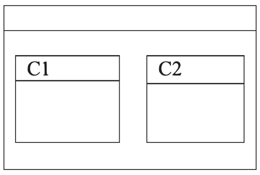
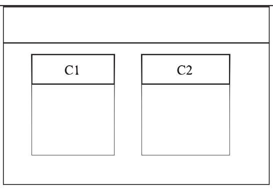
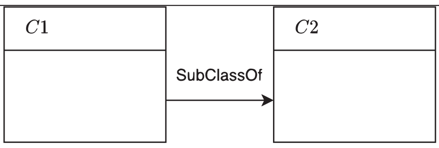
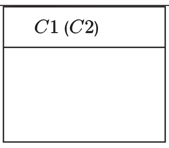
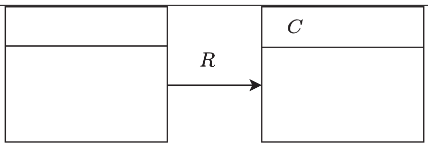
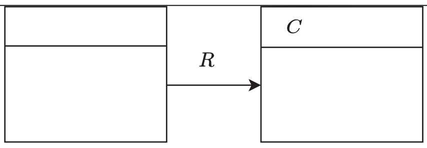
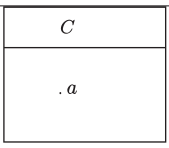
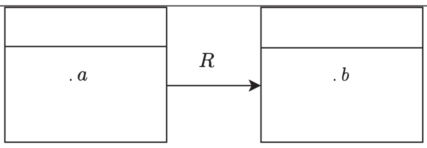
However, for now, there is no tools for visualization of ontologies in the Logic Graphs language.

III. Generating Logic Graphs for ontologies

This section describes the method of generating LG for ontologies we developed and its software implementation. The method consists of three main steps:

1) ontology parsing;
2) conversion;
3) graph building.

TABLE I. LOGIC GRAPHS FOR ALC

1	C		2	$\neg C$	
3	$C1 \sqcap C2$		4	$C1 \sqcup C2$	
5	C1 $\sqsubseteq$ C2		6	C1 $\equiv$ C2	
7	$\exists R.C$		8	$\forall R.C$	
9	$a : C$		10	$(a, b) : R$	

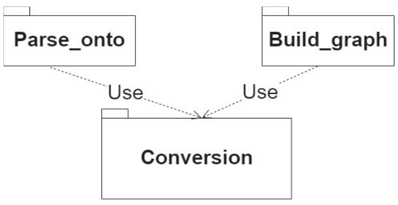


Fig. 5. Package diagram

Further subsections discuss more details about each of the steps.

The architecture of the application can be seen in the package diagram in Fig. 5. As can be seen, it matches the main steps of the method proposed. Python was used as the development language.

A. Ontology parsing

The first stage is ontology parsing. The parser extracts data from an ontology and transform it into Python classes. For class diagram of the application see Fig. 6.

The data extraction pipeline is the following:

1) The parser receives an ontology in rdf format as input.
2) Ontology classes are extracted using regular expressions.
3) Class properties including logical operations are extracted from the class with the depth-first search algorithm (DFS) [14]. An example of extracting data from a class of the Pizza ontology [15] is shown in Fig. 7.
4) After extracting data from the ontology, instances of the corresponding Python classes are created.

For more details, see diagram in Fig. 8.

The depth-first search algorithm allows building a traversal of a directed or undirected graph, which places all vertices accessible from the initial vertex. The idea is to move from the starting vertex in a certain direction, along a certain path, until we reach the end of the path (the desired vertex). If the end of the path is reached, but it is not the desired destination, then we return back (to the point of divergence of the paths) and follow a different route. In this way, all elements and properties of an ontology class are extracted.

B. Conversion

This component is intended to convert instances of the Python class describing classes of an ontology to instances of the Python class describing nodes of Logic Graphs and further, to nodes descriptions in DOT format [16]. The description of the Python class of nodes is shown in the listing below:

```
class Node:
    def __init__(self, name,
        flag_negation=False):
         self.name = name
         self.node = 'node_' + str(
             NumberElementStatic.node_cnt)
         NumberElementStatic.node_cnt += 1
         self.flag_nested = False
```

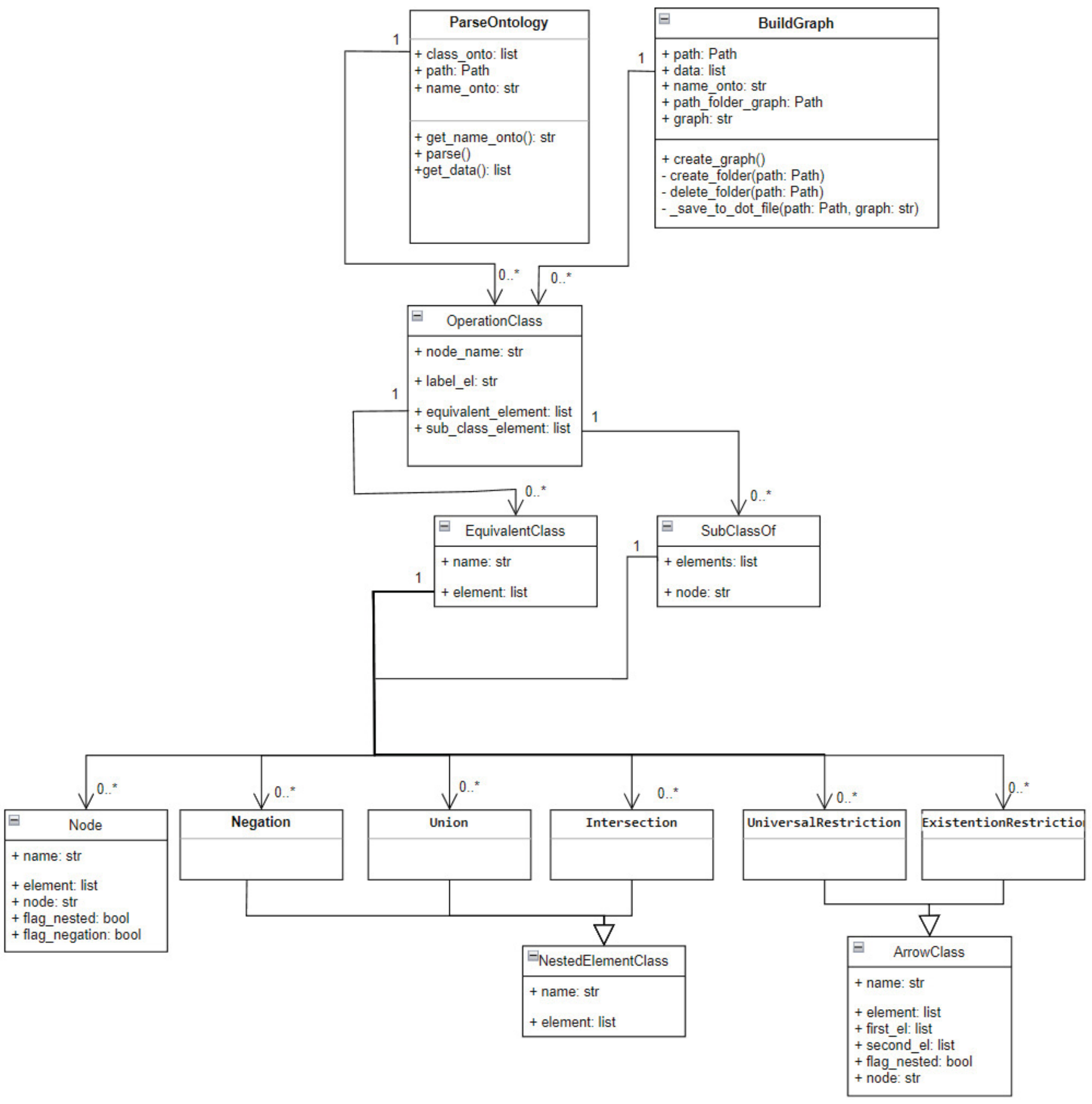
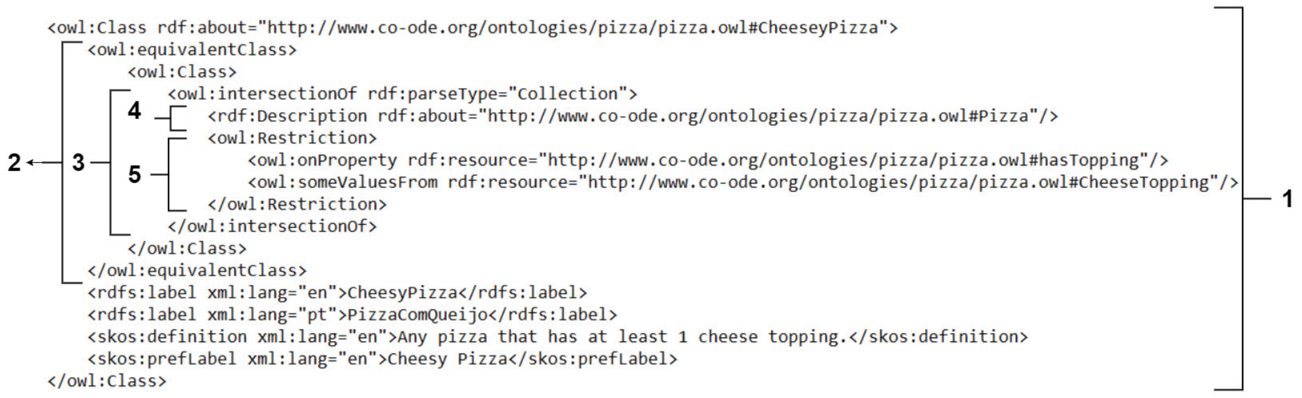


Fig. 6. Class diagram



```
<owl:Class rdf:about="http://www.co-ode.org/ontologies/pizza/pizza.owl#CheeseyPizza">
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <rdf:Description rdf:about="http://www.co-ode.org/ontologies/pizza/pizza.owl#Pizza"/>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="http://www.co-ode.org/ontologies/pizza/pizza.owl#hasTopping"/>
                    <owl:someValuesFrom rdf:resource="http://www.co-ode.org/ontologies/pizza/pizza.owl#CheeseTopping"/>
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
    <rdfs:label xml:lang="en">CheesyPizza</rdfs:label>
    <rdfs:label xml:lang="pt">PizzaComQueijo</rdfs:label>
    <skos:definition xml:lang="en">Any pizza that has at least 1 cheese topping.</skos:definition>
    <skos:prefLabel xml:lang="en">Cheesy Pizza</skos:prefLabel>
</owl:Class>
```

Fig. 7. Parsing CheesyPizza class of the Pizza ontology

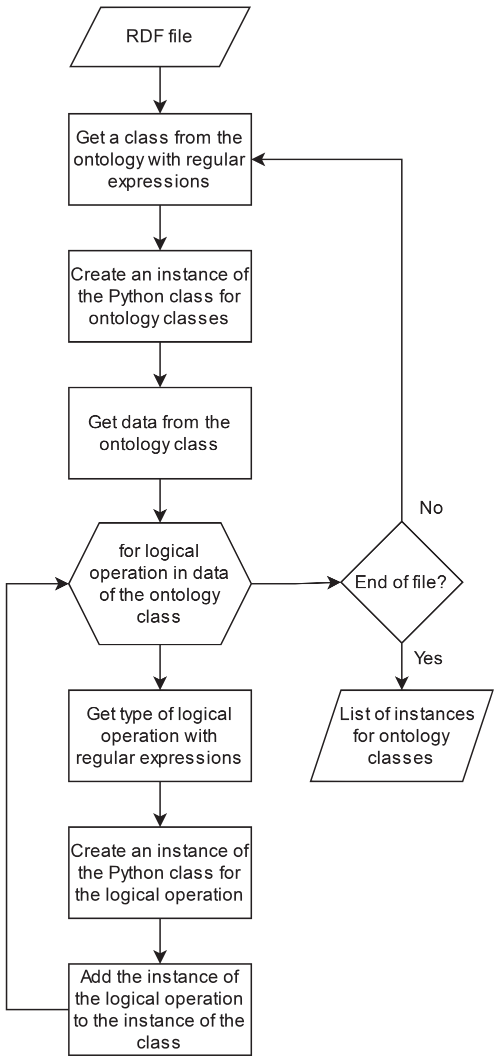

RDF file
Get a class from the ontology with regular expressions
Create an instance of the Python class for ontology classes
Get data from the ontology class
for logical operation in data of the ontology class
End of file?
No
Yes
List of instances for ontology classes
Get type of logical operation with regular expressions
Create an instance of the Python class for the logical operation
Add the instance of the logical operation to the instance of the class

Fig. 8. Parsing algorithm

```
        self.flag_negation =
            flag_negation

    def __str__(self):
        return 'Node␣' + self.name
```

As can be seen, this class contains properties of a Logic Graphs node: an original name of the ontology class, a unique name of the node, which is used to generate the DOT file, a flag that indicates whether the node is nested into another node, and a flag that indicates whether the complement operation is applied to the node.

Regarding conversion of ontology classes to Logic Graphs nodes, logical operations are divided into three types, which are given in Table II. The CheesyPizza class of the Pizza ontology is used to illustrate conversion. As a result, this component outputs the descriptions of the ontology classes and their logical operations in DOT format.

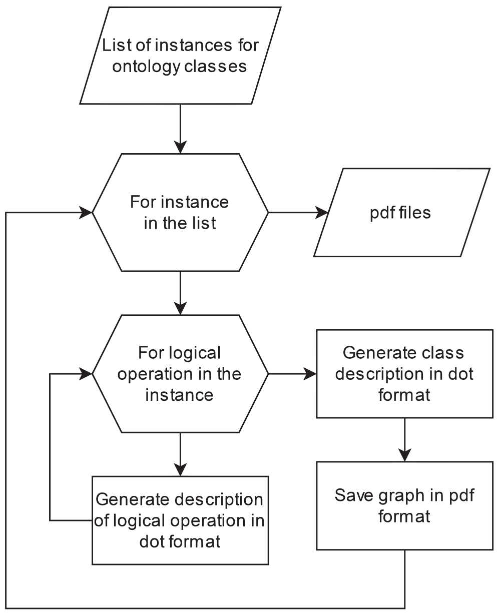

List of instances for ontology classes
For instance in the list
pdf files
For logical operation in the instance
Generate class description in dot format
Generate description of logical operation in dot format
Save graph in pdf format

Fig. 9. Graph building algorithm

C. Graph building

Graphs representing classes of an ontology in the Logic Graphs language are formed by generating graphs from their descriptions in the DOT format using recurrent traversal. The algorithm for constructing the graphs is as follows:

1) reading a DOT file that was obtained in the previous step;
2) passing the file data to the Graphviz [17] library to build a graph;
3) saving the resulting graph in PDF file.

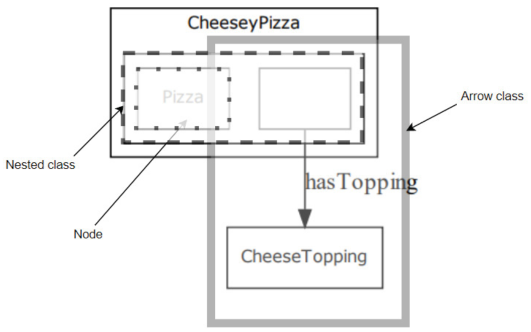
For more details, see diagram in Fig. 9.

As can be seen, the algorithm for constructing a graph is quite simple. This is due to the Graphviz graph visualization library that has rich functionality. It is sufficient to load the correct description of the graph in DOT format to get the visualization. Also, the functionality of this library allows drawing nested nodes, which is one of the key reasons for using this library, since the Logic Graphs language contains nested elements.

As an example of the resulting graph, consider the class 'American' from the Pizza ontology, see Fig. 10. The most demonstrative is representation of the axiom (2). The graph for the american pizza class reflects a universal restriction and disjunctions from the axiom, showing that american pizza is not a subclass of objects that have no tomatto, peperoni sausage or mozzarella toppings. Thus, our solution generates correct visualizations of ontologies on the Logic Graphs language. Considering computational costs, it required 8.64 seconds with AMD Ryzen 5 2600 and 10.03 MB RAM to generate all graphs for the pizza ontology, containing 801 axioms.

TABLE II. CONVERSION OF ONTOLOGY CLASSES TO LOGIC GRAPHS NODES

Type	Example of description in DOT	Example of graphic representation

Node (Class)	node_100[
	label="Pizza",
	  style=filled ,
	  fillcolor=white ,
	  color=black
	]
		 CheeseyPizza
		Pizza
		Arrow class
		hasTopping
		Nested class
		Node
		CheeseTopping

Arrow (Universal restriction, Existential restriction, Subclass)	node_101->node_102 [
	    ltail=node_102
	    lhead=node_102
	    label="hasTopping"
	]	

Nested node (Intersection, Union)	subgraph cluster_87 {
	    node [
	        shape=record];
	        style=filled ;
	        label="";
	        color=black ;
	        fillcolor=white ;
	    ];
	    node_100 [
	        label="Pizza",
	        style=filled ,
	        fillcolor=white ,
	        color=black
	    ];
	    node_101[
	        label="",
	        style=filled ,
	        fillcolor=white ,
	        color=black
	    ];
	}	

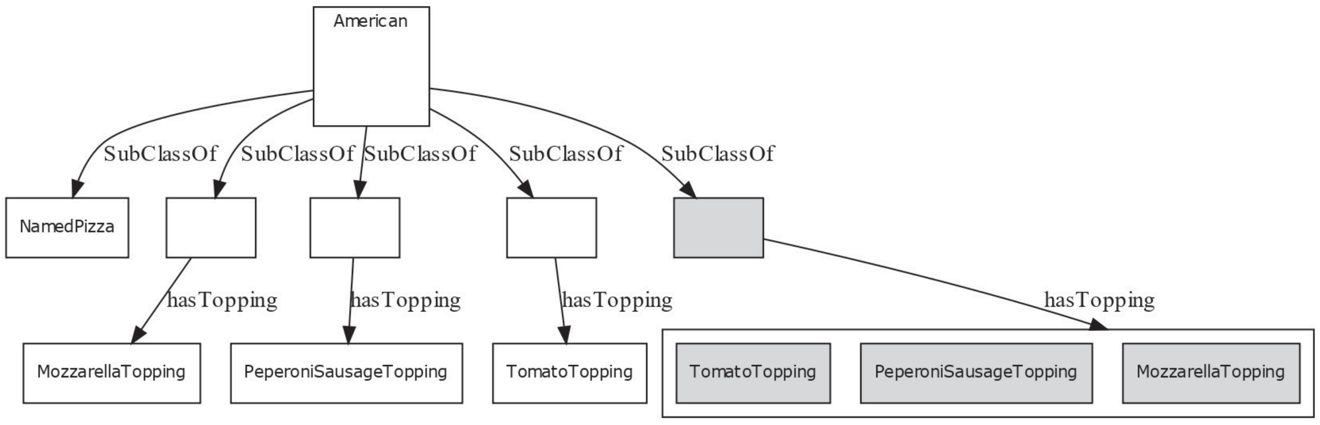


Fig. 10. Generated graph for American pizza

$$American \sqsubseteq \forall hasTopping.(MozzarellaTopping \sqcup PeperoniSausageTopping \sqcup TomatoTopping) \quad (2)$$

IV. Conclusion

We analyzed the existing ontology visualization tools and concluded that there is no tool that represents the semantics of logical operations in ontologies, as Ch. S. Pierce's existential graphs do for first order logic. Therefore, we developed a tool for generating a visualization of ontologies in Logic Graphs – a semantic-oriented language, complete with respect to OWL DL. The workflow of this tool consists of three steps: i) ontology parsing for extracting ontology classes and their properties, ii) conversion of ontology classes to their descriptions in DOT format, iii) and building graphical representations from DOT descriptions. As a result, our application generates a PDF file with graphical representation on the Logic Graphs language for each class in an ontology. These visualizations facilitate the users perception of ontologies, and provide the possibility of diagrammatic reasoning.

References

[1] F. Antoniazzi and F. Viola, "Rdf graph visualization tools: a survey," in *2018 23rd Conference of Open Innovations Association (FRUCT)*, 2018, pp. 25–36.

[2] M. Dudáš, S. Lohmann, V. Svátek, and D. Pavlov, "Ontology visualization methods and tools: a survey of the state of the art," *The Knowledge Engineering Review*, vol. 33, 07 2018.

[3] M. F. Joseph and R. Lourdusamy, "Feature analysis of ontology visualization methods and tools," *Computer Science and Information Technologies*, vol. 1, no. 2, 2020.

[4] I. Baimuratov and T. Nguyen, "Non-empirical metrics for ontology visualizations evaluation and comparing," *CEUR Workshop Proceedings*, vol. 2744, 2020.

[5] I. Baimuratov, T. Nguyen, R. Golchin, and D. Mouromtsev, "Developing non-empirical metrics and tools for ontology visualizations evaluation and comparing," *Scientific Visualization*, vol. 12, no. 4, pp. 71–84, 2020.

[6] S. Lohmann, S. Negru, F. Haag, and T. Ertl, "Visualizing ontologies with VOWL," *Semantic Web*, vol. 7, no. 4, pp. 399–419, 2016.

[7] *OntoGraf*. [Online]. Available: https://protegewiki.stanford.edu/wiki/OntoGraf

[8] C. Peirce and J. Sowa, "Existential graphs: Ms x 514 by charles sanders peirce with commentary by john f. sowa."

[9] D. Mouromtsev and I. Baimuratov, "Logic graphs: A complete visualization method for logical languages based on ch. s. peirce's existential graphs," *CEUR Workshop Proceedings*, vol. 2344, no. 12, pp. 1–10, 2019.

[10] T. Nguyen and I. Baimuratov, "Logic graphs: Complete, semantic-oriented and easy to learn visualization method for owl dl language," *CEUR Workshop Proceedings*, vol. 2893, 2020.

[11] *ProtégéVOWL*, vOWL Plugin for Protégé. [Online]. Available: http://vowl.visualdataweb.org/protegevowl.html

[12] *Protégé*, web-based Visualization of Ontologies. [Online]. Available: https://protege.stanford.edu/

[13] *WebVOWL*, web-based Visualization of Ontologies. [Online]. Available: http://vowl.visualdataweb.org/webvowl.html

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, ser. Second Edition. MIT Press and McGraw-Hill, 2001, a Wiley-Interscience Publication, *(Example for books)*.

[15] *Pizza*. [Online]. Available: https://protege.stanford.edu/ontologies/pizza/pizza.owl

[16] *DOT*, graph description language. [Online]. Available: https://graphviz.org/doc/info/lang.html)

[17] *Graphviz*. [Online]. Available: https://graphviz.org/