ISSN 0361-7688, Programming and Computer Software, 2020, Vol. 46, No. 6, pp. 384–396. © Pleiades Publishing, Ltd., 2020.
Russian Text © The Author(s), 2020, published in Programmirovanie, 2020, Vol. 46, No. 6.

Approach to Blank Node Processing in Incremental Data Visualization by the Example of Ontodia

D. S. Razd'yakonov[a,*], A. V. Morozov[a,], D. S. Pavlov[b,***], and D. I. Muromtsev[a,****]**

[a] *Faculty of Software Engineering and Computer Technologies, St. Petersburg National Research University of Information Technologies, Mechanics, and Optics (ITMO University), Kronverkskii pr. 49, St. Petersburg, 197101 Russia*

[b] *Metaphacts East Europe, Bol'shaya Raznochinnaya ul. 14, St. Petersburg, 197110 Russia*

*e-mail: ladone3@mail.ru

**e-mail: am@metaphacts.com

***e-mail: dp@metaphacts.com

****e-mail: mouromtsev@mail.ifmo.ru

Received April 5, 2020; revised June 16, 2020; accepted July 9, 2020

Abstract—The problem of lazy visualization of ontological graphs has certain constraints. Visualization of structures that contain blank nodes is especially challenging. In this paper, we propose an approach to visualization of these structures that is implemented in Ontodia, analyze the limitations of this approach, and consider some alternative solutions.

DOI: 10.1134/S0361768820060067

1. INTRODUCTION

Researchers working in the field of semantic technologies are very well familiar with blank nodes (BNs). They are used to represent objects that do not have internationalized resource identifiers (IRIs), e.g., to represent a class that is a union of several classes or represent a collection. When representing unions, e.g., the construct `owl:unionOf`, the resulting class is a BN that refers to an RDF list, a structure in which all intermediate elements, including the root one, are BNs, while its enumerated elements are ordinary nodes with readable identifiers (see Fig. 1). The same is true when representing the construct `owl:Restriction` (see Fig. 2): a BN has no meaning without specifying which property is restricted, as well as without value restriction.

The absence of IRIs for BNs is well justified; however, from a technical perspective, this is a hindrance, because the absence of IRIs makes it impossible to refer to these nodes. It turns out that BNs cannot be considered separately from the structure that includes them; however, in some cases, this separation is strictly necessary.

This necessity arises, e.g., when visualizing a graph with the possibility of saving and retrieving its visualization to/from a file or other artifact. The problem of visualizing graphs with BNs is not very complex if we receive the entire structure in one bundle. However, in the case of Ontodia [1], we are faced with lazy visualization whereby data are sent incrementally in batches; in addition, it is not known which data were visualized and which were not. In this case, the absence of IRIs for BNs significantly complicates the process. With

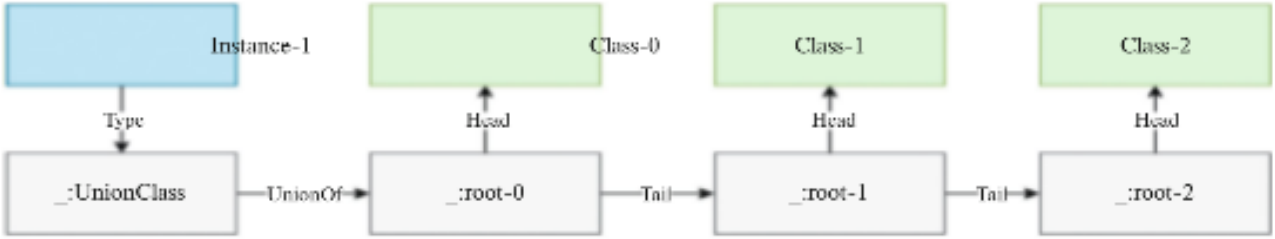


Fig. 1. Representation of the RDF list.

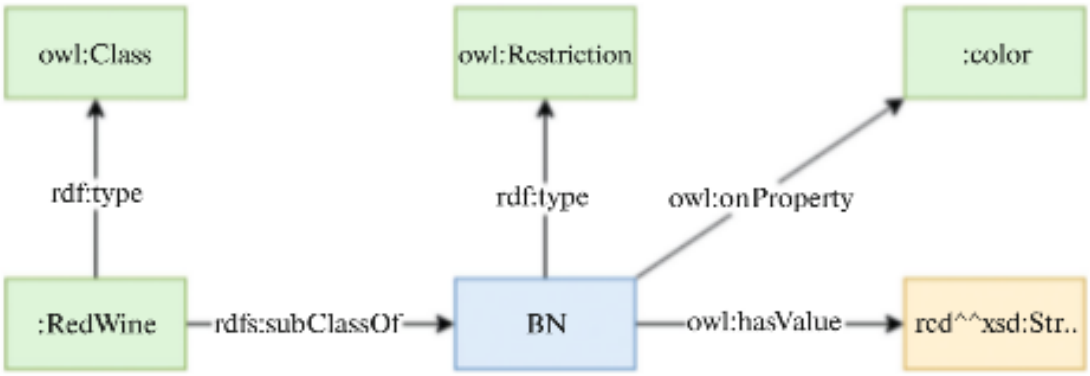


Fig. 2. Representation of `owl:Restriction`.

lazy visualization, we cannot definitively say whether a node sent with a new batch is the same node that has already been displayed on a diagram, or it is some other node that has the same set of links and the same type. This problem also arises when we try to retrieve a previously saved visualization, where it does not matter whether lazy loading is used or the visualization was built of a single batch.

In this paper, we demonstrate how Ontodia solves the problem of visualizing ontological graphs with BNs under conditions of lazy loading, as well as the problem of saving and retrieving this visualization (diagram).

Visualization in general and the proposed approach in particular can be useful in a number of tasks associated with logical expression flows, e.g., those considered in the papers "Knowledge visualization based on a semantic network" [2] and "A method for acceleration of logical inference in the production knowledge model" [3].

2. EXISTING APPROACHES

Hereinafter, we use the data shown in Listing 1. The data are part of the wine ontology (`http://www.w3.org/TR/owl-guide/wine.rdf`). For illustrative purposes, one of the links is duplicated and its identifier is replaced with a non-standard one: `owl:intersectionOf` → `<http://example.com/unknownTypeOfProperty>`. This is done to demonstrate how visualization tools handle non-standard cases.

There are several approaches to BN processing. Some approaches directly solve the problem of lazy visualization of ontological graphs; other approaches also solve the problem of comparing graphs with BNs while pursuing other goals. Ontop Protégé [4] imposes a constraint on BN properties: each BN must have a literal property `rdfs:label`, which obviously acts as a stable identifier. Another plugin, OntoGraph Protégé, makes it possible to visualize BNs, yet in a non-standard way. In OntoGraph, BNs are visualized as annotations (see Fig. 3a), which are represented as pop-up windows, with the construct `owl:Restriction` being represented as a `Superclasses` property and `owl:IntersectionOf` being represented as `Equivalent classes`. Non-standard structures are visualized in OntoGraph Protégé in a different way. In Fig. 3b, it can be seen that a non-standard link to an RDF list is included in a separate `Annotations` block; then, only the first BN of the list is displayed.

Some tools, e.g., Graffoo [5], do not support BN visualization and discard BNs in the process of rendering. When visualizing ontologies that contain BNs, WebVOWL [6] implements an individual approach to visualization of supported structures; however, the list of these structures is short. For instance, WebVOWL represents the construct `owl:intersectionOf` as a star topology, where the central node is an intersection of other topology classes (see Fig. 4), while almost ignoring the constructs `owl:Restriction` and displaying only the links to the basic entity `owl:Thing`. The non-standard cases are ignored. It should also be noted that WebVOWL focuses more on visualization of classes rather than on class instances.

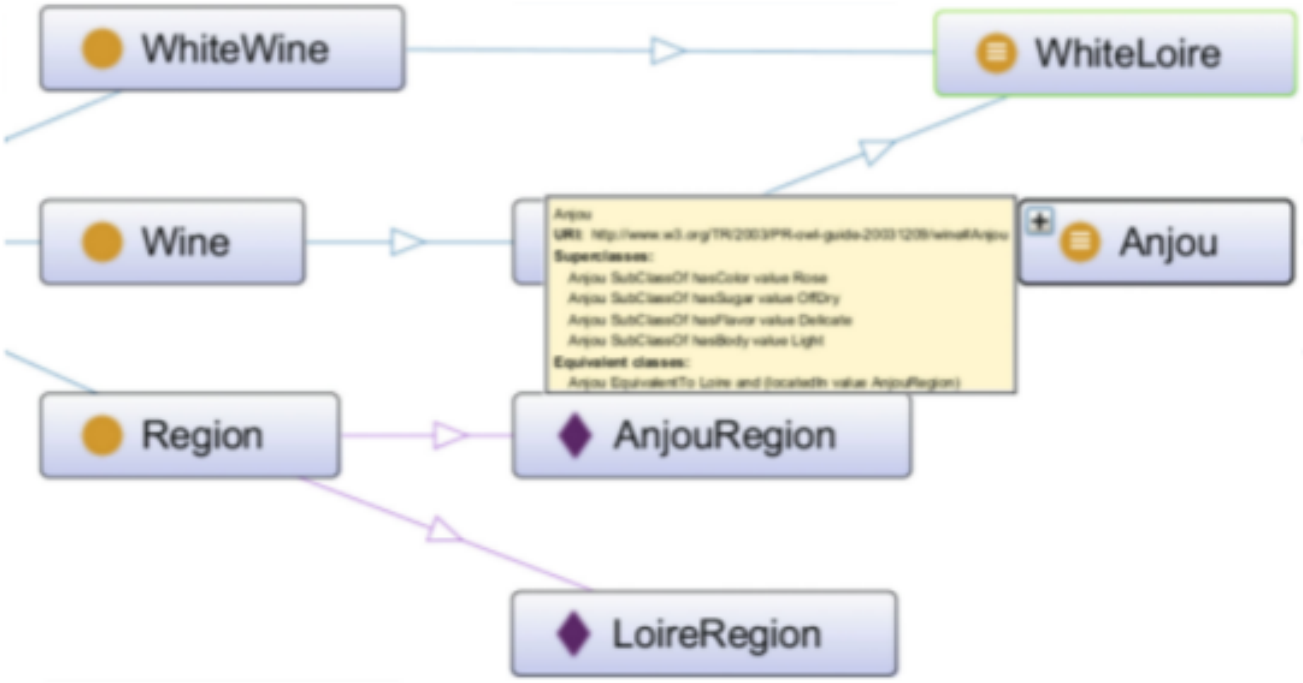
A similar approach is implemented in OWLGrEd [7]. The construct `owl:intersectionOf` is visualized as a tree. The construct `owl:Restriction` is displayed as an annotation in the body of a target element (see Fig. 5a). In contrast to WebVOWL, OWLGrEd can render class instances while highlighting them in green; however, it does not cope with non-standard cases. The constructs of the RDF list type are ignored unless they are part of the construct `owl:IntersectionOf` (see Fig. 5b).

The PHP library EasyRdf [8] uses another approach.

In the process of visualization, EasyRdf displays the structure of the ontological graph as accurately as possible; however, it does not support individual visualization of common structures (see Fig. 6a). Figure 6a shows only a part of the result because the entire result is too cumbersome and EasyRdf does not support visualization editing. The entire graph is immediately

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix wines: <http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#> .

<http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine>  a owl:Ontology ;
  rdfs:label "Wine Ontology" .

wines:Anjou a owl:Class ;
  rdfs:subClassOf [ a owl:Restriction ; owl:onProperty wines:hasColor ; owl:hasValue wines:Rose ],
   [ a owl:Restriction ; owl:onProperty wines:hasBody ; owl:hasValue wines:Light ],
   [ a owl:Restriction ; owl:onProperty wines:hasFlavor ; owl:hasValue wines:Delicate ],
   [ a owl:Restriction ; owl:onProperty wines:hasSugar ; owl:hasValue wines:OffDry ] ;
  owl:intersectionOf ( wines:Loire  _:blank6 ) .

wines:Loire a owl:Class .
_:blank6 a owl:Restriction ; owl:onProperty wines:locatedIn ; owl:hasValue wines:AnjouRegion .

wines:Loire a owl:Class ; owl:intersectionOf ( wines:Wine _:blank9 ) .
wines:Wine a owl:Class .
wines:LoireRegion a wines:Region ; wines:locatedIn wines:FrenchRegion .

wines:AnjouRegion a wines:Region ; wines:locatedIn wines:LoireRegion .
_:blank6 a owl:Restriction ; owl:onProperty wines:locatedIn ; owl:hasValue wines:AnjouRegion .
_:blank9 a owl:Restriction ; owl:onProperty wines:locatedIn ; owl:hasValue wines:LoireRegion .

wines:WhiteLoire a owl:Class ; owl:intersectionOf ( wines:Loire wines:WhiteWine ) ;
      <http://example.com/unknownTypeOfProperty> ( wines:Loire wines:WhiteWine ) .
wines:WhiteWine a owl:Class .
```

Listing 1. Representation of owl:IntersectionOf and owl:Restriction in the Turtle format.



(a) `owl:IntersectionOf` and `owl:Restriction`

(b) The nonstandard case, the RDF list construct RDF list

Fig. 3. Representation of constructs in OntoGraph Protégé.

placed in the browser's memory, which makes it possible to refer to BNs and visualize target structures. An advantage of this approach is that non-standard cases are displayed exactly as they were defined (see Fig. 6b). Considering the examples of OWLGrEd and WebVOWL, it should be noted that the individual approach has its advantages in terms of visualization; however, by combining it with the global approach (as in EasyRdf), we obtain efficient visualization of common structures together with universal visualization.

This combined approach was implemented in TopBraid Composer [9] (see Figs. 7a and 7b). It can be seen that TopBraid uses the individual approach to visualize `owl:IntersectionOf` and `owl:Restriction` (see Fig. 7a) while displaying the restrictions and intersections in the body of the elements. TopBraid uses the same approach to visualization of RDF lists, whereby list items are sequentially displayed in the body of the list head. In addition, this tool can display the entire structure while visualizing BNs as a set of RDF lists linked by rdf:rest (see Fig. 7b). In this case, the whole data are loaded into the application's memory and are indexed depending on a problem to be solved, which allows the graph to be visualized in portions. The entire model is stored in the memory of the user's machine, which is why visualization is limited by its resources. Large knowledge bases like, for example, Wikidata and DBpedia, cannot be visualized in such a way.

Similarly to TopBraid, Ontodia tries to combine the two approaches with the individual approach in its current implementation being relatively limited, supporting only RDF list processing, and the other constructs being visualized using the global approach (see Fig. 8a). It can be seen from the figure that, for all elements of the RDF list, additional links to the list head are created with the list index property on each link. The same is done for the list tail, which, in turn, is a separate list. This approach enables efficient visualization of non-standard cases (see Fig. 8b). It is important to stress that this tool implements lazy loading

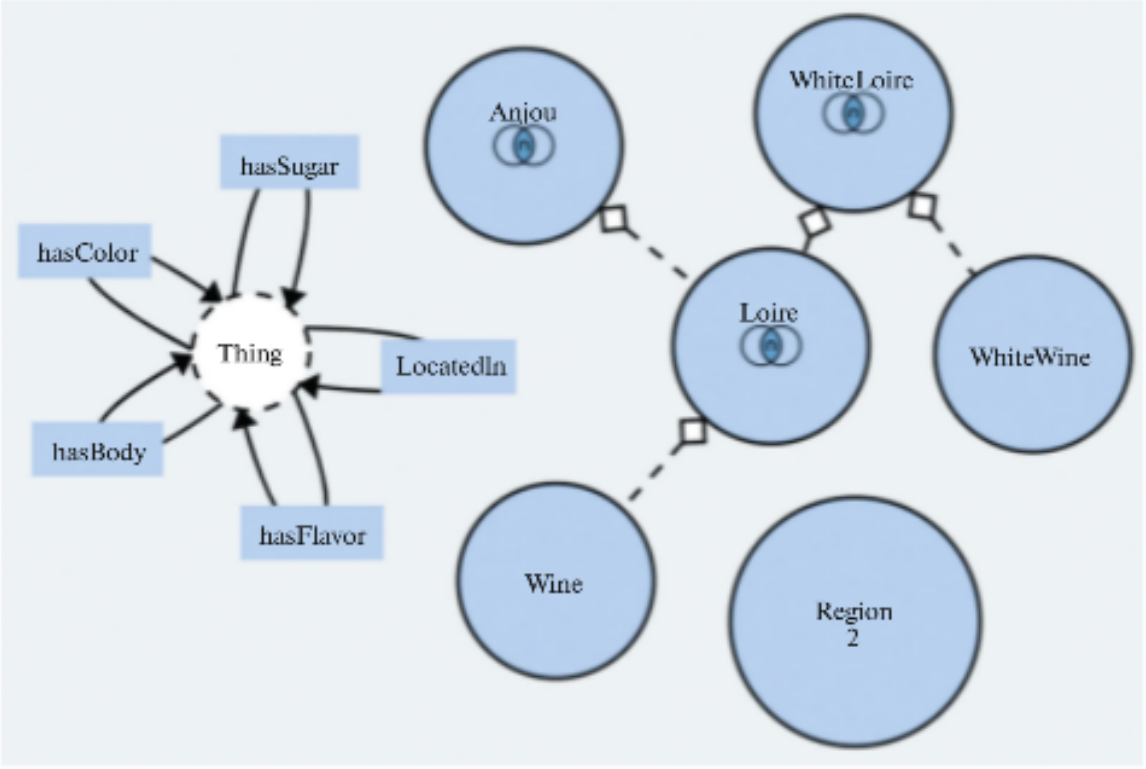


Fig. 4. Constructs `owl:Restriction` and `owl:IntersectionOf` in WebVOWL.

(only the visualized part of an ontological graph is loaded), which enables big data processing.

Taking into account the above discussion, Table 1 summarizes the approaches used by these tools to visualize structures with BNs.

3. DESCRIPTION OF THE PROPOSED SOLUTION

3.1. Data Circulation in Ontodia

Lazy loading of graph data in Ontodia is carried out using the `DataProvider` object, which implements the idea of the *data access object* pattern [10]. `DataProvider` is responsible for fetching the scheme and data from the basic storage, as well as for translating the loaded data into the Ontodia's internal model. In addition, at the data translation stage, the data structure can be transformed depending on the needs of a particular application. This includes, for example, grouping individual nodes into tables in the user interface, grouping nodes into supernodes, and collapsing paths between them. Moreover, `DataProvider` supports lazy and incremental loading on request.

Possible data queries to `DataProvider` are listed in Table 2. Most of the queries are used in Ontodia-specific cases and serve to optimize data circulation in this visualization tool for efficient display of its interface elements.

Hereinafter, by additional information, we mean the information that does not affect the topological structure of an ontological graph (it includes names of elements, types of elements and links, and literal properties of elements).

3.2. General Description of the Solution

BN visualization in Ontodia consists in preprocessing queries and assigning context-dependent identifiers to BNs. The preprocessing stage includes several steps: collecting the context, generating context-dependent identifiers, and saving the preprocessing results for subsequent output. The identifiers must be composed in such a way as to enable restoration of the context directly from each identifier generated, as well as ensure unambiguous comparison of the nodes.

The context is defined as follows: *the context for a target BN is a subgraph of the main graph that includes the target BN and transitively all BNs between non-blank nodes (NBNs) that surround this subgraph and are contained in it (see Fig. 9)*. Thus, the context for a target BN is a graph that contains the target BN, all of its neighbors, and neighbors of their neighbors, up to the first NBN.

Below, we describe our method in more detail by answering the following questions:

1. When and where is data preprocessing carried out?
2. How is the context collected?
3. How is the context-dependent identifier created?
4. Where and when are the results output?

(a) `owl:IntersectionOf` and `owl:Restriction`

(b) The nonstandard case, the RDF list construct RDF list

Fig. 5. Representation of constructs in OWLGrEd.

3.3. Where is Data Preprocessing Carried Out?

Data exchange between `DataProvider` and the diagram is limited to the queries that the diagram sends to `DataProvider` to fetch links and elements (where elements are classes and their instances), as well as additional information about elements and links. Additional information queries are as follows:

- query for all possible types of links in the graph, regardless of particular nodes;
- query for properties of classes and elements, also known as `DataProperty`;
- query for properties of links.

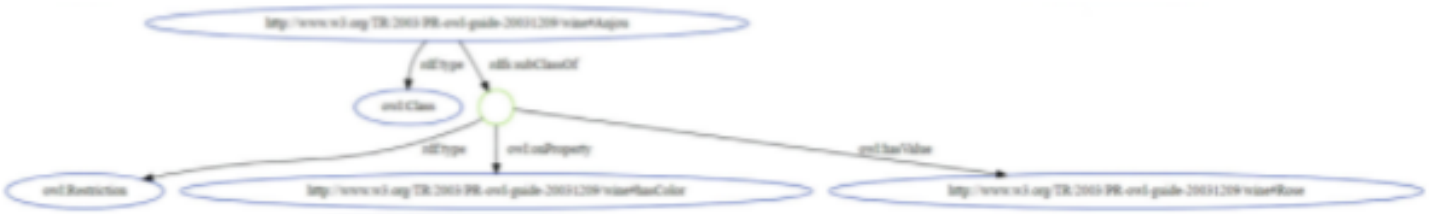
To solve the problem formulated above, we focus on the queries associated with nodes, because the additional information does not affect the structure of the ontological graph and the links in the case of BNs are encoded (together with BN contexts) in their IRIs. From the set of queries used by Ontodia (see Section 3.1), we single out those that operate exclusively with graph vertices:

- classTree: query for a tree of classes/types;
- filter: query for a list of nodes with search parameter specification.

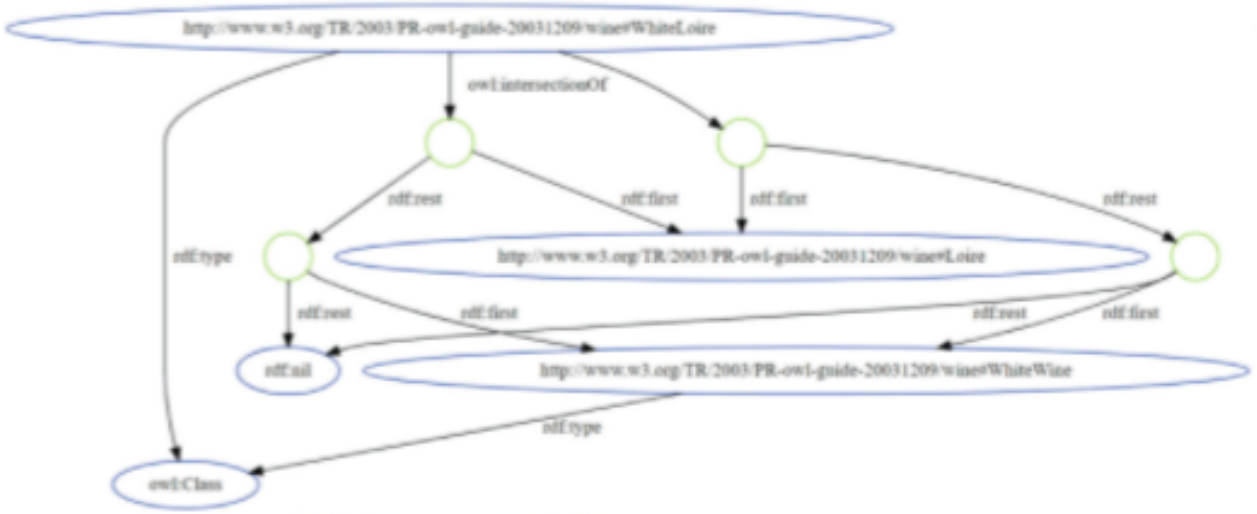
The other queries either return information about links or operate with additional information. When developing our approach, we decided that the tree of classes must not include BNs because BNs usually do not have readable identifiers and meaning outside the structure that contains them. Hence, the only point at which data preprocessing can be performed is the filter query. This query is used in Ontodia wherever it is required to obtain a list of elements (only filtering parameters are varied). Data preprocessing is carried out immediately upon receiving the query result. The result is an array of triplets [subject, predicate, object]. Before returning the result, the triplets that contain BNs in an object or subject position are selected to be processed individually (see Section 3.4).

3.4. Context Retrieval

The context is retrieved by recursively forming a SPARQL query and then executing it. Suppose that the function `filter DataProvider` returns a BN. The result is fed to the preprocessing pipeline. Here, it should be clarified that the `filter` function, in most cases, carries out search depending on a target element, i.e., it searches for the neighbors of the target element or filters the list of possible links to it. The only exception is keyword search, which can be carried out without specifying a target element. BNs cannot be found by keyword search. That is why, together with the BN, the pipeline receives the IRI of the element for which the search was carried out and the type of the link between the BN and the target element. Based on these data, the initial SPARQL query is created. Then, this query is executed, and its result is checked for the presence of BNs. If no new BNs are found, then the context is considered complete; otherwise, the result is used to extend the initial query. In the latter case, the extended SPARQL query is executed and the first step is repeated. The cycle continues until a result without BNs is obtained or until the context includes the entire graph. The result of the last query represents the desired context.



(a) owl:Restriction



(b) The nonstandard case, the RDF list construct RDF list and owl:IntersectionOf

Fig. 6. Representation of constructs in EasyRdf.

3.5. Context-Dependent Identifiers

Upon retrieving the context, context-dependent identifiers are generated. The idea is to create an identifier from which the context can later be restored. For this purpose, Ontodia converts the context into a hash and uses it as an identifier. In this case, the hash function implements the following steps.

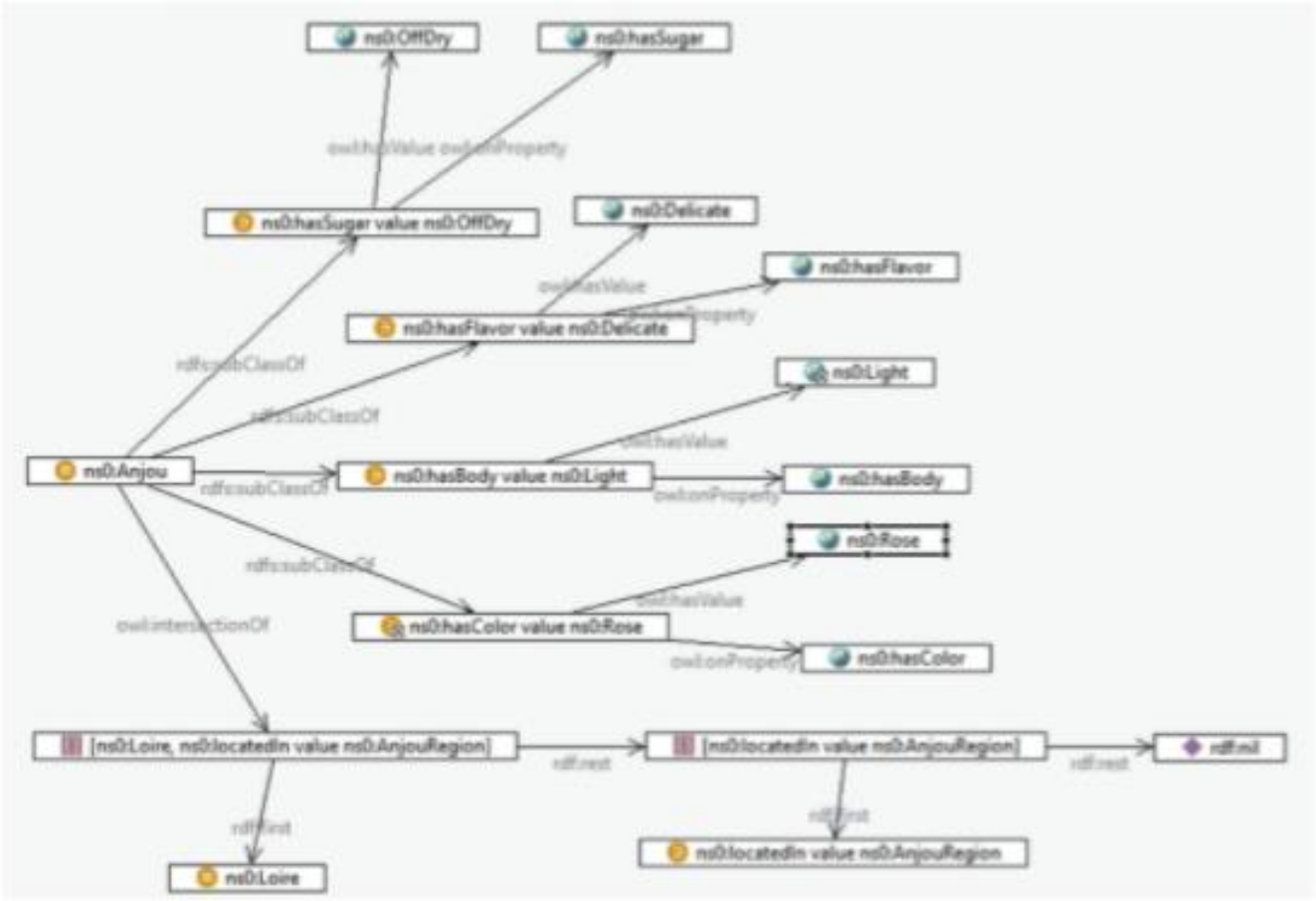
1. The context graph is converted to the canonical form (see Listing 2). A method for converting an RDF graph to the canonical form was described in the paper "Canonical forms for isomorphic and equivalent RDF graphs: Algorithms for leaning and labeling blank nodes" [11].

2. The IRI elements are encoded by the JavaScript function encodeURIComponent in such a way that the hash can be used as part of the IRI.

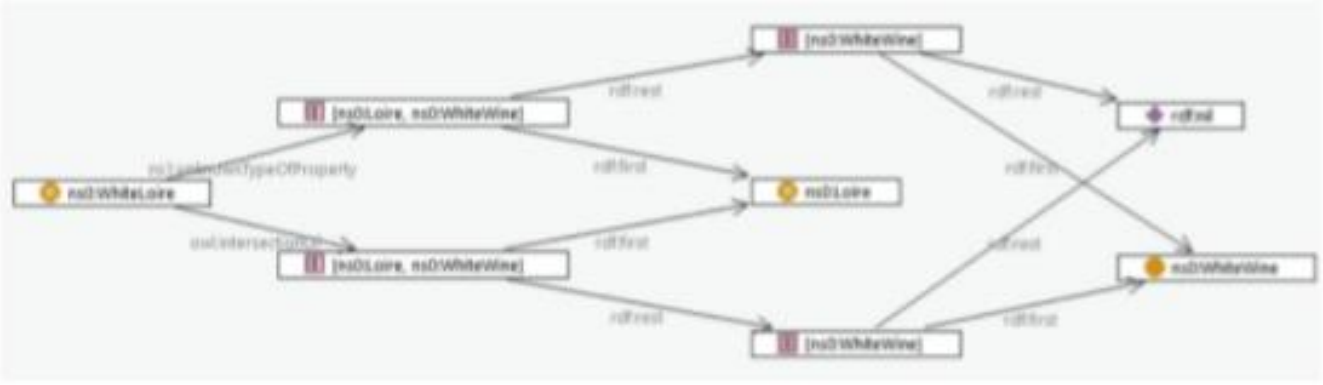
3. A vocabulary of terms is extracted from the graph and a special array is formed to describe the types of its terms (N is NamedNode, B is BlankNode, L is Literal with language, D is Literal with datatype, V is Variable, and G is DefaultGraph).

Table 1. Ontology visualization tools

Tool	Individual visualization	Global visualization	Incrementally	Loading
OntoGraph Protege	Yes	No	Yes	Full
Graffoo	No	No	Yes	Full
WebVOWL	Yes	No	No	Full
OWLGrEd	Yes	No	No	Full
EasyRdf	No	Yes	No	Full
TopBraid Composer	Yes	Yes	Yes	Full
Ontodia	Yes	Yes	Yes	Lazy



(a) `owl:IntersectionOf` and `owl:Restriction`



(b) The nonstandard case, the RDF list construct RDF list and `owl:IntersectionOf`

Fig. 7. Representation of constructs in TopBraid Composer.

4. The vocabulary and graph represented as a set of quads are compressed and encoded using a prefix tree [12] (see Listing 3).

5. The resulting elements are combined into an array.

6. In the resulting line, the following substitutions are performed: [→ (,] →), , → : , " → '.

7. To obtain the final IRI (see Listing 4), an individual index and prefix are added to the resulting hash. Nodes of different types use different prefixes:

(a) "`ontodia:blank:`" for an individual BN;

(b) "`ontodia:list:List`" for an RDF list.

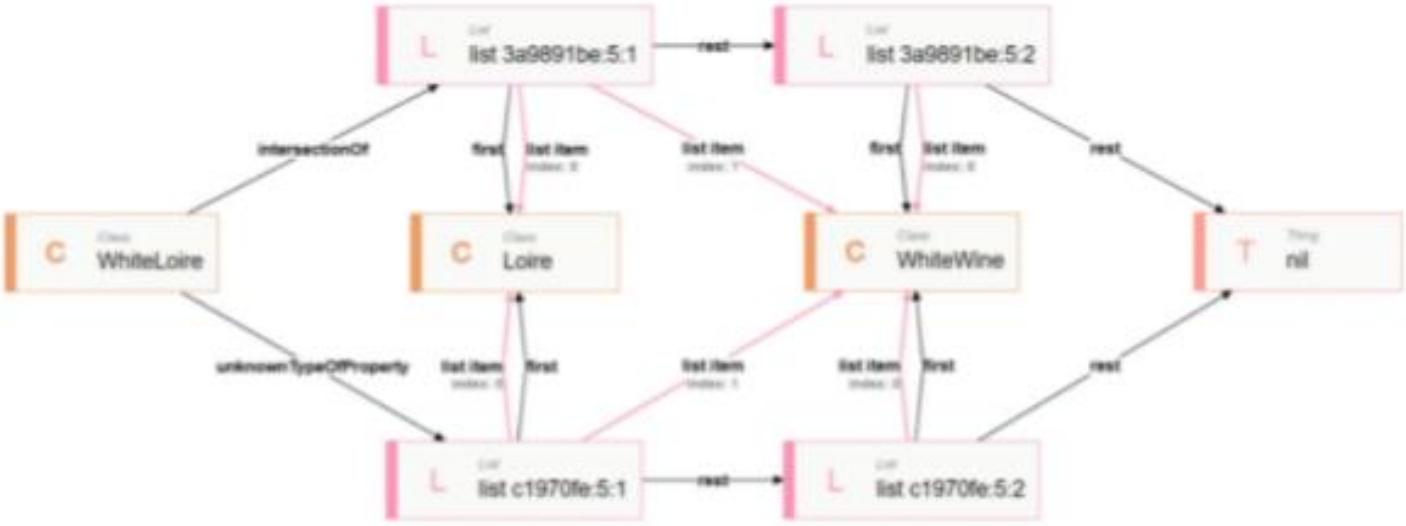
The last step is required to uniquely determine, based on the IRI of an element, whether the IRI is an encoded context or not, as well as to provide subsequent individual visualization of the lists.

To restore the context from the encoded IRI, it is required to perform these steps in reverse order.

3.6. Output of the Preprocessing Results

Once the identifiers are formed, the modified context is put in the local storage, which, like the basic

(a) `owl:IntersectionOf` and `owl:Restriction`



(b) The nonstandard case, the RDF list construct RDF list and `owl:IntersectionOf`

Fig. 8. Representation of constructs in Ontodia.

one, implements the `DataProvider` interface. First, the output of the results obviously occurs in the `filter` function immediately after preprocessing; however, it is not the only place. For instance, a call of the `filter` function returns a set of BNs that we add to the graph, while the information about the links included in the context remains unused. At the next step of the graph rendering process, when the query for links between the elements is executed, this information becomes useful. In this case, the information about the links is retrieved from the local storage rather than the basic one. Access to the local storage is

Table 2. Possible data queries to DataProvider

`classTree`	Query for a tree of classes/types
`linkTypes`	Query for possible types of links in a graph
`classInfo`	Query for additional information about a class (if available)
`linkTypesInfo`	Query for information about a link
`elementInfo`	Query for information about an element
`linksInfo`	Query for links between elements
`linkTypesOf`	Query for a list of types of incoming/outgoing links
`filter`	Query for a list of elements with search parameter specification

more efficient and the data in it are already canonicalized. Below is the complete list of queries that use the preprocessed data:

- `elementInfo`
- `linksInfo`
- `linkTypesOf`
- `filter`

When loading a saved diagram, the reverse procedure is carried out. Once we encounter an element with a special prefix "`ontodia:blank:`" or "`ontodia:list:List`," we restore the context from the IRI, put the result in the local storage, and then return the result as described above.

4. CONCLUSIONS

The proposed method solves the problems of visualization and restoration of saved graphs. When saving a graph, the identifiers of its nodes are also saved; with its problem-specific part (context with BNs) being encoded in identifiers, it can also be easily restored when loading the saved graph. The method has been

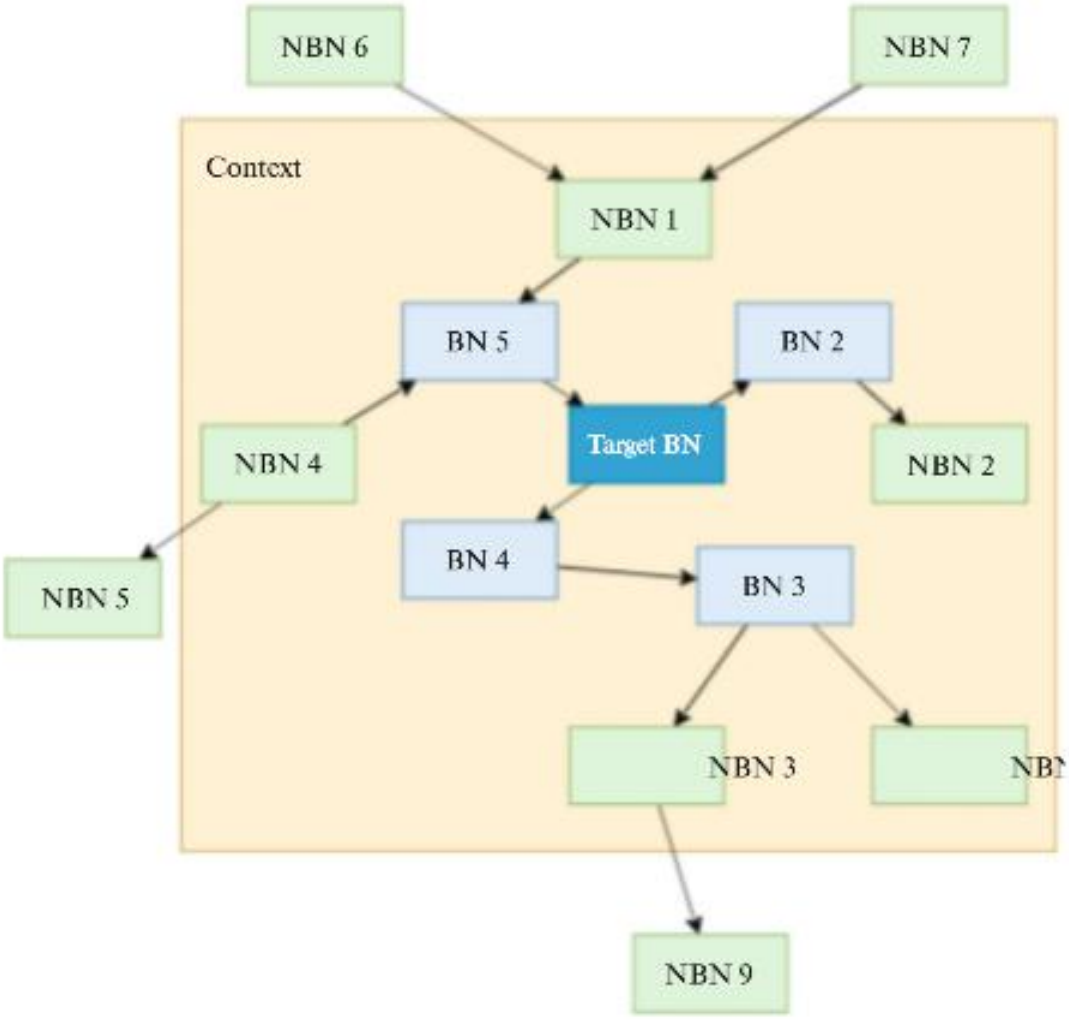


Fig. 9. Representation of the BN context.

```
{
    "quads": [{
            "subject": {"value": "b3"},
            "predicate": {"value": "http://www.w3.
              org/1999/02/22-rdf-syntax-ns#first"},
            "object": {"value": "http://www.w3.org/
              TR/2003/PR-owl-guide-20031209/wine#
              Loire"},
            "graph": {"value": ""}
        },
        ...
        {
            "subject": {"value": "b4"},
            "predicate": {"value": "http://www.w3.
              org/1999/02/22-rdf-syntax-ns#type"},
            "object": {"value": "http://www.w3.org/2
              002/07/owl#Class"},
            "graph": {"value": ""}
        }, {
            "subject": {"value": "b4"},
            "predicate": {"value": "http://www.w3.
              org/2002/07/owl#intersectionOf"},
            "object": {"value": "b3"},
            "graph": {"value": ""}
        },
        ...
    ], "pointer": {  "value": "b4" }
}
```

Listing 2. Part of the canonicalized context graph with a pointer to the target element.

described in general and is suitable for lazy BN visualization not only in Ontodia but also in other ontology visualization tools. The next step in the further development of the proposed method is to formalize the processing of special structures that use BNs as structural components:

- list (RDF list);
- axiom (`owl:Axiom`);
- restriction (`owl:Restriction`);
- mutual distinction (`owl:AllDifferent`);
- union (`owl:unionOf`);
- intersection (`owl:intersectionOf`);
- complement (`owl:complementOf`);
- enumeration (`owl:oneOf`) [13].

For instance, an RDF list can be preprocessed and visualized on a diagram as a table node while preserving the original sequence order of its elements.

In addition, data visualization often borders on visual data editing. In this direction, there are also a number of challenges. For instance, the proposed algorithm regards an ontology as something complete and unmodifiable. If an ontological graph is modified in its BN context area, then all node identifiers lose their meaning and the context needs to be recollected. In this case, it is not always easy to say whether the ontology was modified in the BN context area or not; to determine this, it is required to compare the old context graph with the new one, which is often a nontrivial task. In this regard, it seems interesting to adapt

```
[[
    [0,"http%3A%2F%2Fwww.w3.org%2F",[0,"1999%2F02%2F
      22-rdf-syntax-ns%23",[0,"first",1,"nil",1,"
      rest",1,"type",1],"2002%2F07%2Fowl%23",[0,"
      Class",1,"Restriction",1,"equivalentClass",1,
      "hasValue",1,"intersectionOf",1,"onProperty",
      1],"TR%2F2003%2FPR-owl-guide-20031209%2Fwine%
      23",[0,"Anjou",[1,"Region",1],"Loire",1,"
      locatedIn",1]]],
    ["N",0,"N",1,"N",2,"N",3,"N",4,"N",5,"N",6,"N",7
      ,"N",8,"N",9,"N",10,"N",11,"N",12,"N",13]
],
[-1,0,-2,-1,2,1,-3,0,12,-3,2,-1,-4,3,4,-4,8,-3,-2,3,
  5,-2,7,11,-2,9,13,10,6,-4]]
```

Listing 3. Compact representation of the context graph in the JSON format, restored from the IRI element.

```
ontodia:blank:sparql2:4:(((0:'http%3A%2F%2Fwww.w3.
  org%2F':(0:'1999%2F02%2F22-rdf-syntax-ns%23':(0:'
  first':1:'nil':1:'rest':1:'type':1):'2002%2F07%2
  Fowl%23':(0:'Class':1:'Restriction':1:'
  equivalentClass':1:'hasValue':1:'intersectionOf':
  1:'onProperty':1):'TR%2F2003%2FPR-owl-guide-20031
  209%2Fwine%23':(0:'Anjou':(1:'Region':1):'Loire':
  1:'locatedIn':1))):('N':0:'N':1:'N':2:'N':3:'N':4
  :'N':5:'N':6:'N':7:'N':8:'N':9:'N':10:'N':11:'N':
  12:'N':13)):(-1:0:-2:-1:2:1:-3:0:12:-3:2:-1:-4:3:
  4:-4:8:-3:-2:3:5:-2:7:11:-2:9:13:10:6:-4))
```

Listing 4. Example of a generated IRI.

the proposed method to the problem of editing ontological graphs with BNs.

REFERENCES

1. Mouromtsev, D., Pavlov, D., Emelyanov, Y., Morozov, A., Razdyakonov, D., and Galkin, M., The simple web-based tool for visualization and sharing of semantic data and ontologies, *Proc. Int. Semantic Web Conf.*, 2015.
2. Bessmertny, I.A., Knowledge visualization based on semantic networks, *Program. Comput. Software*, 2010, vol. 36, no. 4, pp. 197–204.
3. Katerinenko, R.S. and Bessmertnyi, I.A., A method for acceleration of logical inference in the production knowledge model, *Program. Comput. Software*, 2011, vol. 37, no. 4, pp. 197-199.
4. Gennari, J.H., Musen, M.A., Fergerson, R.W., Grosso, W.E., Crubzy, M., Eriksson, H., Noy, N.F., and Tu, S.W., The evolution of Protégé: An environment for knowledge-based systems development, *Int. J. Hum.–Comput. Stud.*, 2003, vol. 58, no. 1, pp. 89–123.
5. Falco, R., Gangemi, A., Peroni, S., Shotton, D., and Vitali, F., Modelling OWL ontologies with Graffoo, *The Semantic Web: ESWC 2014 Satellite Events*, Presutti, V., Blomqvist, E., Troncy, R., Sack, H., Papadakis, I., and Tordai, A., Eds., Springer, 2014, pp. 320–325.
6. Lohmann, S., Link, V., Marbach, E., and Negru, S., WebVOWL: Web-based visualization of ontologies, pp. 154–158.

7. Cerans, K., Ovinnikova, J., Liepin, R., and Grasmanis, M., Extensible visualizations of ontologies in OWLGrEd, *The Semantic Web: ESWC 2019 Satellite Events,* Hitzler, P., Kirrane, S., Hartig, O., de Boer, V., Vidal, M.-E., Maleshkova, M., Schlobach, S., Hammar, K., Lasierra, N., Stadtmuller, S., Hose, K., and Verborgh, R., Eds., Springer, pp. 191–196.

8. Humfrey, N., EasyRdf: A PHP library designed to make it easy to consume and produce RDF. http://www.easyrdf.org.

9. TopBraid Composer FAQ. https://www.topquadrant.com/knowledge-assets/faq/tbc.

10. Nock, C., *Data Access Patterns: Database Interactions in Object-Oriented Applications,* Boston: Addison-Wesley, 2004.

11. Hogan, A., Canonical forms for isomorphic and equivalent RDF graphs: Algorithms for leaning and labelling blank nodes, *ACM Trans. Web,* 2017.

12. Gudkova, T.S., Prefix compression of indexes, *Inf.: Probl., Metodol., Tekhnol.,* 2019, pp. 1310–1314.

13. Zaikin, I.A., Algorithm for comparison of ontology variants, *Elektron. Sredstva Sist. Upr.,* 2010, no. 1, pp. 132–135.

Translated by Yu. Kornienko