ПРОГРАММИРОВАНИЕ, 2020, № 2, с. 3–??

КОМПЬЮТЕРНАЯ ГРАФИКА И ВИЗУАЛИЗАЦИЯ

УДК 004.424.64

Подход к обработке пустых узлов при порционной визуализации данных на примере инструмента Ontodia

© 2020 г. Д.С. Раздьяконов*, А.В. Морозов*, Д.С. Павлов, Д.И. Муромцев***

**Факультет Программной Инженерии и Компьютерных Технологий, Университет ИТМО*

197101 Санкт-Петербург, Кронверкский пр., д. 49

***Metaphacts East Europe*

197110 Санкт-Петербург, Большая Разночинная ул., д. 14

Поступила в редакцию 03.07.2020

Задача ленивой визуализации онтологических графов имеет ограничения. Особые затруднения вызывает необходимость визуализации структур содержащих пустые узлы. В данной статье мы рассмотрим подход к визуализации подобных структур, реализованный в инструменте Ontodia, рассмотрим ограничения данного подхода, а также альтернативные пути решения.

1. ВВЕДЕНИЕ

Среди исследователей, работающих в области семантических технологий, всем известно, что такое пустые узлы (ПУ), иначе называемые Blank nodes. Они используются для представления объектов, не имеющих постоянных идентификаторов, то есть IRI (Internationalized Resource Identifier — интернационализированный идентификатор ресурса). Примером использования может быть, например, представление класса, являющегося объединением нескольких классов, или представление коллекции. В случае представления объединений, например, конструкции `owl:unionOf`, получившийся класс будет пустым узлом, который ссылается на RDF list — структуру, в которой все промежуточные элементы, включая корневой, являются пустыми узлами, а перечисляемые элементы — обычными узлами с читаемыми идентификаторами (Рис. 1). То же самое касается представления конструкции `owl:Restriction`(Рис. 2). Подразумевается, что подобный узел не имеет значения без указания, на какое свойство вводится ограничение, а также без ограничения на значение.

Отсутствие идентификаторов у пустых узлов вполне оправдано, однако с технической точки зрения это является помехой, т.к. отсутствие идентификаторов означает невозможность ссылаться на такие узлы. Получается, что ПУ нельзя воспринимать отдельно от структуры, частью которой они являются, тем не менее в некоторых случаях именно это является сугубо необходимым.

Подобная необходимость, например, возникает, когда перед нами встает задача визуализации графа вкупе с возможностью сохранять и восстанавливать из сохраненного файла или иного артефакта ранее созданную визуализацию. Задача создания инструмента визуализации графов, содержащих ПУ, не является очень трудоемкой до тех пор, пока всю структуру мы получаем одной порцией, но в случае с инструментом Ontodia [8], мы сталкиваемся с ленивой визуализацией, где данные получаются порциями, встает также вопрос о том какие данные уже визуализированы, а какие нет, и здесь отсутствие идентификаторов у пустых узлов играет ключевую роль. При подобной визуализации мы не можем однозначно сказать является ли узел, пришедший вместе с новой порцией данных, тем же самым узлом, который уже отображен на диаграмме, или это иной узел, имеющий тот же набор связей и тот же тип. Подобная проблема возникает, когда мы пытаемся восстановить сохраненную ранее визу-

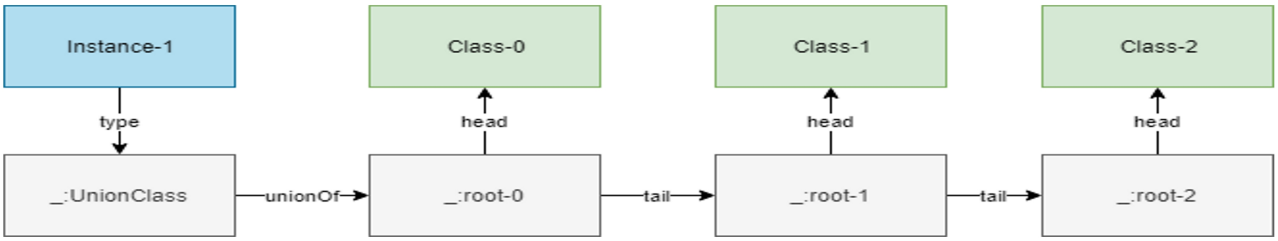


Рис. 1.: Представление конструкции RDF list

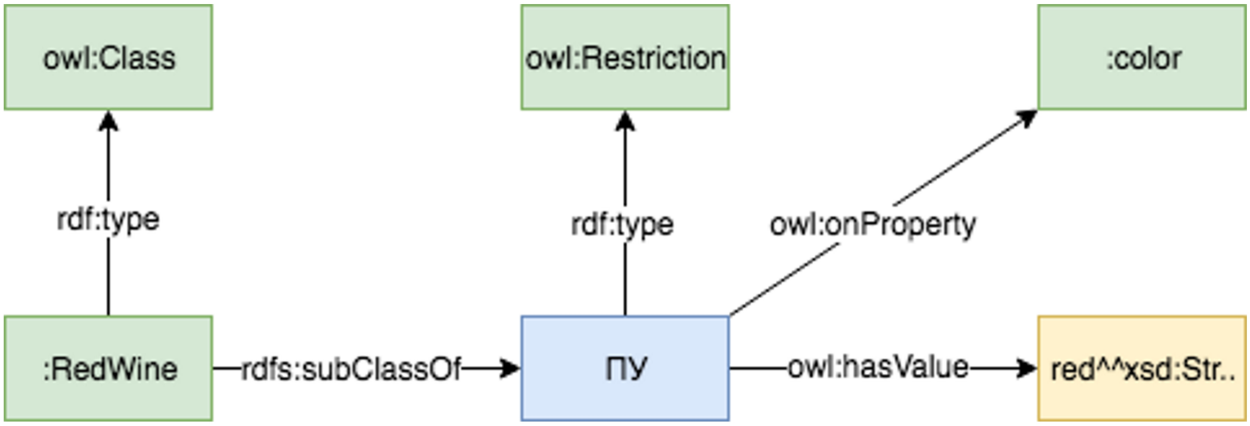


Рис. 2.: Представление конструкции `owl:Restriction`

ализацию, и в этом случае уже не важно, используется ленивая загрузка, или визуализация была построена из одной порции данных.

В данной статье мы рассмотрим, как в инструменте Ontodia решаются вопросы построения визуализации онтологического графа, содержащего ПУ, в условиях ленивой загрузки данных, а также вопрос сохранения и восстановления подобной визуализации (диаграммы).

Визуализация в целом и предложенный подход в частности могут быть полезны в ряде задач по протоке логических выражений, рассмотренных в таких статьях как "Визуализация знаний на основе семантической сети"[10] и "Методы ускорения логического вывода в продукционной модели знаний"[13].

2. СУЩЕСТВУЮЩИЕ ПОДХОДЫ

Здесь и далее для примеров используются данные, представленные на листинге 1 на странице 5. Данные являются частью онтологии вин `http://www.w3.org/TR/owl-guide/wine.rdf`. Для наглядности одна из связей продублирована, и её идентификатор заменен на нестандартный: `owl:intersectionOf` → `<http://example.com/unknownTypeOfProperty>`. Это сделано, чтобы показать, как средства визуализации обрабатывают нестандартные случаи.

Есть несколько подходов к обработке ПУ. Некоторые подходы включают непосредственно решение задачи ленивой визуализации онтологического графа, другие также решают задачу сравнения графов, содержащих ПУ, но преследуют другие цели. Так Ontop Protégé[4] для работы с ПУ вводит ограничение на свойства ПУ — каждый ПУ должен иметь литеральное свойство `rdfs:label`, которое, очевидно, выступает в качестве постоянного идентификатора ПУ. Другой плагин OntoGraph Protégé позволяет визуализировать ПУ, однако нестандартным образом. В OntoGraph ПУ визуализируются в виде аннотаций, как это представлено на рисунке 3а. На рисунке 3а видно, что аннотации представляются в виде всплывающих окон, при этом конструкция `owl:Restriction` представляется как свойство `Superclasses`, а `owl:IntersectionOf` отображается в виде

```
@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 .
```

Листинг 1: Представление конструкций `owl:IntersectionOf` и `owl:Restriction` в формате Turtle

`Equivalent classes`. Иначе OntoGraph Protégé разбирается с отображением нестандартных структур. На рисунке 3b видно, что нестандартная связь к структуре RDF list выделяется в отдельный блок `Annotations`, после чего отображается только первый ПУ структуры.

В некоторых инструментах, таких как Graffoo[3], отсутствует поддержка визуализации ПУ, и при рендеринге ПУ исключаются из визуализации. WebVOWL[7] при визуализации онтологий, содержащих ПУ, осуществляет индивидуальный подход к визуализации известных структур, однако список поддерживаемых структур ограничен. Так, например, конструкция `owl:intersectionOf` представляется в виде топологии звезда, где центральный узел является пересечением других классов топологии (см. Рис. 4), при этом WebVOWL практически игнорирует конструкции `owl:Restriction`, отображая только связи присоединенные к базовой сущность `owl:Thing`. Нестандартные случаи игнорируются. Стоит также отметить, что WebVOWL сосредотачивается в большей степени на визуализации классов, нежели экземпляров классов.

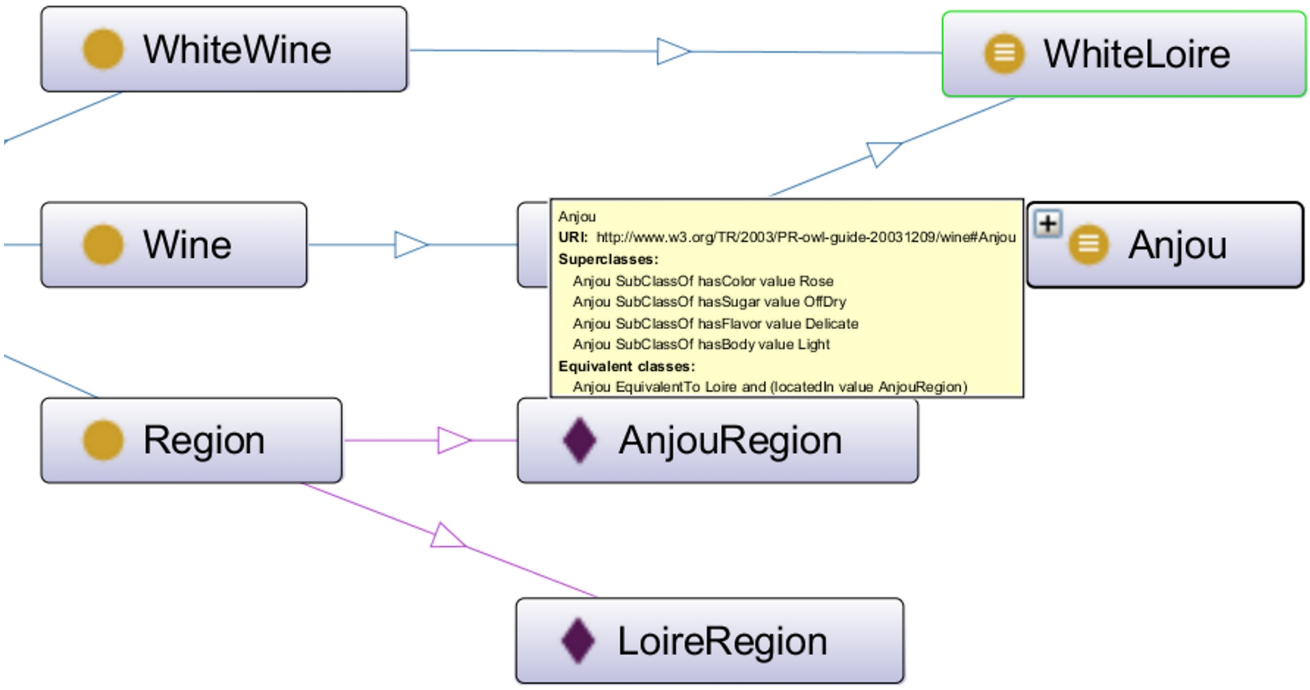
Подобным путем индивидуального подхода идёт инструмент OWLGrEd[2]. Конструкция `owl:intersectionOf` визуализируется в виде дерева. Конструкция `owl:Restriction` отображается в теле целевого элемента в виде аннотации (Рис. 5a). В отличие от WebVOWL, OWLGrEd способен рисовать экземпляры классов, выделяя их зеленым цветом, однако не справляется с отображением нестандартных случаев. Конструкции типа RDF list игнорируются, если не являются частью конструкции `owl:IntersectionOf` (Рис. 5b).

С другой стороны к данному вопросу подходит php-библиотека EasyRdf[6].

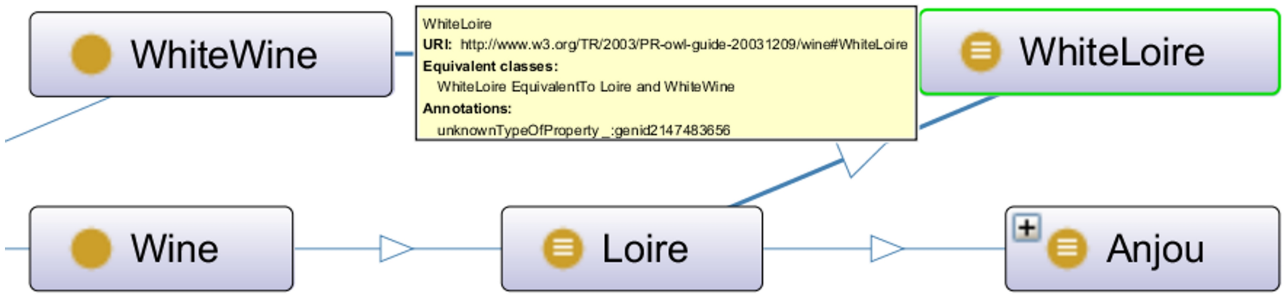
При визуализации EasyRdf максимально точно передает структуру онтологического графа, но не предпринимает попыток индивидуальной визуализации общеизвестных структур (см. Рис 6a). На рисунке 6a представлена только часть результата. Это сделано вследствие того, что EasyRdf не предоставляет возможность изменять результат визуализации, и полный результат будет сложен для восприятия. Весь граф единомоментно помещается в память браузера, что дает возможность ссылаться на ПУ и производить визуализацию целевых структур. Плюсом данного подхода является то, что нестандартные случаи отображаются в точность так, как они определены (см. Рис 6b). Рассматривая примеры OWLGrEd, WebVOWL, следует сказать, что индивидуальный подход имеет свои преимущества при визуализации, если же его совмещать с глобальным подходом, как это делает EasyRdf, то мы получим комфортную визуализацию знакомых структур, совмещенную с возможностью универсальной визуализации.

Данный подход, например, осуществляет инструмент работы с онтологиями TopBraid Composer[1] (см. Рис 7a и 7b). Как можно видеть из результатов визуализации, TopBraid индивидуально подходит к отображению конструкций `owl:IntersectionOf` и `owl:Restriction` (см. Рис 7a), отображая ограничения и пересечения в теле элементов. Подобным образом TopBraid подходит к отображению конструкций RDF list, где элементы списка по порядку отображаются в теле головы списка, но в то же время инструмент способен показать всю структуры целиком, отображая пустые узлы в виде набора RDF list соединенных связями `rdf:rest` (см. Рис 7b). При этом весь объем данных загружается в память приложения и индексируется в соответствие с задачей, что позволяет порционно визуализировать граф. Вся модель хранится в памяти компьютера пользователя, поэтому визуализация ограничена ресурсами компьютера. Большие базы знаний, такие как Wikidata и DBpedia, не могут быть отображены подобным образом.

Что же касается инструмента Ontodia, здесь, как и в TopBraid, совершена попытка совместить два подхода, при этом индивидуальный подход в текущей реализации очень беден и включает только обработку конструкций типа RDF list, остальные конструкции визуализируются в рамках глобального подхода (см. Рис. 8a). Как можно видеть из рисунка, для всех элементов RDF list проводятся дополнительные связи, указывающие на голову списка, а также указывается порядковый номер элемента в списке. То же самое делается и для хвоста списка, который, в свою очередь, является самостоятельным списком. Такое подход помогает легко справляться с



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



(b) Нестандартный случай, конструкция RDF list

Рис. 3.: Представление конструкций в инструменте OntoGraph Protégé

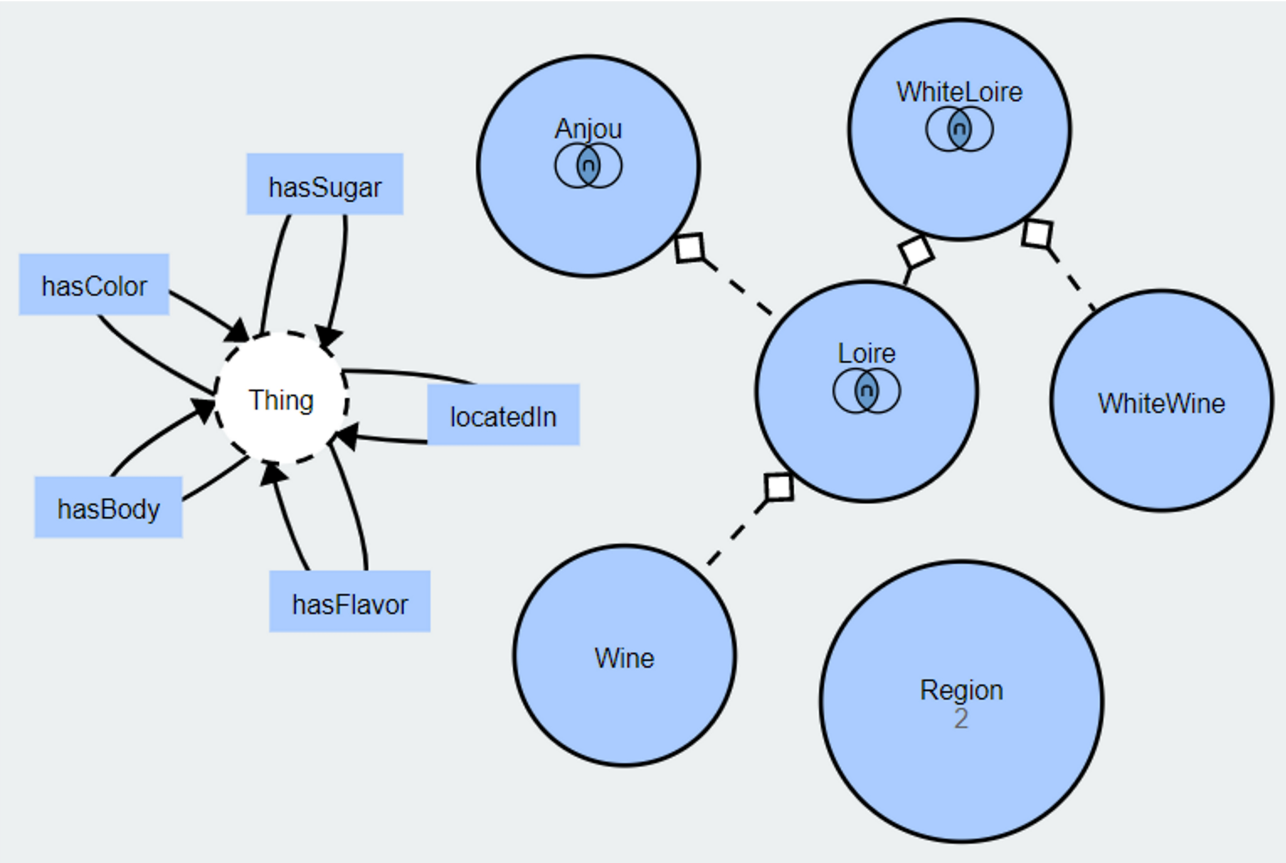
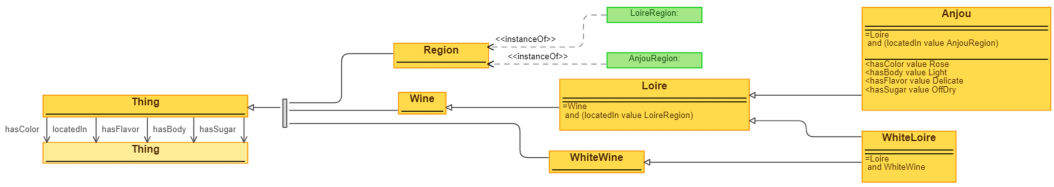
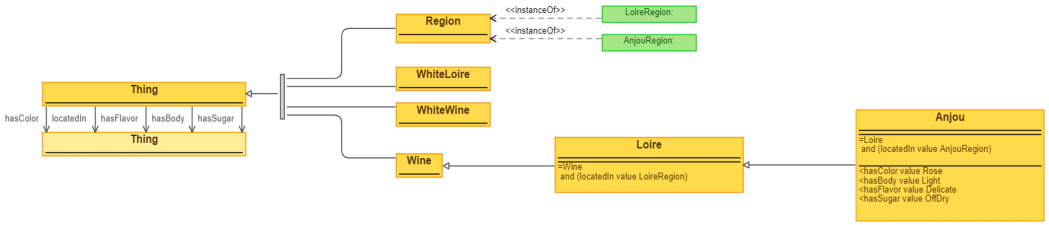


Рис. 4.: Конструкции `owl:Restriction` и `owl:IntersectionOf` в инструменте WebVOWL



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

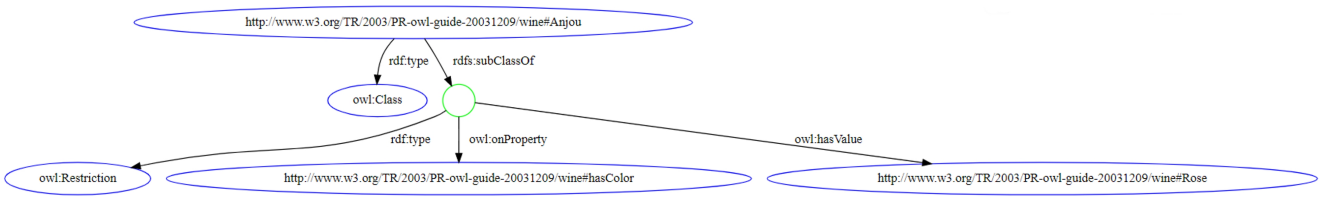


(b) Нестандартный случай, конструкция RDF list

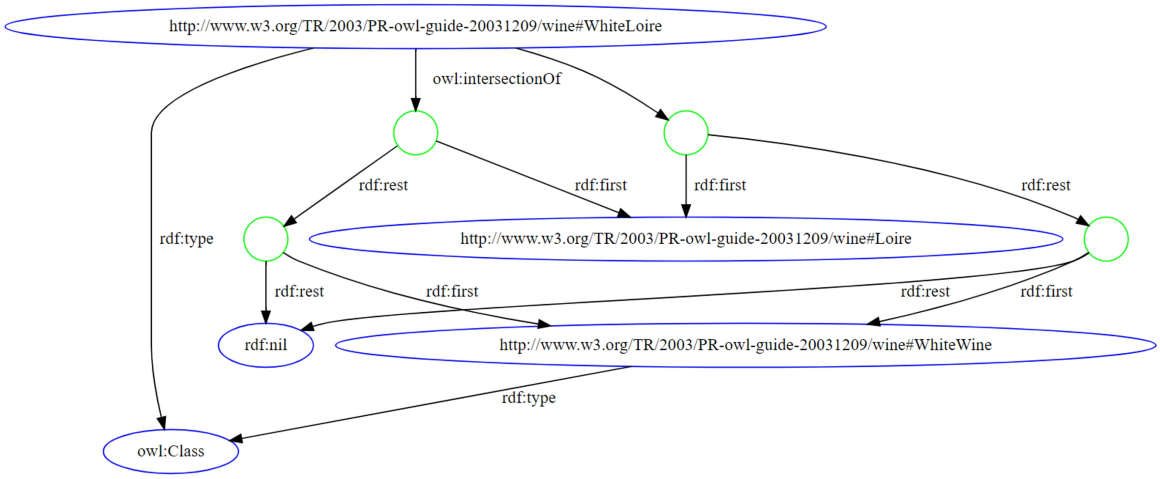
Рис. 5.: Представление конструкций в инструменте OWLGrEd

визуализацией нестандартных случаев (см. Рис. 8b). Стоит отметить, также, что загрузка данных в инструменте осуществляется лениво, загружается только визуализированная часть онтологического графа, что позволяет работать с большими данными.

Исходя из вышеперечисленного, получаем следующую сводную таблицу 1, которая описывает подходы рассмотренных инструментов к визуализации структур, содержащих ПУ.



(a) `owl:Restriction`

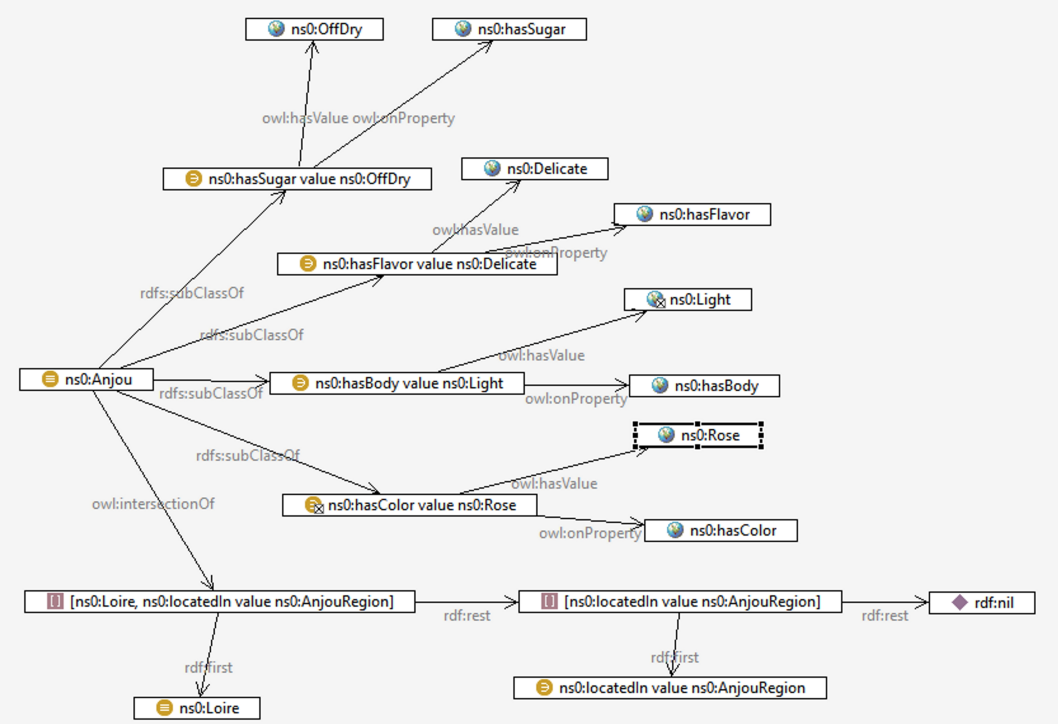


(b) Нестандартный случай, конструкция RDF list и `owl:IntersectionOf`

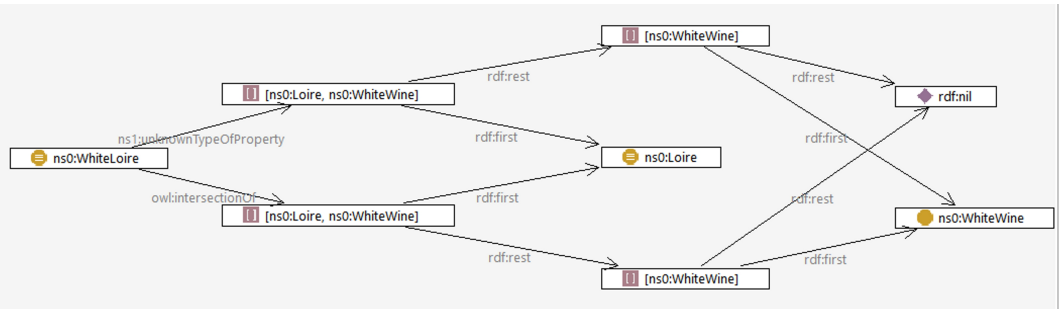
Рис. 6.: Представление конструкций в инструменте EasyRdf

Инструмент	Инд. виз.	Глоб. виз.	Порционно	Загрузка
OntoGraph Protege	да	нет	да	Целиком
Graffoo	нет	нет	да	Целиком
WebVOWL	да	нет	нет	Целиком
OWLGrEd	да	нет	нет	Целиком
EasyRdf	нет	да	нет	Целиком
TopBraid Composer	да	да	да	Целиком
Ontodia	да	да	да	Лениво

Таблица 1.: Инструменты визуализации онтологических графов

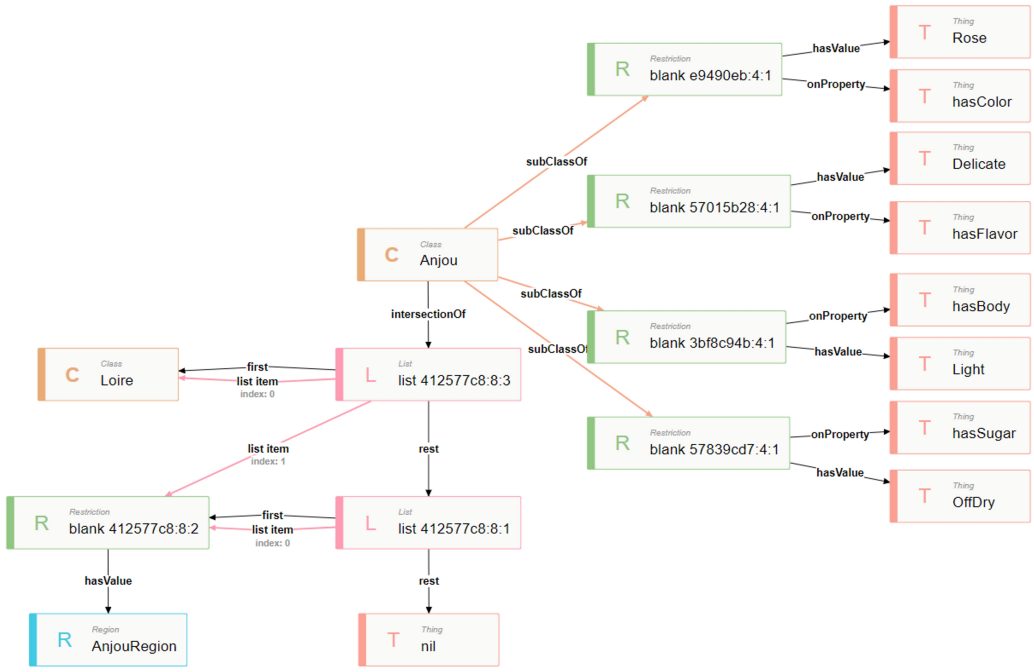


(a) `owl:IntersectionOf` и `owl:Restriction`

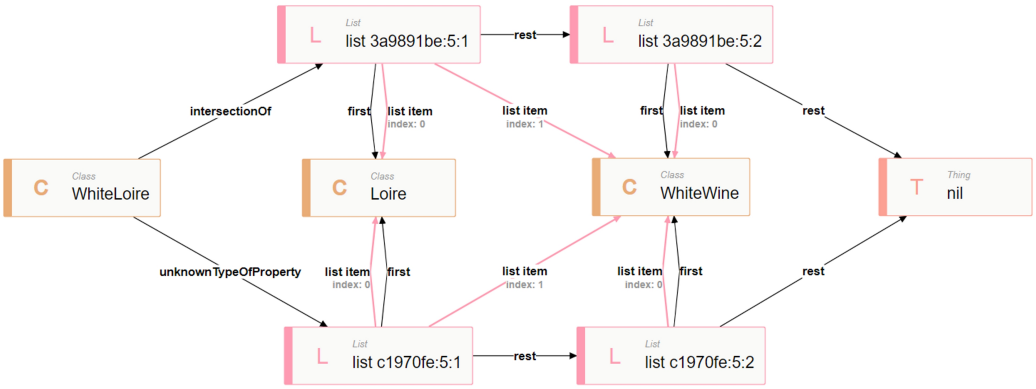


(b) Нестандартный случай, конструкция RDF list и `owl:IntersectionOf`

Рис. 7.: Представление конструкций в инструменте TopBraid Composer



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



(b) Нестандартный случай, конструкция RDF list и `owl:IntersectionOf`

Рис. 8.: Представление конструкций в инструменте Ontodia

3. ОПИСАНИЕ РЕШЕНИЯ

3.1. ЦИРКУЛЯЦИЯ ДАННЫХ В ONTODIA

Ленивая загрузка данных графа в инструменте Ontodia осуществляется средствами объекта `DataProvider`. `DataProvider` воплощает идею паттерна проектирования Data Access Object [9]. `DataProvider` отвечает за выбор схемы и данных из базового хранилища, а также за перевод загруженных данных во внутреннюю модель Ontodia. Кроме того, на этапе преобразования данных есть возможность изменять структуру данных в соответствии с потребностями конкретного приложения. Это включает, например, группирование отдельных узлов в таблицы в пользовательском интерфейсе, группирование узлов в суперузлы или свертывание путей между ними. В том числе `DataProvider` позволяет лениво подгружать данные в ответ на запросы.

Возможные запросы данных к `DataProvider` описаны в таблице 2. Большинство запросов используются в специфичных только для Ontodia случаях и существуют для оптимизации циркуляции данных в инструменте для эффективного отображения элементов интерфейса.

Здесь и далее под дополнительной информацией мы будем понимать ту информацию, которая не влияет на топологическую структуру онтологического графа. Иными словами: названия элементов, типы элементов и связей, литеральные свойства элементов.

3.2. ОБЩЕЕ ОПИСАНИЕ РЕШЕНИЯ

Решение, которое предлагает Ontodia для отображения ПУ, состоит в том, чтобы ввести стадию предобработки запросов и выдать контекстно зависимые идентификаторы пустым узлам. Стадия предобработки включает в себя несколько шагов: сбор контекста, формирование контекстно зависимых идентификаторов, сохранение результатов предобработки для последующей выдачи. Идентификаторы должны быть составлены так, чтобы было возможно восстановить контекст напрямую из каждого сгенерированного идентификатора и однозначно сравнить узлы.

Контекст определяется следующим образом: *Контекст для целевого пустого узла — это подграф основного графа, который включает целевой ПУ, а также транзитивно все ПУ, заключенные между непустыми узлами, окружающими данный подграф и включающий их (см. Рис. 9).* То есть контекст для целевого ПУ — это граф, который содержит целевой ПУ и всех его соседей и соседей его соседей, вплоть до первого непустого узла (НПУ).

Далее, чтобы подробней описать метод, ответим на следующие вопросы:

1. В какой момент и где производится предобработка данных?
2. Как собирается контекст?
3. Как создаются контекстно зависимый идентификатор?
4. Где и когда происходит выдача результатов?

3.3. ГДЕ ПРОИЗВОДИТСЯ ПРЕДОБРАБОТКА ИНФОРМАЦИИ

Весь обмен данными между `DataProvider` и диаграммой сводится к запросам диаграммой (у `DataProvider`) связей и элементов, где элементы — это классы и их экземпляры, а также дополнительной информации об элементах и связях. Под запросами на дополнительную информацию понимаются следующие запросы:

- запрос всех возможных типов связей в графе без привязки к конкретным узлам;
- запрос свойств классов и элементов, иначе называемых `DataProperty`;
- запрос свойств связей.

Для выполнения поставленной нами задачи сосредоточимся на тех запросах, которые оперируют узлами, потому что дополнительная информация никак не влияет на структуру онтологического графа, а связи, в случае ПУ, будут закодированы вместе с контекстом в IRI. Из списка запросов, которые используются Ontodia (см. раздел "Циркуляция данных в Ontodia") для получения данных, мы можем выделить те, которые оперируют исключительно вершинами графа:

`classTree`	Запрос дерева классов/типов.
`linkTypes`	Запрос возможных типов связей в графе;
`classInfo`	Запрос дополнительной информации о классе, если имеется.
`linkTypesInfo`	Запрос данных о связи
`elementInfo`	Запрос данных об элементах
`linksInfo`	Запрос связей между элементами
`linkTypesOf`	Запрос списка типов входящих/исходящих связей элемента
`filter`	Запрос списка элементов с указанием условий поиска

Таблица 2.: Возможные запросы данных к `DataProvider`

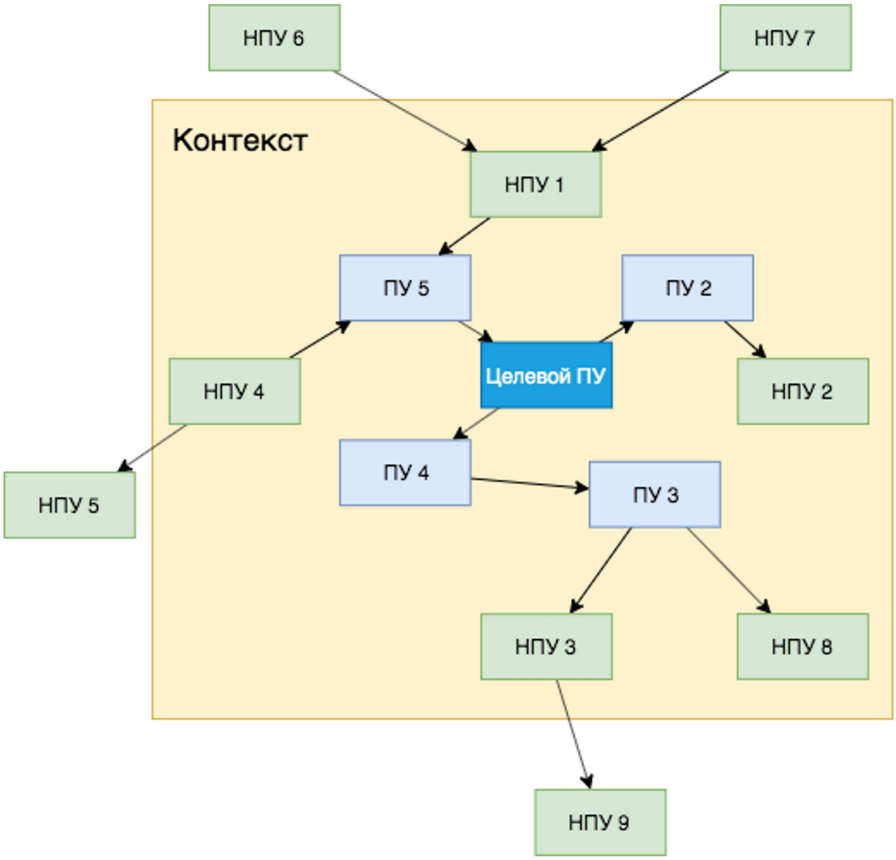


Рис. 9.: Представление контекста ПУ

- classTree — Запрос дерева классов/типов;
- filter — Запрос списка узлов с указанием условий поиска.

Остальные запросы или возвращают информацию о связях, или оперируют дополнительной информацией. В ходе разработки также было принято решение о том, что дерево классов не должно включать ПУ, так как ПУ обычно не имеют читаемых идентификаторов и не имеют смысла вне структуры, частью которой являются. Соответственно, единственной точкой, в которой мы можем производить предобработку данных, является filter-запрос. Этот запрос используется в Ontodia везде, где нужно получить список элементов — меняются лишь параметры фильтрации. Предобработка информации происходит сразу после получения результатов запроса. Результаты представляют собой массив троек [subject, predicate, object]. Прежде чем вернуть результаты, в запросе выделяются тройки, содержащие в качестве объекта или субъекта ПУ, они обрабатываются отдельно — это детально разобрано в следующем разделе 3.4 "ПОЛУЧЕНИЕ КОНТЕКСТА", стр. 14.

3.4. ПОЛУЧЕНИЕ КОНТЕКСТА

Сбор контекста осуществляется путем рекурсивного формирования SPARQL-запроса всего контекста и последующего его выполнения. Допустим, используя функцию `filter DataProvider`, мы получили ПУ. Результат поступает на конвейер предобработки. Здесь нужно пояснить, что функция `filter` в большинстве случаев осуществляет поиск с привязкой к целевому элементу, то есть поиск соседей целевого элемента или фильтрацию списка возможных связей для целоевого элемента. Исключением является только поиск по ключевому слову, который может осуществляться без указания целевого элемента. Пустой узел не может быть найден с использованием поиска по ключевому слову, поэтому, когда ПУ поступает на конвейер, вместе с ним поступает IRI элемента, для которого выполнялся поиск, и тип связи между ПУ и целевым элементом. На основе этих данных создается первичный SPARQL-запрос. Далее запрос выполняется, и его результаты проверяются на наличие ПУ. Если новые ПУ не обнаружены, то контекст считается завершенным, в противном случае результат используется для расширения первичного запроса, после чего осуществляется выполнение расширенного SPARQL-запроса и первый шаг повторяется. Цикл происходит до тех пор, пока мы не получем результате без ПУ, или контекст не будет включать весь граф. Результат последнего запроса и будет искомым контекстом.

3.5. КОНТЕКСТНО ЗАВИСИМЫЕ ИДЕНТИФИКАТОРЫ

Когда контекст собран, начинается фаза формирования контекстно зависимых идентификаторов. Идея состоит в том, чтобы создать идентификатор, из которого в дальнейшем можно будет вычислить контекст. Для этого в Ontodia контекст преобразовывается в хэш и используется в качестве идентификатора. При этом хэш функция реализует следующие шаги:

1. Граф контекста преобразуется в каноничную форму (см. Листинг 2). Метод преобразования RDF—графа в каноничную форму описан в статье "Canonical Forms for Isomorphic and Equivalent RDF Graphs: Algorithms for Leaning and Labelling Blank Nodes"[5].
2. IRI элементы кодируются JavaScript-функцией `encodeURIComponent`, чтобы данный хэш можно было использовать как часть IRI у компонента.
3. После чего из графа извлекается словарь терминов и выделяется специальный массив, описывающий типы каждого термина (N — NamedNode, B — BlankNode, L — Literal (Строка с указанием языка), D — Literal (Строка с указанием типа), V — Variable, G - DefaultGraph).
4. Словарь и граф, представленный в виде набора четверок (quads), сжимаются и кодируются с использованием префиксного дерева[11] (см. Листинг 3).
5. Полученные элементы объединяются в массив.

6. В полученной строке заменяются символы: `[` → `(`, `]` → `)`, `,` → `:`, `"` → `´`.

7. Далее, для получение финального IRI (см. Листинг 4) к полученному хэшу добавляется индивидуальный индекс и префикс. Префикс различен для разных типов узлов:

   (a) `"ontodia:blank:"` — если одиночный ПУ;

   (b) `"ontodia:list:List"` — если RDF list.

   Последнее делается для того, чтобы по IRI элемента можно было однозначно определить является ли IRI закодированным контекстом или нет, а также для возможности последующего индивидуального отображения списков.

Чтобы получить контекст из закодированного IRI, достаточно выполнить описанные шаги в обратном порядке.

3.6. ВЫДАЧА РЕЗУЛЬТАТОВ ПРЕДОБРАБОТКИ

После того как идентификаторы сформированы, измененный контекст кладется в локальное хранилище, которое, также как и основное, реализует интерфейс `DataProvider`. Впервые выдача результатов, очевидно, происходит в функции `filter`, сразу после предобработки, однако это не единственное место. Например, вызов функции `filter` возвращает набор пустых узлов, которые мы добавляем на граф, при этом информация о связях, входящих в контекст, остается неиспользованной. На следующем шаге процесса отрисовки графа, когда выполняется запрос на получение связей между элементами, эта информация становится полезной. Здесь данные о связях уже берутся не из основного хранилища, а из локального. Доступ к локальном хранилищу более эффективен, и данные здесь уже канонизированы. Полный список запросов, в которых используются предварительно обработанные данные, таков:

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

Когда мы загружаем сохраненную диаграмму, происходит обратное. Как только мы встречаем элемент со специальным префиксом `"ontodia:blank:"` или `"ontodia:list:List"`, мы выполняем восстановление контекста из IRI и кладем результат в локальное хранилище, после чего возвращаем результат, как это было описано выше.

4. ЗАКЛЮЧЕНИЕ

Разработанный метод решает задачи визуализации и восстановления сохраненного графа. При сохранении графа сохраняются и идентификаторы узлов, а поскольку проблемная часть (контекст с ПУ) кодируется в идентификаторах, она также легко восстанавливается при загрузке сохраненного графа. Метод описан в общем виде и подходит для ленивой визуализации ПУ онтологических графов не только с использованием инструмента Ontodia, но и при работе с другими инструментами визуализации. Следующим шагом развития данного метода может быть формализация обработки специальных структур, использующих ПУ в качестве структурных компонентов:

- Список (RDF list);
- Аксиома (`owl:Axiom`);
- Ограничение (`owl:Restriction`);
- Взаимное различие (`owl:AllDifferent`);
- Объединение (`owl:unionOf`);
- Пересечение (`owl:intersectionOf`);
- Дополнение (`owl:complementOf`);
- Перечисление (`owl:oneOf`)[12].

. Например, RDF list может быть предварительно обработан и визуализирован на диаграмме в виде узла-таблицы с сохранением оригинального порядка следования элементов.

В то же время визуализация данных часто граничит с задачей визуального редактирования данных. В этом направлении имеется также

```
{
    "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-2
               0031209/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/2002/07/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" }
}
```

Листинг 2: Часть канонизированного графа контекста в каноничной форме, включающая указатель на целевой элемент

```
[[
    [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%2Fowl%23",[0,"
       Class",1,"Restriction",1,"equivalentClass",1,"hasValue",1,"
       intersectionOf",1,"onProperty",1],"TR%2F2003%2FPR-owl-guide-200312
       09%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]]
```

Листинг 3: Компактное представление графа контекста в формате JSON, полученное из IRI элемента

```
ontodia:blank:sparql2:4:(((0:'http%3A%2F%2Fwww.w3.org%2F':(0:'1999%2F02%2
   F22-rdf-syntax-ns%23':(0:'first':1:'nil':1:'rest':1:'type':1):'2002%2F
   07%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))
```

Листинг 4: Пример сгенерированного IRI

ряд трудностей. Например, представленный алгоритм рассматривает онтологию как нечто завершенное и не подлежащее изменению. В случае, если онтологический граф в области контекста ПУ изменится, все идентификаторы узлов потеряют своё значение, и контекст нужно будет собирать повторно. При этом не всегда просто сказать, была ли изменена онтология в области контекста ПУ, для этого нужно сравнить старый граф контекста с новым, что само по себе часто является нетривиальной задачей. В связи с этим интересно будет адаптировать представленный метод для задачи редактирования онтологических графов, содержащих ПУ.

СПИСОК ЛИТЕРАТУРЫ

1. Topbraid composer maestro edition is a modeling tool and an ide for enterprise solutions. learn how to to model ontologies, connect data sources, design queries, and to develop applications that work with semantic models. `https://www.topquadrant.com/knowledge-assets/faq/tbc/`. Accessed: 25 июня 2020 г..

2. Čerāns, K., Ovčiņņikova, J., Liepiņš, R., and Grasmanis, M. Extensible visualizations of ontologies in owlgred. In *The Semantic Web: ESWC 2019 Satellite Events* (Cham, 2019), P. Hitzler, S. Kirrane, O. Hartig, V. de Boer, M.-E. Vidal, M. Maleshkova, S. Schlobach, K. Hammar, N. Lasierra, S. Stadtmüller, K. Hose, and R. Verborgh, Eds., Springer International Publishing, pp. 191–196.

3. Falco, R., Gangemi, A., Peroni, S., Shotton, D., and Vitali, F. Modelling owl ontologies with graffoo. In *The Semantic Web: ESWC 2014 Satellite Events* (Cham, 2014), V. Presutti, E. Blomqvist, R. Troncy, H. Sack, I. Papadakis, and A. Tordai, Eds., Springer International Publishing, pp. 320–325.

4. Gennari, J. H., Musen, M. A., Fergerson, R. W., Grosso, W. E., Crubézy, M., Eriksson, H., Noy, N. F., and Tu, S. W. The evolution of protégé: an environment for knowledge-based systems development. *International Journal of Human-Computer Studies 58*, 1 (2003), 89 – 123.

5. Hogan, A. Canonical forms for isomorphic and equivalent rdf graphs: Algorithms for leaning and labelling blank nodes. *ACM Trans. Web 11*, 4 (July 2017).

6. Humfrey N. Easyrdf a php library designed to make it easy to consume and produce rdf. `http://www.easyrdf.org`. Accessed: 25 июня 2020 г..

7. Lohmann, S., Link, V., Marbach, E., and Negru, S. Webvowl: Web-based visualization of ontologies. 154–158.

8. 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. In *International Semantic Web Conference (Posters & Demos)* (2015).

9. Nock, C. *Data access patterns: database interactions in object-oriented applications*. Addison-Wesley Boston, 2004.

10. Бессмертный, Визуализация знаний на основе семантической сети. *Программирование 36*, 4 (2010), 16–24.

11. Гудкова, Префиксное сжатие индексов. In *Информатика: проблемы, методология, технологии* (2019), pp. 1310–1314.

12. Заикин, Алгоритм сравнения вариантов онтологий. *Электронные средства и системы управления*, 1 (2010), 132–135.

13. Катериненко, , and Бессмертный, Метод ускорения логического вывода в продукционной модели знаний. *Программирование 37*, 3 (2011), 76–80.