Научная статья

УДК 004.822:004.89
DOI 10.25205/1818-7900-2022-20-4-24-38

Модель представления онтологии предметных областей на основе графовых баз данных

Владислав Александрович Лисин[1], Алексей Сергеевич Серый[2]
Елена Анатольевна Сидорова[3]

[1]Новосибирский государственный университет
Новосибирск, Россия

[2,3]Институт систем информатики им. А.П. Ершова СО РАН
Новосибирск, Россия

[1]vladlisin2@gmail.com
[2]alexey.seryj@iis.nsk.su
[3]lsidorova@iis.nsk.su

Аннотация

В статье представлен подход к моделированию онтологий предметных областей на основе графовых баз данных. Онтология традиционно рассматривается как средство изучения и формализации предметной области. На основе онтологий формируются базы знаний информационных систем, которые в дальнейшем можно пополнять и использовать для исследования определенных прикладных аспектов предметной области. В то же время с развитием технологий NoSQL и графовых баз данных, направленных на оптимизацию работы со связанными данными, появляется возможность проектирования хранилища данных без строгой предустановленной модели предметной области. В силу очевидной графовой природы онтологий графовые базы данных выступают перспективным решением для хранения и работы с онтологиями. Тем не менее при всем сходстве онтологии и модели данных применяемые в графовых СУБД не так просто объединить. В работе предложен подход к интеграции онтологической и графовой модели представления знаний и созданию веб-редактора онтологий. Описываются визуальные средства редактирования онтологий, приведен алгоритм послойной отрисовки ориентированного графа, описан механизм генерации динамических форм для редактирования классов и экземпляров онтологии, а также реализована базовая поддержка паттернов онтологического проектирования.

Ключевые слова

онтология, редактор онтологии, графовая база данных, визуализация графа, алгоритм послойной отрисовки направленного графа, редактор паттернов содержания

Для цитирования

Лисин В. А., Серый А. С., Сидорова Е. А. Модель представления онтологии предметных областей на основе графовых баз данных // Вестник НГУ. Серия: Информационные технологии. 2022. Т. 20, № 4. С. 24–38. DOI 10.25205/1818-7900-2022-20-4-24-38

Domain Ontology Representation Model Based on Graph Databases

Vladislav A. Lisin[1], Aleksey S. Sery[2], Elena A. Sidorova[3]

[1]Novosibirsk State University
Novosibirsk, Russian Federation

[2,3]Ershov Institute of Informatics Systems SB RAS
Novosibirsk, Russian Federation

[1]vladlisin2@gmail.com
[2]alexey.seryj@iis.nsk.su
[3]lsidorova@iis.nsk.su

Abstract

The article presents an approach to modeling domain ontologies based on graph databases. Ontology is traditionally considered as a means of studying and formalizing the subject area. Based on ontologies, knowledge bases of information systems are formed, which can later be replenished and used to study certain applied aspects of the subject area. At the same time, with the development of NoSQL technologies and graph databases aimed at optimizing work with related data, it becomes possible to design a data warehouse without a strict pre-established domain model. Due to the obvious graph nature of ontologies, graph databases are a promising solution for storing and working with ontologies. However, with all the similarities, the ontology and data models used in graph DBMSs are not so easy to combine. The paper proposes an approach to the integration of ontological and graph models of knowledge representation and considers its application in creating a database for a prototype of a web ontology editor. Visual tools for editing ontologies are described, an algorithm for layer-by-layer rendering of a directed graph is given, a mechanism for generating dynamic forms for editing classes and ontology instances is described, and basic support for ontological design patterns is implemented.

Keywords

ontology, ontology editor, graph database, graph visualization, directed graph layer-by-layer drawing algorithm, content pattern editor

For citation

Lisin V. A., Sery A. S., Sidorova E. A. Domain Ontology Representation Model Based on Graph Databases. *Vestnik NSU. Series: Information Technologies*, 2022, vol. 20, no. 4, pp. 24–38. (in Russ.) DOI 10.25205/1818-7900-2022-20-4-24-38

Введение

Онтология традиционно рассматривается как средство изучения и формализации предметной области. На основе онтологий формируются базы знаний информационных систем, которые в дальнейшем можно пополнять и использовать для исследования определенных прикладных аспектов предметной области. Описание декларативных знаний в виде классов, объектов и отношений между ними формирует семантический базис для коммуникации между человеком и компьютерными агентами. Для формального представления онтологий традиционно применяются модель RDF и язык OWL [1], позволяющие хранить данные в виде множества RDF-триплетов, используя для этого одно из известных хранилищ, например Jena TDB [2] или Virtuoso [3], и язык запросов SPARQL. При всех своих достоинствах хранилища триплетов часто проигрывают в скорости обработки запросов популярным СУБД, вынуждая разработчиков информационных систем, основанных на онтологиях, искать альтернативные пути представления своих данных, используя, в том числе, реляционные и графовые СУБД [4].

Технологии традиционных реляционных баз данных сфокусированы на оптимальной организации данных для повышения эффективности их хранения, управления и поиска. С другой стороны, набирающая популярность технология графовых баз данных представляет собой решение, ориентированное на поэлементный просмотр компонентов графа, понимание их взаимосвязей, а не на пакетную обработку данных. Несмотря на взаимодополняемость и взаимосвязь между этими технологиями их полная интеграция и взаимодействие еще не стандар-

тизованы, в том числе по причине отсутствия формализма [5], описывающего схемы графовых баз данных и обеспечивающего совместимость данных, а также задания соответствия между графовой моделью БД, выраженной этим формализмом, и стандартной онтологической моделью. В силу очевидной графовой природы онтологий графовые базы данных выступают перспективным решением для хранения и работы с онтологиями. В то же время при всем сходстве структуры NoSQL баз данных RDF и NoSQL графовых баз данных относятся к разным графовым моделям, которые не так просто объединить.

Графовые базы данных предоставляют более обобщенную структуру, основанную на модели LPG – Labeled Property Graphs (графы меток и атрибутов) [6]. В данной структуре узлы и дуги могут обладать персональными именами (метки и типы соответственно), а также способны хранить свойства, представленные в виде полей «ключ – значение». Также LPG предоставляет возможность создания множества дуг между узлами графа, наделяя каждую дугу уникальным идентификатором. Основными отличиями между LPG и RDF являются:

– атомарность узлов RDF: возможность хранения информации непосредственно в узлах LPG обеспечивает более компактную структуру, уменьшая общее число необходимых узлов в десятки раз;

– RDF не способен находить различие между встречающимися одинаковыми связями между одинаковыми элементами, в то время как LPG предоставляет уникальные идентификаторы дуг и узлов;

– RDF не поддерживает прикрепление свойств к экземплярам связей (данное ограничение можно преодолеть при помощи трансформации экземпляра связи в объект, но это влечет ухудшение читабельности);

– RDF поддерживает атрибуты, принимающие множество значений (данное ограничение преодолевается в LPG путем хранения данных атрибута в соответствующем векторе);

– в LPG есть только один вид узлов, а в RDF-графах – два (URI или литеральные значения для объектов триплетов).

В табл. 1 представлено сопоставление терминов, используемых для обозначения схожих концептов в базах данных, онтологиях и LPG.

Таблица 1

Сопоставление терминов баз данных, онтологий и LPG

Table 1

Matching Database, Ontology, and LPG Terms

Базы данных	Онтологии	LPG
Экземпляр Значение Атрибут Отношение Таблица	Экземпляр Значение Свойство – значение (Datatype Property) Свойство – объект (Object Property) Класс	Узел / Дуга Значение Свойство Дуга (тип) Узел (метка)

В данной работе предлагается модель представления онтологий предметных областей на основе графовых баз данных, а также приведено описание подхода к разработке инструмента для их визуализации и редактирования.

Оставшаяся часть статьи организована следующим образом. В разделе 1 представлен метод организации хранения онтологий и графов знаний в графовых базах данных. Раздел 2 посвящен описанию алгоритма послойной отрисовки графа на плоскости для визуального представления онтологий. В разделе 3 приведено описание редактора онтологий, включающего конструктор паттернов онтологического проектирования и систему полнотекстового поиска и фильтрации.

1. Представление онтологии в формате графовой базы данных

В областях научной деятельности, связанных с компьютерными технологиями, часто используется общее определение онтологии Т. Грубера [7]. Грубер описывает онтологию как полную спецификацию концептуализации. Она представляет собой описание предметной области (концептуализации), состоящее из системы понятий, определений и описаний предметов, а также методов исследования. Основные составляющие онтологии – это сущности (классы, концепты) предметной области, отношения между классами и экземпляры классов, также называемые объектами.

На 12-й Европейской конференции по искусственному интеллекту в 1996 г. А. Бернарас, И. Ларесгоити и Х. Корера в своей работе [8] предложили метод построения онтологии предметной области электрических сетей. Работа была проделана в рамках проекта Esprit KACTUS (KACTUS is a European ESPRIT project). Одной из ключевых задач проекта являлась проверка возможности повторного использования знаний в сложных технических системах, а также использование онтологий для поддержки данного процесса. Метод заключался в гибком построении онтологий – архитектура и требования, накладываемые на онтологию, были тесно связаны с разработкой программного обеспечения. При разработке новой программы параллельно шел процесс построения онтологии для представления знаний, применяемых при разработке. Онтологии такого вида могут быть построены путем повторного использования фрагментов других онтологий, а затем, в свою очередь, могут применяться как составные части онтологий, которые будут разрабатываться позднее. Таким образом, была предложена методология разработки онтологий, использующая уже созданные онтологии и/или их фрагменты.

В данной работе была разработана модель представления онтологии в виде ориентированного помеченного графа (Labeled Property Graph, LPG), совместимого с моделью представления данных графовой БД Neo4j, обеспечивающая поддержку рассмотренной выше методологии. Предлагаемая модель обеспечивает возможность одновременного хранения нескольких онтологий в одном графе, а также заимствования элементов существующих онтологий при создании новых онтологий без необходимости дублировать данные.

1.1. Метки вершин в графовом представлении онтологии

Элементы онтологий, такие как классы, свойства, экземпляры и т. д. обладают уникальными идентификаторами (URI), позволяющими однозначно идентифицировать их принадлежность. В графовом представлении (LPG) эту функцию берут на себя метки. Множество меток вершины графа также включает пространство имен соответствующей онтологии, что позволяет хранить множество онтологий в одном графе. При этом одни и те же вершины могут одновременно быть частью более чем одной онтологии.

Уникальной меткой вершины является ее идентификатор URI. Идентификаторы новых элементов создаются простой конкатенацией пространства имен онтологии и уникального токена, полученного функцией генерации случайной строки. Далее будут приведены примеры основных меток и типов, используемых для описания вершин в графовом представлении онтологий.

На рис. 1 изображен пример проекции owl-описания онтологий и их классов в графовое представление.

На рис. 1 показаны графовые представления двух онтологий Ex1 и Ex2. Для их обозначения на графе созданы вершины с метками **Ontology**. Каждая онтология состоит из одного класса с именем C1. Как можно видеть из рис. 1, вершина C1, представляющая класс онтологии Ex1, имеет метку **Class**, метку http://www.ex1.com, указывающую на принадлежность класса онтологии Ex1 и метку http://www.ex1.com/c1 – URI класса. Аналогичным образом устроена вершина, соответствующая классу онтологии Ex2. Несмотря на идентичность имен двух

```
<owl:Ontology rdf:about="http://www.ex1.com">
    <dc:title>Ex1</dc:title>
    <dc:description>An example 1</dc:description>
</owl:Ontology>

<owl:Class rdf:about="http://www.ex1.com/c1">
    <rdfs:label>C1</rdfs:label>
    <rdfs:comment>The class of ex1.</rdfs:comment>
</owl:Class>
```

```
<owl:Ontology rdf:about="http://www.ex2.com">
    <dc:title>Ex2</dc:title>
    <dc:description>An example 2</dc:description>
</owl:Ontology>

<owl:Class rdf:about="http://www.ex2.com/c1">
    <rdfs:label>C1</rdfs:label>
    <rdfs:comment>The class of ex2.</rdfs:comment>
</owl:Class>
```

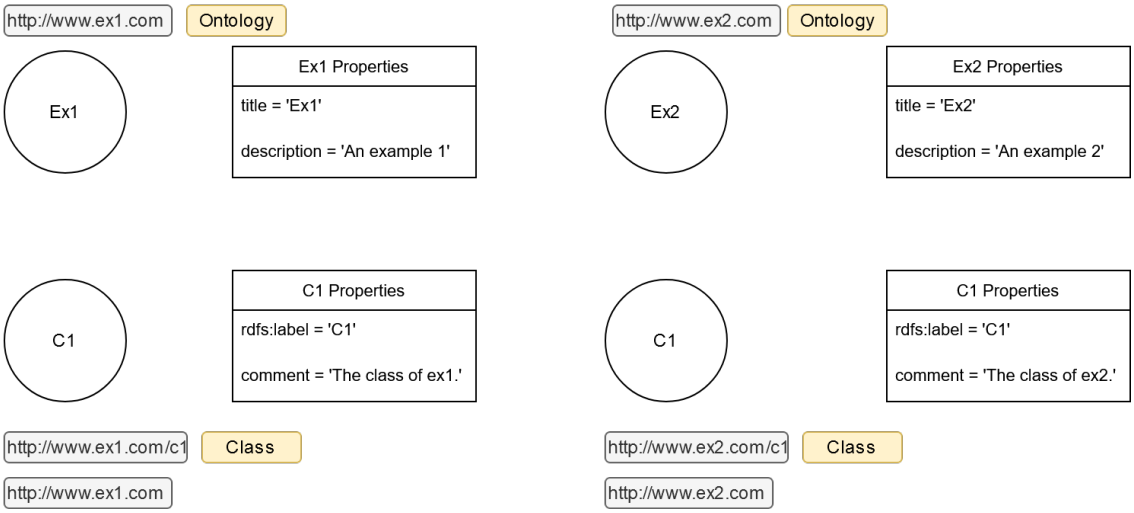

Рис. 1. Пример описания онтологий и их классов в графовой БД
Fig. 1. An example of describing ontologies and their classes in a graph database

классов на рис. 1, им соответствуют две разные вершины графа. Их различие заключается в принадлежности двум разным онтологиям, о чем свидетельствуют метки пространства имен и URI. На рисунке приведены примеры основных меток, применяемых для типизации узлов – **Ontology** и **Class**. Данные метки используются для определения основных типов элементов графа: главный узел онтологии, классы, объекты и узлы атрибутов классов. Использование данных меток упрощает последующие запросы к графовой базе данных.

Для корректного отображения в пользовательском интерфейсе каждый элемент онтологии обладает атрибутом rdfs:label. Данный атрибут заимствован из OWL, и его значениями, как правило, являются корректные названия элемента на всех необходимых языках.

1.2. Иерархия онтологий

Разработанная модель представления позволяет выстраивать онтологии в иерархические структуры. Данное решение избавляет пользователя от необходимости создания каждой онтологии с нуля, предоставляя возможность использовать созданную ранее уже готовую онтологию, расширив ее новыми элементами или, наоборот, взяв ее подмножество. Модель различает базовые и наследуемые онтологии. Онтология, которая не наследуется от других, называется корневой. Метки **Secondary** и **Exclude** присваиваются расширенным и ограниченным онтологиям соответственно. Каждый граф, представляющий онтологию, имеет главную вершину, в которой содержится информация об онтологии в целом, в том числе об имеющихся иерархических связях с другими онтологиями. Подобный подход позволяет расширять и ограничивать уже существующие онтологии без необходимости полного дублирования их содержания. Помимо главной вершины, метками **Secondary** или **Exclude** помечаются и все классы соответствующей онтологии.

При расширении онтологии пользовательскими классами каждому новому элементу присваивается метка **Secondary**. При ограничении некоторой онтологии *O*, т. е. порождении из ее подмножества онтологии *O1*, каждому элементу *O*, который мы хотим исключить из *O1*, присваивается метка, составленная как конкатенация URI онтологии *O1* и ключевого слова **Exclude**. Таким образом, порождается онтология, являющаяся подмножеством существующей. На рис. 2 приведен пример расширения корневой онтологии.

```
<owl:Ontology rdf:about="http://www.ex1.com">
    <dc:title>Ex1</dc:title>
    <dc:description>An example 1</dc:description>
</owl:Ontology>

<owl:Class rdf:about="http://www.ex1.com/c1">
    <rdfs:label>C1</rdfs:label>
    <rdfs:comment>The class of ex1.</rdfs:comment>
</owl:Class>
```

```
<owl:Ontology rdf:about="http://www.ex2.com">
    <dc:title>Ex2</dc:title>
    <dc:description>An example 2</dc:description>
    <rtat:primary>http://www.ex1.com</rtat:primary>
</owl:Ontology>

<owl:Class rdf:about="http://www.ex2.com/c2">
    <rdfs:label>C2</rdfs:label>
    <rdfs:comment>The class of ex2.</rdfs:comment>
</owl:Class>
```

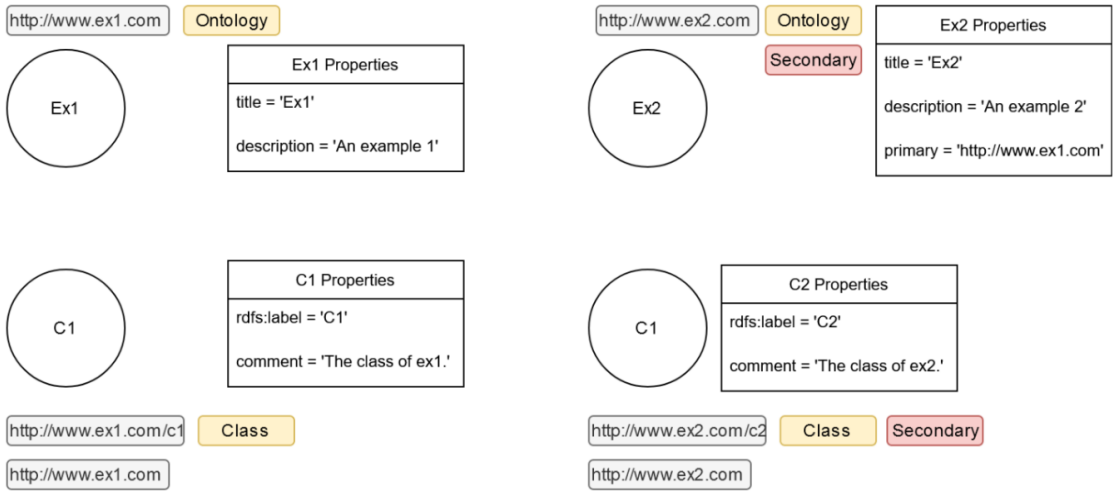

Рис. 2. Пример наследования онтологией элементов корневой онтологии с использованием меток типа Secondary на графе

Fig. 2. An example of ontology inheritance of elements of the root ontology using labels of the “Secondary” type on the graph

Здесь узлу Ex2 присвоен атрибут primary, в котором записан URI онтологии, от которой наследуется онтология Ex2, т. е. онтологии Ex1. Таким образом, при поиске элементов онтологии Ex2 к запросу будут прикреплены все элементы онтологии Ex1 – в частности класс C1. Метки Secondary используются для упрощения фильтрации запросов к графовой БД.

1.3. Свойства классов

Класс в разработанной модели представляет собой узел в графе, помеченный основной меткой **Class**. Для задания и отображения иерархии классов используются дуги с основным типом **subClassOf**.

Представлением атрибута класса является узел, наделенный основной меткой **DatatypeProperty**. URI атрибута задается типом дуги, связывающей узел **DatatypeProperty** и узел класса. Связи класса представляются вершинами с меткой **ObjectProperty**. URI отношения задается типом дуги, связывающей узел **ObjectProperty** и узел класса.

На рис. 3 приведен пример иерархии классов C1 и C3 и описание **DatatypeProperty** и **ObjectProperty** класса C3.

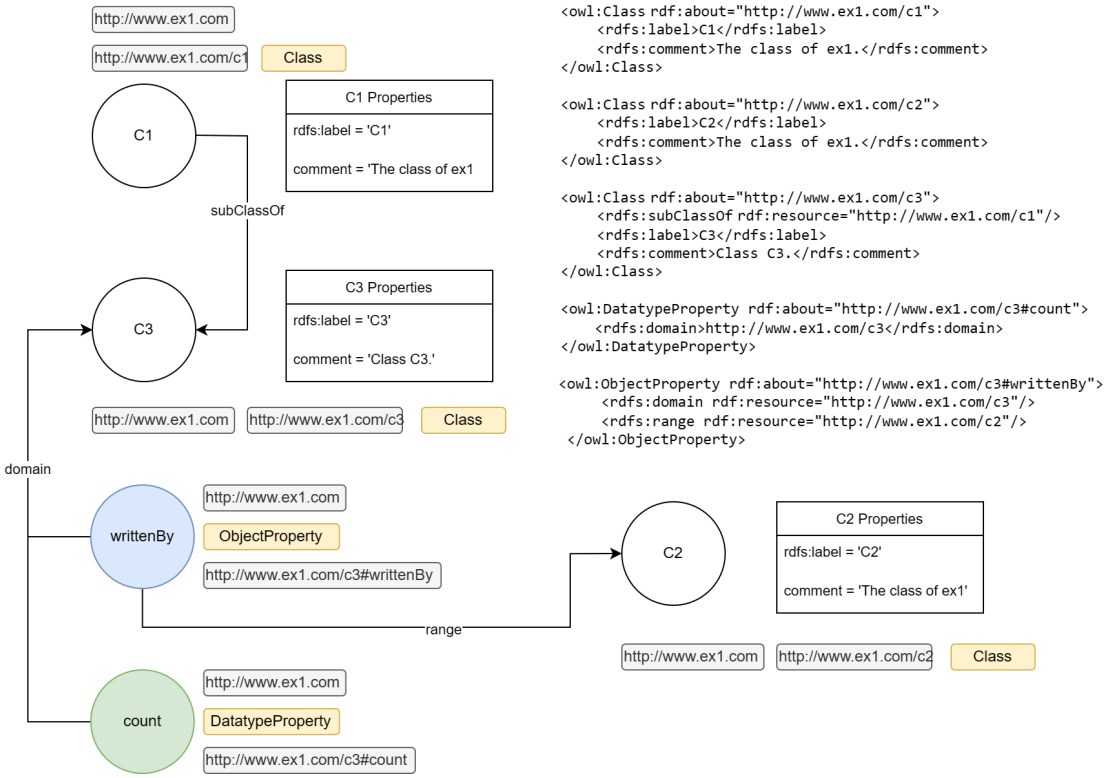

Рис. 3. Пример реализации иерархии классов онтологии и их атрибутов на графе
Fig. 3. An example of the implementation of the hierarchy of ontology classes and their attributes on a graph

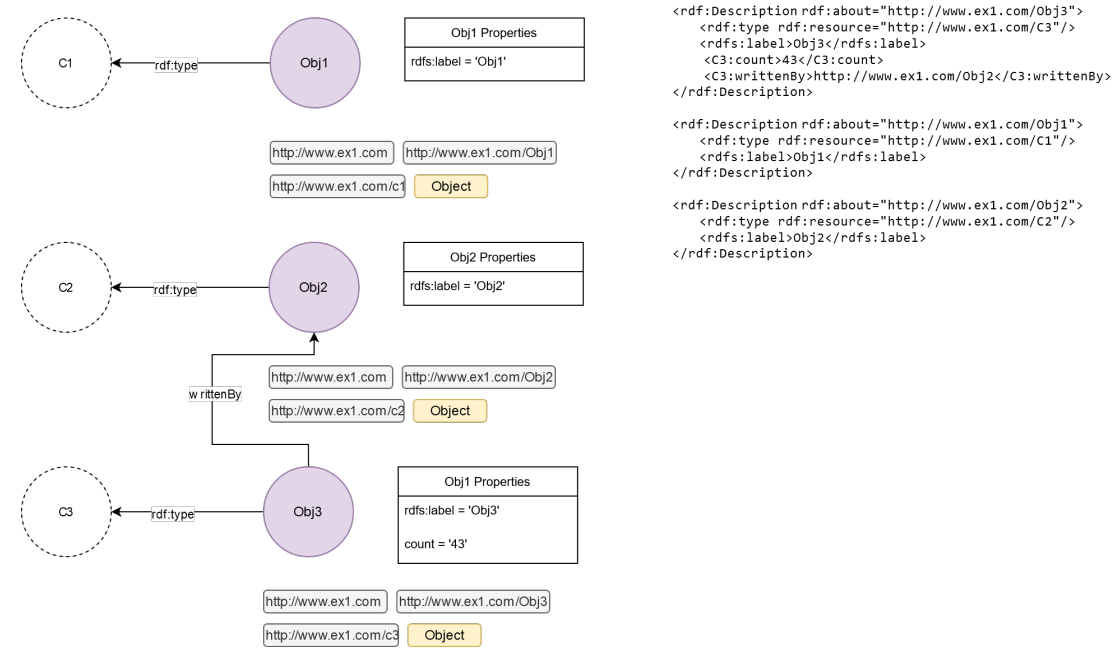

Рис. 4. Пример реализации экземпляров классов C1, C2 и C3 онтологии на графе
Fig. 4. An example of the implementation of instances of classes C1, C2 and C3 of an ontology on a graph

1.4. Свойства экземпляров классов

Каждый экземпляр класса онтологии на графе представляется узлом с меткой **Object**. Каждый такой объект связан со своим классом дугой типа **rdf:type**. Атрибуты объекта хранятся в структуре dict узла объекта. Связи объекта строятся через дуги соответствующих типов, определенные в классе объекта. На основе примера, изображенного на рис. 3, на рис. 4 приведен пример экземпляров классов C1, C2 и C3.

2. Визуализация онтологии

Визуализация является неотъемлемой частью процесса исследования и анализа онтологий предметных областей. Корректная и наглядная визуализация предоставляет инженеру знаний средства для эффективного управления, просмотра и навигации по онтологиям, а также – получения необходимой информации и выводов на основе данных онтологий. Кроме того, непрерывный рост количества и размеров онтологий требует современных и качественных методов визуализации элементов, способных справляться с возникающими алгоритмическими проблемами масштабируемости.

Из этого следует, что разработчики средств визуализации и редактирования онтологий должны принять меры по улучшению дизайна посредством анализа существующих и создания новых инструментов, обладающих более продвинутой функциональностью для конечных пользователей.

В данной работе для визуализации онтологии был выбран алгоритм послойной отрисовки графа, основанный на работе [9], посвященной техникам представления ориентированных графов. Реализация выполнена средствами библиотеки Javascript Dagre. Этот алгоритм прост в реализации, дает наглядное послойное отображение графа с возможностью интерактивной настройки расстояний между слоями или объектами слоя, а также достаточно эффективен для интерактивного использования. Последнее является ключевым фактором в его применении на ReactFlow.

По результатам анализа разработанных вручную визуальных представлений конечных автоматов, авторы в [9] сформулировали следующие принципы эстетического восприятия, которым должно удовлетворять визуальное представление графа.

1. Дуги графа следует по возможности располагать таким образом, чтобы они были ориентированы примерно в одном направлении, например сверху вниз. Такой подход придает графу иерархическую структуру и помогает пользователю в навигации по онтологии предметной области, в частности наследования атрибутов классов.

2. Следует избегать пересечения дуг и их изгиба под острыми углами. По мнению авторов [8], подобные визуальные аномалии искажают восприятие графа.

3. Дуги должны быть как можно более короткими, что делает более наглядными связи между вершинами.

4. По возможности отдавать предпочтение симметрии и балансу при расположении вершин. Данный принцип задает общий подход, которому необходимо следовать для создания корректных и наглядных визуальных представлений и обеспечивает минимизацию рабочего пространства, в котором расположен граф.

Для рендера графа (т. е. получения изображения по модели) был использован ReactFlow – инструмент на основе языка Javascript и фреймворка React с широкими возможностями настройки для создания редакторов на основе узлов и интерактивных диаграмм.

Для графового представления онтологии была предложена следующая цветовая схема:

– синий цвет узла – класс онтологии;
– красный цвет узла – экземпляр класса;
– оранжевый цвет узла – атрибут класса.

Согласно алгоритму визуализации, на первых слоях каждого подграфа располагаются вершины, соответствующие классам онтологии с последующим спуском к их экземплярам. Каждая дуга помечена названием соответствующего ей отношения в онтологии. Каждый подграф или группа вершин могут быть произвольно перемещены для организации рабочего пространства пользователя.

На рис. 5 показан пример визуализации графа, соответствующего фрагменту онтологии из области фольклора «Культурные универсалии народов Сибири» [10].

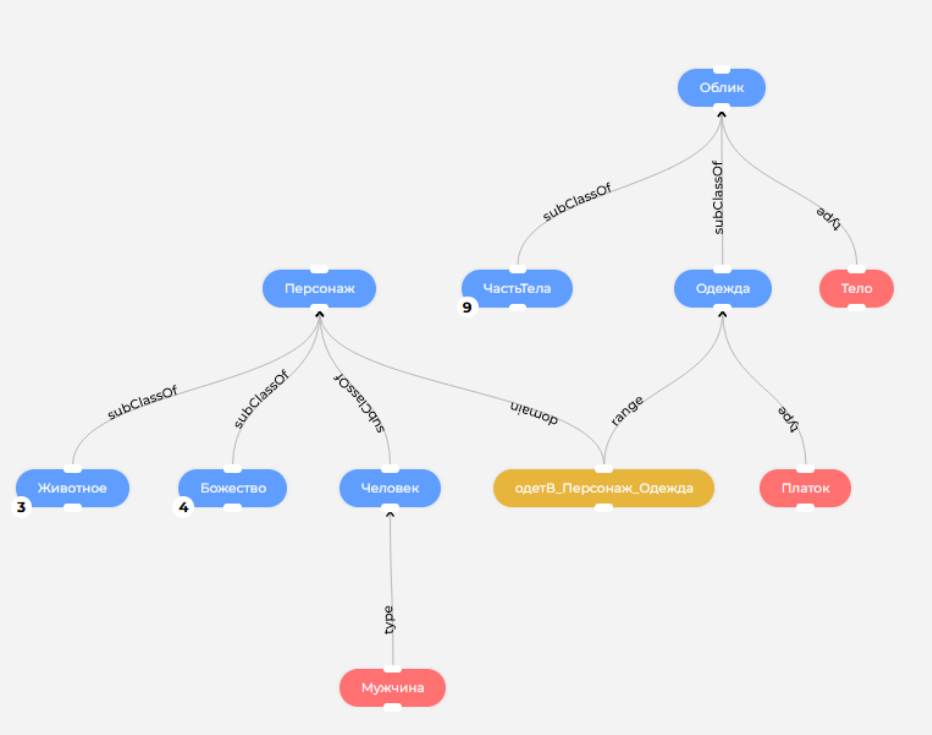

Рис. 5. Визуализация фрагмента онтологии из области фольклора «Культурные универсалии народов Сибири» на основе алгоритма послойной отрисовки графа
Fig. 5. Visualization of a fragment of the ontology from the field of folklore “Cultural universals of the peoples of Siberia” based on the algorithm of layered graph rendering

Объекты классов «Животное», «Божество» и «Часть тела» были скрыты разработанной функцией интерактивной фильтрации элементов графа. Число скрытых объектов определяется специальным маркером, расположенным в левой нижней части маркера узла (у «Животного» скрыто 3 объекта, у класса «Божество» – 4 объекта, у класса «Часть тела» – 9 объектов). Несмотря на то что класс «Персонаж» является корневым, его расположение на графе относительно других элементов было смещено на один слой ниже, чтобы избежать пересечения дуг.

3. Редактор онтологий

Для реализации модели представления и средств визуализации, приведенных в разделах 1 и 2, разрабатывается веб-редактор онтологий. Данный редактор предоставляет пользователю возможность работать непосредственно с графовыми представлениями онтологий и включает следующие базовые функции.

А. Создание и редактирование элементов: корневых вершин онтологий, вершин-классов, вершин-экземпляров и т. д. Таким образом, пользователь редактора имеет возможность добавить новую или отредактировать существующую онтологию.

B. Полнотекстовый поиск вершин. При положительном результате поиска граф в рабочем пространстве ориентируется таким образом, чтобы найденная вершина занимала центральную позицию.
C. Фильтрация вершин путем выбора определенного набора меток.
D. Создание онтологических паттернов содержания в виде «фрагмента» онтологии, снабженного описанием.
E. Применение паттернов при разработке онтологий.

3.1. Редактор элементов онтологии

Каждый элемент графа является представлением определенного элемента онтологии. Для совершения действий над элементом в интерфейсе приложения-редактора предусмотрено персонализированное контекстное меню, пример которого показан на рис. 6.

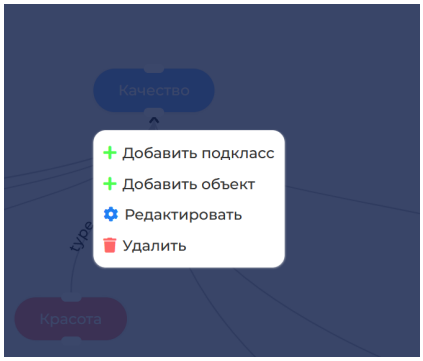

Рис. 6. Контекстное меню узла графа
Fig. 6. Context menu of the graph node

Содержимое данного меню зависит от типа узла графа. Для объектов онтологии существуют опции редактирования и удаления, в то время как для классов предусмотрен список отфильтрованных структурных паттернов, а также кнопки для создания их подклассов и объектов.

Таблица 2

Запросы для извлечения информации о структуре объектов

Table 2

Queries to Retrieve Information about the Structure of Objects

Запрос	Описание
match(n) - [:`{subClass}`*] -> (s) <-[:`{domain}`] - (d: `{type}`) where n.uri = ‘{uri}’ return d	Сбор всех Datatype атрибутов (d) всех родителей класса n (s), где URI класса (n) равен переменной (uri)
match (d:`{dataType}`)-[:`{domain}`]->(n) where n.uri = ‘{uri}’ return d	Сбор Datatype атрибутов (d) конкретного класса (n), где URI класса (n) равен переменной (uri)
match(n)-[:`{subClass}`*]->(s)<-[:`{domain}`]-(d:`{type}`)-[:`{range}`]->(r) where n.uri = ‘{uri}’ return d,r	Сбор Object атрибутов (d) и их типов (r) всех родителей класса n (s), где URI класса (n) равен переменной (uri)
match(n)<-[:`{domain}`]-(d:`{type}`)-[:`{range}`]->(r) where n.uri = ‘{uri}’ return d,r	Сбор Object атрибутов (d) и их типов (r) конкретного класса (n), где URI класса (n) равен переменной (uri)

Разрабатываемый редактор предполагает его применимость для любой предметной области, поэтому поля редактора генерируются автоматически с помощью шаблонных запросов

на языке запросов Cypher графовой СУБД Neo4j. В табл. 2 представлены примеры шаблонных запросов для генерации полей.

На рис. 7 приведен пример формы для создания/редактирования объектов класса «Персонаж», сгенерированной на основе приведенных запросов. Фрагмент структуры класса «Персонаж» представлен на рис. 5.

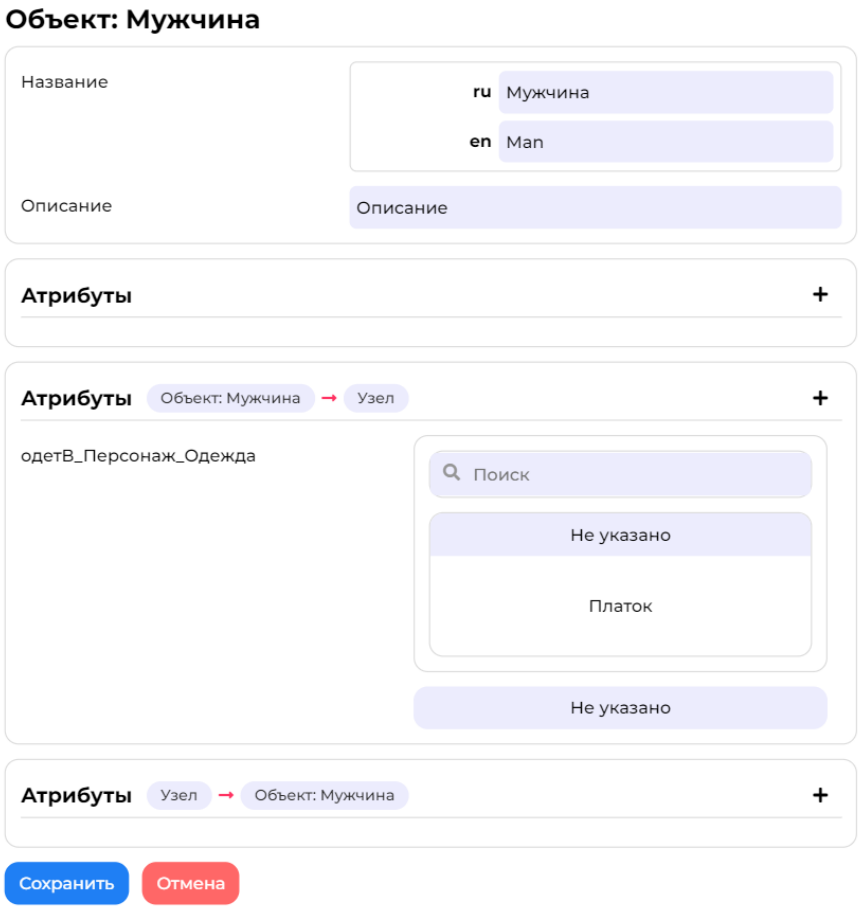

Рис. 7. Форма редактирования узла
Fig. 7. Node editing form

В соответствии с онтологическим представлением, объекты класса «Персонаж» имеют атрибуты (DatatypeProperty) «Название» и «Описание», которые пользователь может заполнить в верхней части формы. Связи объекта с другими объектами (ObjectProperty) заполняются в нижней части формы, в данном примере связь «Персонаж_одетВ_Одежда».

3.2. Редактор паттернов содержания

Чтобы упростить разработку онтологий, была поставлена задача обеспечения поддержки паттернов онтологического проектирования (Ontology Design Patterns, ODP) [11]. На практике чаще всего используются онтологические паттерны содержания (Ontology Content Design Patterns) – небольшие целостные фрагменты онтологии, формализующие обобщенные ситуации предметной области (например, участие в событии, исполнение роли, наличие частей у объекта и др.). Паттерны содержания применяются в качестве строительных блоков при разработке и редактировании онтологий.

Для поддержки онтологических паттернов в редакторе онтологии был разработан инструмент для создания макетов паттернов на основе классов текущей предметной области. В основу редактора макетов входят 5 основных строительных блоков: «якорь», «создание», «существование», «выбор», «условие».

Узел «якорь» представляет собой начальную точку, от которой будет построен паттерн. Узел выражает тип элемента онтологии – класс или объект некоторого класса. Узел «создание» описывает элемент онтологии, который будет создан при применении паттерна. Узел «существование» является уже существующим элементов онтологии, который будет прикреплен к некому узлу паттерна, а узел «выбор» – существующий элемент онтологии, который будет выбран непосредственно при применении паттерна.

Для установления более детализированных условий на вышеописанные узлы используется узел «условие». Данный тип узлов не используется при непосредственном построении, а диктует тип используемых узлов путем установления ограничений на их связи.

Рассмотрим данный инструмент на элементарном примере: создание паттерна «Добавить сына». На рис. 8 изображен финальный макет паттерна «Добавить сына».

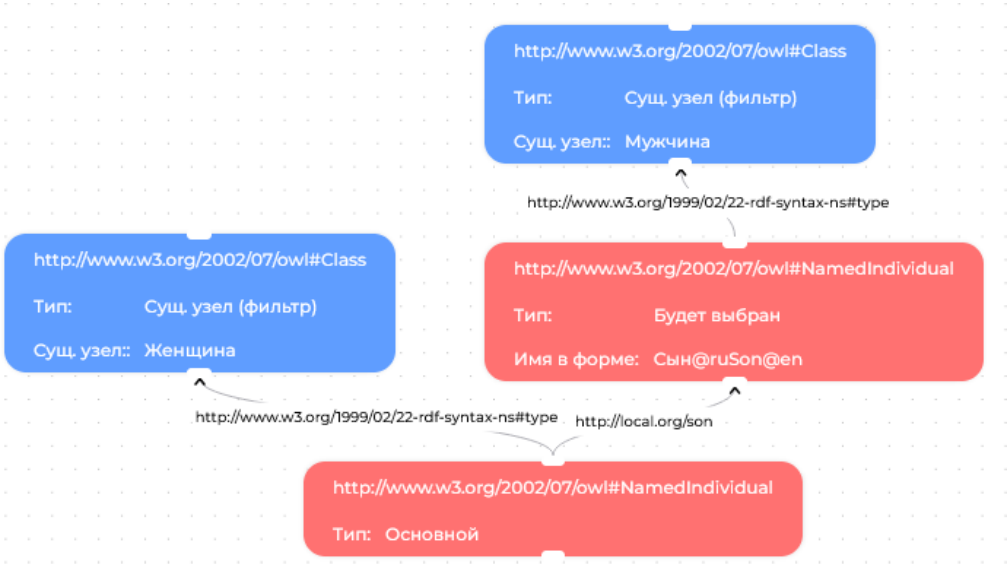

Рис. 8. Паттерн «Добавить сына»
Fig. 8. Pattern «Add son»

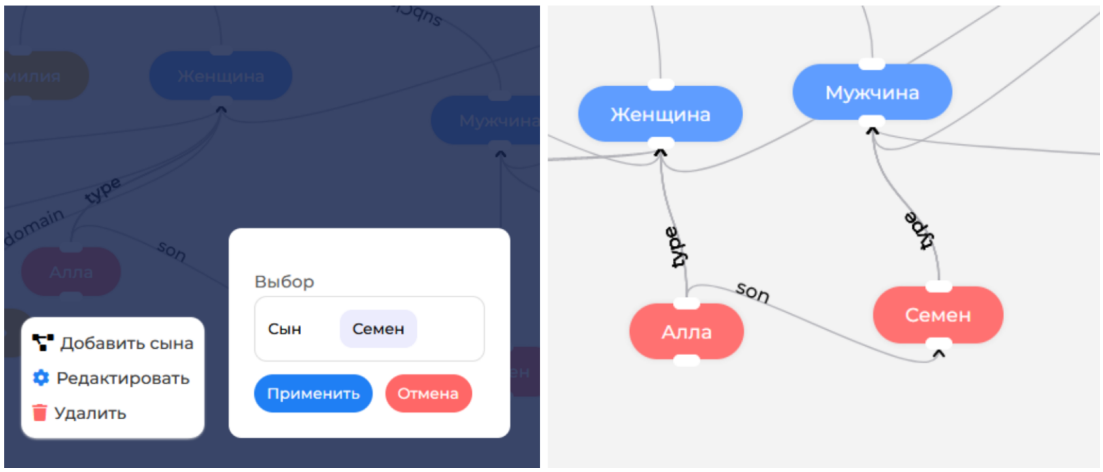

Рис. 9. Результат применения паттерна
Fig. 9. The result of applying the pattern

Применение паттернов осуществляется через контекстное меню определенного узла. Благодаря использованию узлов «условия», инструмент автоматически привязывает определенные паттерны к нужным узлам типа «якорь». На рис. 9 представлен результат применения паттерна «Добавить сына» к объекту «Алла» класса «Женщина».

3.3. Поиск и фильтрация

Приложение содержит функцию простого полнотекстового поиска по узлам. При обнаружении элемента у пользователя имеется возможность сфокусировать рабочее пространство графа на найденном элементе с его последующей подсветкой.

Фильтрация предполагает наличие фильтров по типам узлов графа. Такими фильтрами являются:

– фильтр по классам;
– фильтр по классам и их атрибутам;
– фильтр по объектам;
– фильтр по объектам и их атрибутам;
– фильтр по классам и объектам.

Заключение

В данной работе предлагается метод представления онтологий в графовых базах данных на примере графовой БД Neo4j. Данный метод основан на принципе наследования и расширения онтологий и предполагает использование специальных меток для обозначения ключевых элементов, таких как онтология, класс, экземпляр, атрибут, отношение, а также иерархических связей между онтологиями.

Создаваемый веб-редактор использует графовое представление онтологий, а для визуализации вершин и ребер графа — алгоритм послойной отрисовки помеченных графов, что позволяет повысить наглядность и визуальную привлекательность пользовательских онтологий. Особенностью предлагаемого решения является поддержка применения паттернов онтологического проектирования. Данный функционал позволит в дальнейшем создать полноценное хранилище и среду разработки паттернов онтологического проектирования, а также обеспечит их автоматическую интеграцию с разрабатываемыми пользовательскими онтологиями.

Список литературы

1. OWL 2 Web Ontology Language Document Overview (Second Edition) [Электронный ресурс]. URL: https://www.w3.org/TR/owl2-overview (дата обращения: 13.01.2023).
2. **Curé O, Blin G.** RDF database systems: triples storage and SPARQL query processin // Burlington: Morgan Kaufmann, 2014.
3. Virtuoso [Электронный ресурс]. URL: https://virtuoso.openlinksw.com (дата обращения: 13.01.2023)
4. **Ben Mahria B., Chaker I., Zahi A.** An empirical study on the evaluation of the RDF storage systems // Journal of Big Data. 2021. Pp. 1–20. DOI 10.1186/s40537-021-00486-y
5. **Stefano Ferilli.** Integration Strategy and Tool between Formal Ontology and Graph Database Technology // Electronics. 2021. V. 10(21). P. 2616. DOI 10.3390/electronics10212616
6. **Jesús Barrasa.** RDF Triple Stores vs. Labeled Property Graphs: What's the Difference? [Электронный ресурс] // URL: https://neo4j.com/blog/rdf-triple-store-vs-labeled-property-graph-difference (дата обращения: 13.01.2023).
7. **GruberT. R.** Toward Principles for the Design of Ontologies Used for Knowledge Sharing // International Journal Human-Computer Studies. 1995. V. 43. Pp. 907–928.

8. **Bernaras, A.; Laresgoiti, I.; Corera, J.** Building and Reusing Ontologies for Electrical Network Applications // Proceedings of the European Conference on Artificial Intelligence. ECAI 1996. V. 96. Pp. 298–302.
9. **Gansner, Emden & Koutsofios, Eleftherios & North, Stephen & Vo, Khoi.** A Technique for Drawing Directed Graphs // IEEE Transactions on Software Engineering. 1993. V. 19. Pp. 214–230.
10. **Гриневич А., Серый А.** Анализ культурных универсалий фольклора народов Сибири и Дальнего Востока // Материалы XXIII Международной конференции по аналитике и управлению данными в информационно емких областях (DAMDID/RCDL 2021). Москва, Россия, 26–29 октября 2021 г. Материалы семинара CEUR, 2021. Vol. 3036. С. 387–401.
11. **Gangemi A., Presutti V.** Ontology Design Patterns // Handbook on Ontologies, Eds., Staab, S. and R. Studer. Berlin: Springer Verlag, 2009. Pp. 221–243.

References

1. OWL 2 Web Ontology Language Document Overview (Second Edition) [Online]. URL: https://www.w3.org/TR/owl2-overview/ (accessed on: 13.01.2023).
2. **Curé O, Blin G.** RDF database systems: triples storage and SPARQL query processing. Burlington: Morgan Kaufmann, 2014.
3. Virtuoso [Online]. URL: https://virtuoso.openlinksw.com (accessed on: 13.01.2023).
4. **Ben Mahria B., Chaker I., Zahi A.** An empirical study on the evaluation of the RDF storage systems. *Journal of Big Data*, 2021, pp. 1–20. DOI 10.1186/s40537-021-00486-y
5. **Stefano Ferilli.** Integration Strategy and Tool between Formal Ontology and Graph Database Technology. *Electronics*, 2021, vol. 10(21), p. 2616. DOI 10.3390/electronics10212616
6. **Jesús Barrasa.** RDF Triple Stores vs. Labeled Property Graphs: What's the Difference? [Online]. URL: https://neo4j.com/blog/rdf-triple-store-vs-labeled-property-graph-difference (accessed on: 13.01.2023).
7. **Gruber T. R.** Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *International Journal Human-Computer Studies*, 1995, no. 43, pp. 907–928.
8. **Bernaras A., Laresgoiti I., Corera J.** Building and Reusing Ontologies for Electrical Network Applications. Proceedings of the European Conference on Artificial Intelligence. ECAI 96, 1996. Pp. 298–302.
9. **Gansner E., Koutsofios E., North S., Vo K.** A Technique for Drawing Directed Graphs. Software Engineering, IEEE Transactions on. 19. 1993. pp. 214–230.
10. **Grinevich A., Sery A.** Analyzing the Cultural Universals of the Folklore of Peoples of Siberia and the Far East. Proceedings of the XXIII International Conference on Data Analytics and Management in Data Intensive Domains (DAMDID/RCDL 2021). Moscow, Russia, October 26–29, 2021. CEUR Workshop Proceedings, 2021, Vol-3036. Pp. 387–401.
11. **Gangemi A., Presutti V.** Ontology Design Patterns. Handbook on Ontologies, Eds., Staab, S. and R. Studer. Berlin: Springer Verlag, 2009. Pp. 221–243.

Информация об авторах

Владислав Александрович Лисин, студент аспирантуры

Елена Анатольевна Сидорова, старший научный сотрудник, кандидат физико-математических наук

Алексей Сергеевич Серый, младший научный сотрудник

Information about the Authors

Vladislav A. Lisin, PhD student, Novosibirsk State University (Novosibirsk, Russia)

Elena A. Sidorova, senior researcher, Ph.D., Ershov Institute of Informatic Systems SB RAS, AI laboratory (Novosibirsk, Russia)

Alexey S. Sery, junior researcher, Ershov Institute of Informatic Systems SB RAS, AI laboratory (Novosibirsk, Russia)

Статья поступила в редакцию 26.12.2022;
одобрена после рецензирования 12.01.2023; принята к публикации 12.01.2023

The article was submitted 26.12.2022;
approved after reviewing 12.01.2023; accepted for publication 12.01.2023