КАЗАНСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Кафедра системного анализа и информационных технологий

М.П. ДЕНИСОВ

ЛАБОРАТОРНЫЕ РАБОТЫ ПО ОПИСАНИЮ ПРЕДМЕТНЫХ ОБЛАСТЕЙ С ИСПОЛЬЗОВАНИЕМ СИСТЕМЫ PROTÉGÉ

Учебно-методическое пособие

Казань – 2020

УДК 004.822
ББК 32.813

Принято на заседании кафедры системного анализа и информационных технологий Казанского (Приволжского) федерального университета
Протокол № 8 от 3 июня 2020 года

Рецензенты:
кандидат технических наук,
доцент кафедры САиИТ института ВМиИТ КФУ **П.В. Пшеничный**
кандидат физико-математических наук,
доцент кафедры САиИТ института ВМиИТ КФУ **В.Ю. Михайлов**

Денисов М.П.
Лабораторные работы по описанию предметных областей с использованием системы Protégé / М.П. Денисов. – Казань: Казан. ун-т, 2020. – 53 с.

Учебно-методическое пособие предназначено для студентов, обучающихся в бакалавриате по направлению «Фундаментальная информатика и информационные технологии» и содержит описание заданий в рамках тем по представлению знаний с использованием онтологий по дисциплине «Интеллектуальные системы».

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Для решения большинства задач разработчикам необходимо изучить предметную область и формализовать ее понятия и отношения. Для этого применяются различные подходы к представлению знаний и данных. Одним из таких подходов является онтологический, а созданное в рамках него описание предметной области в данном пособии будет называться базой знаний (БЗ) или онтологией.

Онтологический подход широко используется в существующих разработках. Например: в рамках стандарта медицинского документооборота HL7 есть возможность представлять таким образом информацию об анализах, исследованиях, рецептах и т.д.; унифицированный язык медицинских систем UMLS; модели разметки веб сайтов (schema.org, в частности форматы microdata, RDFa, JSON-LD) для возможности формирования объектов автора, рейтинга, книг, видео, статей и т.д. на основании страницы; семантический поиск (поиск по синонимам, исправление ошибок и т.д.); онтологии, стандартизирующие общие понятия и по конкретным областям (например, visualthesaurus, productontology);

Для унификации вида БЗ разработан ряд стандартов. В рамках данного пособия будут использоваться стандарты, созданные консорциумом всемирной паутины (w3c): модель RDF (Resource Description Framework), языки OWL (Web Ontology Language), SPARQL, SWRL (Semantic Web Rule Language), а также их различные синтаксисы и нотации.

Использование стандартов также позволяет легко переносить разработанную БЗ между разными средствами работы с ней. В частности, можно создавать онтологию с использование инструментальных средств с удобным графическим интерфейсом (в рамках данного пособия будет рассматриваться система Protégé, разрабатываемая в Стэнфорде), а использовать в дальнейшем в рамках своего программного кода (например, для языка Java существует библиотека Apache Jena, рассмотрение которой выходит за рамки данного пособия).

Данное пособие предназначено для получение начальных представлений о приведенных моделях и языках в рамках приведенной системы. Предполагается, что по результатам работы с данным пособием обучаемый сможет самостоятельно составлять несложные базы знаний и осуществлять запросы к ним.

В рамках пособия рассматривается часть задачи сбора и обработки информации о товарах интернет-магазинов. А именно: создание база знаний, которая позволит в дальнейшем накапливать в нее новые товары, в том числе обладающие характеристиками, не заложенными изначально; отладка некоторых механизмов обработки (отнесение товаров к определенным группам, проверка непротиворечивости информации, выбор определенных товаров и их характеристик). Реализация оставшихся подзадач предполагается на языке Java с использование ранее упомянутой библиотеки Apache Jena и будет описана в следующих пособиях.

Материал разделен на 4 лабораторные работы, каждая из которых содержит: пошаговое решение подзадач описанной выше задачи; теоретический материал, необходимый для понимания принятых решений; задания для самостоятельного выполнения.

В рамках первой работы определяется предметная область, создаются классы, отношения с использованием системы Protégé. Также на примере созданной БЗ рассматривается модель RDF, язык OWL и синтаксис Turtle.

Вторая работа описывает понятия Defined Classes, DL Query в системе Protégé и их реализацию в рамках языка OWL. В рамках работы создаются классы, описывающие группы товаров, отнесение к которым экземпляров производится по характеристикам автоматически в процессе вывода.

Третья работа знакомит с языком запросов SPARQL. В рамках работы создается ряд запросов для выбора информации о товарах.

Четвертая работа показывает работу с языками правил SWRL и SQWRL в рамках системы Protégé. В рамках работы создаются правила для проверки информации о товарах.

ЛАБОРАТОРНАЯ РАБОТА 1. ОСНОВЫ RDF и OWL. СОЗДАНИЕ ОНТОЛОГИИ В СИСТЕМЕ PROTÉGÉ

Будет рассмотрена информация о конкретном товаре (конкретном смартфоне [1]) и на ее основании составлена первая версия БЗ.

Перед началом работы с системой Protégé [6] необходимо составить хотя бы некоторую упрощенную концептуальную модель предметной области. И, для того чтобы потом не было сложностей с ее формализацией, необходимо определить какими сущностями можно оперировать. В данном случае это сущности модели RDF [1, 2] и языка OWL [4], т.к. многие системы и библиотеки, в том числе Protégé поддерживает эти стандарты, и они являются наиболее распространенными.

Стоит сразу отметить, что модель RDF является абстрактной и существует множество языков с разными синтаксисами, реализующих ее [1]. OWL также может быть преобразован в несколько нотаций и в любой из языков RDF [4]. Конкретные синтаксисы будут обсуждаться позднее и все примеры, описанные ниже могут их не придерживаться.

Онтология в рамках модели RDF предполагает определение любых понятий предметной области в виде троек: субъект – предикат – объект. Множество таких троек образует граф понятий. Субъекты и объекты интерпретируются как вершины ориентированного графа, а предикаты как его дуги (и определяют отношения между сущностями). Вершины могут быть следующих типов: ресурс, пустая вершина.

Ресурс описывает некоторое понятие реального мира и разделяется на ссылочные ресурсы и скалярные. Ссылочные ресурсы – это некоторые понятия, которые могут иметь характеристики (например, смартфон, смартфон SM-

[1] https://www.dns-shop.ru/product/052629b3621f3330/62-smartfon-samsung-galaxy-s9-256-gb-fioletovyj/characteristics/

G965FZPHZER, экран). Таким понятиям присваивается глобальный уникальный идентификатор IRI (Internationalized Resource Identifier) (например, http://myontololgy.org/goods#phone1). Соответственно, такие ресурсы могут находиться как на месте субъекта, так и на месте объекта в зависимости от предиката.

Скалярные ресурсы указывают некоторые конкретные характеристики и являются просто значением определенного типа. Соответственно, такие ресурсы могут находится только на месте объекта. Но, при этом для них может определяться тип и для строковых значений может определяться язык. Например, для нашей предметной области такими сущностями могут являться: 6.2 (конкретная диагональ экрана; float), «Samsung Galaxy S9+» (оригинальное название; string); «черный»@ru (цвет; string; русский).

Также необходимо отметить, что «предикат» также может быть «субъектом» или «объектом» в рамках других троек. Это необходимо чтобы, например, описать ограничения данного отношения.

Пустая вершина в основном необходима для «имитация» n-арных связей. Как видно из концепции модели, предикат может связывать только 2 ресурса. Но, это решается с помощью введения промежуточной вершины и разделением связи на несколько бинарных. При этом вершина не будет представлять понятие реального мира и ей нет смысла давать уникальный глобальный идентификатор, достаточно локального для составления троек. Именно для таких вершин и введен обсуждаемый тип. При этом не всегда очевидно стоит ли вводить для той или иной сущности глобальной идентификатор. Например, конкретный смартфон обладает рядом характеристик дисплея и было решено выделить дисплей как отдельную сущность. При этом для нас он является неотъемлемой частью смартфона, а производитель смартфона может ставить разные модели дисплея от разных производителей при условии сохранения характеристик. Тогда может быть целесообразно использовать пустую вершину. С другой стороны, если производитель присвоил данному дисплею определенный код, то может быть целесообразно выделить его

как ресурс. Т.к. он может быть установлен и на другой смартфон, а также будет проще реализовывать, например, запросы для определения сервисов, где есть в наличии данный дисплей. Для примера пустой вершины рассмотрим характеристику «разрешение» у конкретного дисплея:

SM-D-G965FZPHSER разрешение _x

_x горизонталь 2960

_x вертикаль 1440

Данные тройки описывают граф, изображенный на рис. 1.

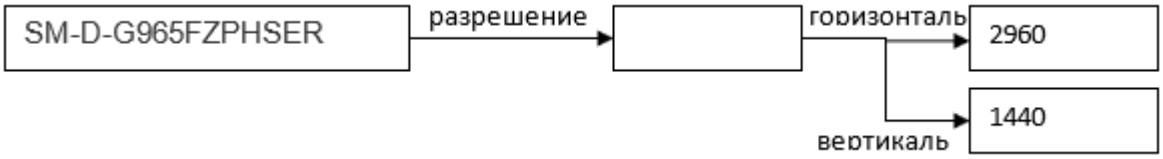

Рис. 1 Граф, описывающий разрешение дисплея

Стоит отметить, что данные понятия также можно было описать с использование 2 связей «разрешение_гор», «разрешение_вер» и без промежуточной вершины. Что показывает, что создание онтологии остается вариативным процессом, зависящим от конкретных целей и предпочтений разработчика.

Как правило IRI ресурса состоит из идентификатора БЗ или пространства имен и идентификатора понятия в нем. Например, «http://myontololgy.org/goods#phone1», может означать, что описывается понятие «phone1» в БЗ товаров с идентификатором «http://myontololgy.org/goods#». Чтобы не писать каждый раз полный идентификатор БЗ или пространства имен, его заменяют на префикс. Например, можно записывать «mo:phone1», где под «mo:» будет подразумеваться полный IRI БЗ.

Стандарт RDF вводит не только концепцию, но и 3 пространства имен с общими понятиями: rdf, rdfs, xsd [3]. Где xsd описывает типы, которые можно использовать в онтологиях на базе RDF (например, xsd:string, xsd:integer, xsd:double, xsd:boolean). Например, разрешение дисплея по горизонтали могло быть описано следующим образом: _x горизонталь “2960”^^xsd:integer. Основные понятия из

rdf, rdfs, которые будут использоваться в рамках данной работы: rdfs:Resource (понятие ресурса, описанного ранее), rdfs:Class (понятие класса), rdf:Property (понятие отношения), rdfs:subClassOf (отношение, которое определяет «субъект» как подкласс «объекта»), rdfs:subPropertyOf (отношение, которое определяет «субъект» как уточнение «объекта», являющегося отношением), rdfs:domain (отношение, определяющее область определения другого отношения), rdfs:range (отношение, определяющее область значения другого отношения), rdf:type (отношение, которое определяет «субъект» как экземпляр «объекта»).

OWL является стандартизированным языком создания онтологий. Он имеет несколько вариантов синтаксиса (в том числе функциональный), а также может быть преобразован в любой из синтаксисов RDF. Т.к. OWL совместим с RDF, то в рамках данного пособия он будет интерпретироваться как надстройка над RDF, которая уточняет подход к созданию онтологий, расширяет возможности модели и вводит новые сущности (из пространства имен owl).

БЗ, созданные с использованием OWL предполагают, что на них производится вывод, который позволяет проверить непротиворечивость БЗ, вывести новые отношения.

В связи с этим описание онтологии можно разделить на:

1. Сущности. Все понятия, описанные ранее в рамках RDF, а также некоторые новые из OWL;
2. Выражения. Описание понятия через комбинацию множества других сущностей. Одним из основных примеров применения является определение класса через выражение. Например, «Камерофон» – это телефон, который имеет не менее 2 основных камер с разрешением не менее 20мегапикселей; новый тип «число» – это объединение множеств целых чисел и чисел с дробной частью.
3. Аксиомы. Точно известные положения и положения, которые всегда должны оставаться истинными. Например, на отношение «плот-

ностьПикселей» могут быть наложены ограничения на область определения – экземпляр класс «Дисплей» и область значения – «целое число». При этом если в базе знаний будет тройка «G плотностьПикселей 529», то это может привести как к ошибке (если G точно не может являться экземпляром класса «Дисплей»), так и к выводу нового отношения «G rdf:type Дисплей».

Как уже было сказано выше, OWL добавляет новые понятия и аксиомы. Для данной работы необходимы следующие понятия:

1. owl:Class. Аналогично rdfs:Class;
2. owl:Thing. Класс, являющийся предком всех классов. Аналогичен Object в некоторых языках программирования;
3. owl:ObjectProperty. Подкласс отношений (rdf:Property), связывающих два ресурса с IRI;
4. owl:DataProperty. Подкласс отношений (rdf:Property), связывающих ресурс с IRI и литерал;
5. owl:topObjectProperty/owl:topDataProperty. Верхнее в иерархии object/data property, являющееся предком всех object/data property;
6. owl:FunctionalProperty. Класс отношений, применяемых к одному «субъекту» только один раз. Т.е. в онтологии не может существовать два триплета «x с y1», «x с y2», где y1 и y2 разные сущности и «с rdf:type owl:FunctionalProperty»;
7. owl:disjointWith. Отношение, декларирующее, что экземпляр субъекта-класса не может быть одновременно экземпляром объекта-класса. Т.е. в онтологии не может существовать два триплета «x rdf:type y1», «x rdf:type y2», где y1 и y2 разные классы и «y1 owl:disjointWith y2».
8. Сущности, необходимые для составления выражений. Будут описывать при появлении (owl:members,, owl:unionOf и т.д.)

Теперь, когда сущности, которые могут быть использованы, обозначены, можно спланировать общее описание предметной области. Оно разделено на описание классов, экземпляров и отношений между ними (рис. 2) и описание самих отношений (рис. 3).

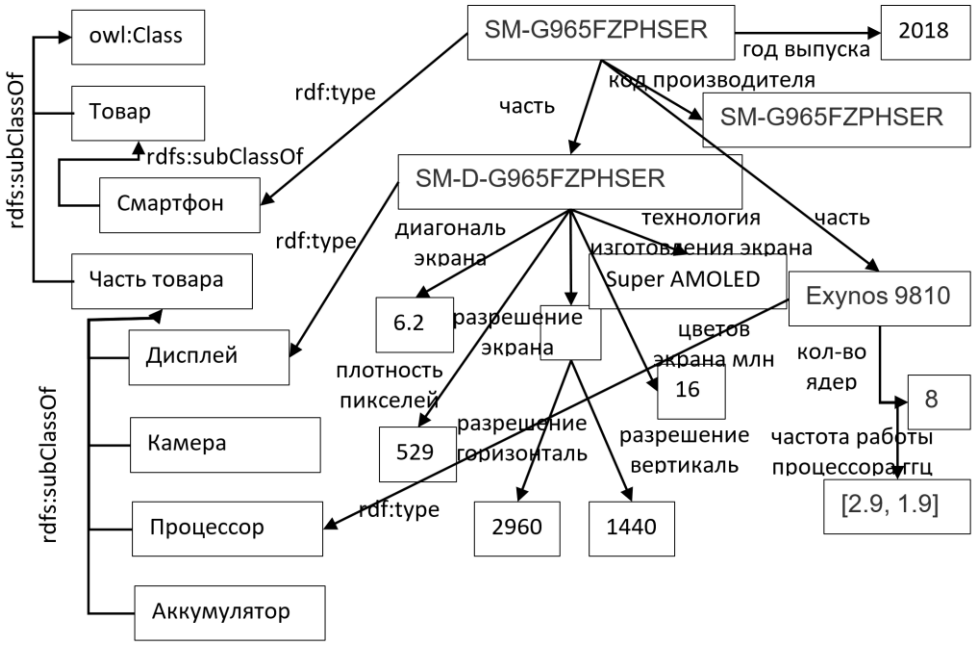

Рис. 2 Фрагмент графа, описывающего классы, экземпляры и их отношения в рамках описания смартфонов

Рис. 3 Фрагмент графа, описывающего свойства в рамках описания смартфонов

В рамках приведенного фрагмента классов выделены очевидные классы: «Товар», «Смартфон». Также выделены некоторые отдельные составляющие смартфона как подклассы класса «Часть товара»: «Дисплей», «Камера», «Процессор», «Аккумулятор». Сюда выделены только самые важные части, которые больше всего интересуют покупателей и наиболее ценны при обработке информации о товаре. Остальные характеристики будут задаваться отношениями между конкретным смартфоном и конкретными значениями (на изображении в качестве примера таких отношений можно видеть: «год выпуска», «код производителя»). Стоит отметить, что некоторые узлы (например, «[2.9, 1.9]») заданы в абстрактном виде и их конкретное задание будет зависеть от используемого синтаксиса и/или возможностей инструментальной системы.

В рамках приведенного фрагмента свойств не сделано разделение на ObjectProperty и DataProperty во избежание излишней конкретизации (при реализации разделение делается очевидным образом). В рамках прямых наследников «owl:Property» выделены типы возможных отношений: «часть» (для связи товара с его частью), «числовое/текстовое/свободное/разрешение» (данные свойства определяют группу свойств, объединенных общей областью определения. Разделение на текстовое и свободное сделано для упрощения дальнейшей обработки. Под текстовыми подразумеваются конкретные значения, которые потом могут быть преобразованы в перечисляемый тип. В рамках свободных же описания могут содержать несколько характеристик, одну характеристику, записанную разным образом и т.д.), «unknown» (группа характеристик, добавляемых автоматически в процессе сбора информации, для которых невозможно определить принадлежность к ранее описанным группам).

Необходимо понимать, что в рамках рис. 2, рис. 3 приведены только фрагменты графа для упрощения. Оставшиеся фрагменты описываются аналогично и не обладают дополнительными особенностями.

Т.к. описание должно подразумевать возможность сбора информации с использованием программных средств, необходимо обговорить как при этом будет происходить добавление новых сущностей:

1. Получить с детальной страницы товара список характеристик в виде «название – текст» и других необходимых отдельных полей (ссылка, код товара, категория и т.д.);
2. Произвести поиск подкласса класса «Товар», соответствующего категории рассматриваемого товара. Если не найден, то создать;
3. Создать экземпляр полученного в п.2 класса (локальная часть IRI формируется как «Код производителя» или как «Код товара» при его отсутствии)
4. Для каждой характеристики
   4.1. Проверить относится ли характеристика к описания какого-то из подклассов класса «Часть товара». Подклассы класса «Часть товара», а также отношения, связанные с ними заданы заранее и не изменяются в процессе сбора информации. Для каждого из таких классов задаются списки полей и правила их преобразования. Конкретные реализации этой части выходят за рамки данного пособия.
   4.2. Если относится, то получить экземпляр (а при отсутствии создать) соответствующей части (формирования локальной части IRI для таких частей также прописано в правилах из п. 4.1.). Для полученного экземпляра задать отношение, связывающего его с рассматриваемой характеристикой.
   4.3. Если не относится, то найти отношение по имени характеристики (при отсутствии создать отношение с таким именем как дочернее свойство для свойства «unknown» или типизированного свойства). Связать экземпляр товара с рассматриваемым значением характеристики полученным отношением.

Здесь проявляется гибкость онтологического подхода. В отличии от реляционной модели, в любой момент легко могут быть добавлены новые отношения,

произведен перенос их в другую группу, произведено задание им новых ограничений. Также работа с описанием предметной области в целом ведется в терминах объектной или любой интересующей нас модели. При этом изменения, как правило, не ведут к долгому ожидания (т.к. чаще всего достаточно изменения нескольких троек).

Получив общее неформальное описание модели, можно приступать к ее формализации и реализации БЗ с использованием конкретного синтаксиса и/или инструментария. Эти процессы будут делаться параллельно с использованием системы Protégé.

Первоначально после запуска системы необходимо задать идентификатор онтологии на вкладке «Active ontology» в поле «Ontology IRI» (рис. 3б). Данный идентификатор также определяет и базовый префикс. Для изображенной онтологии IRI будет «http://myontologies.com/goods», а базовый префикс для всех создаваемых сущностей «http://myontologies.com/goods#»

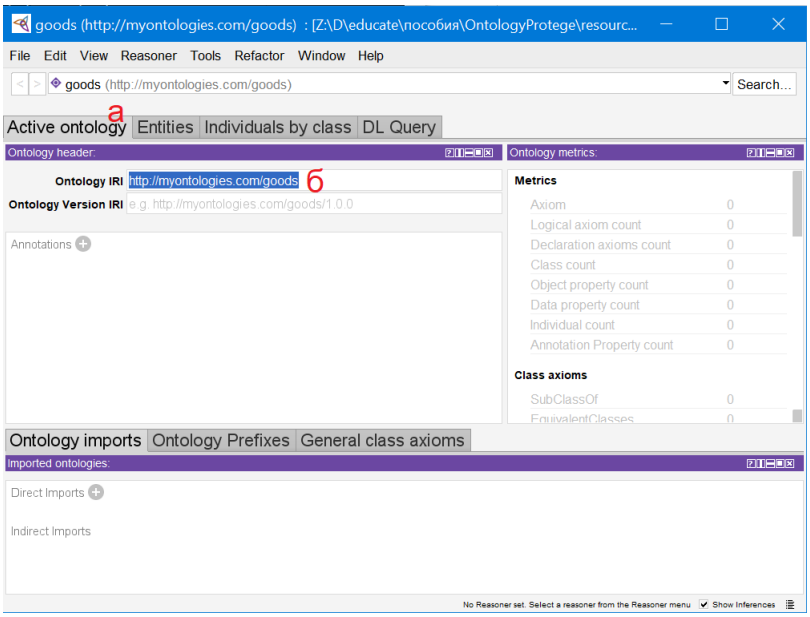

Рис. 4 Вкладка «Active ontology» в системе Protégé (а – панель в вкладок с активной вкладкой «Active ontology»; б – идентификатор онтологии)

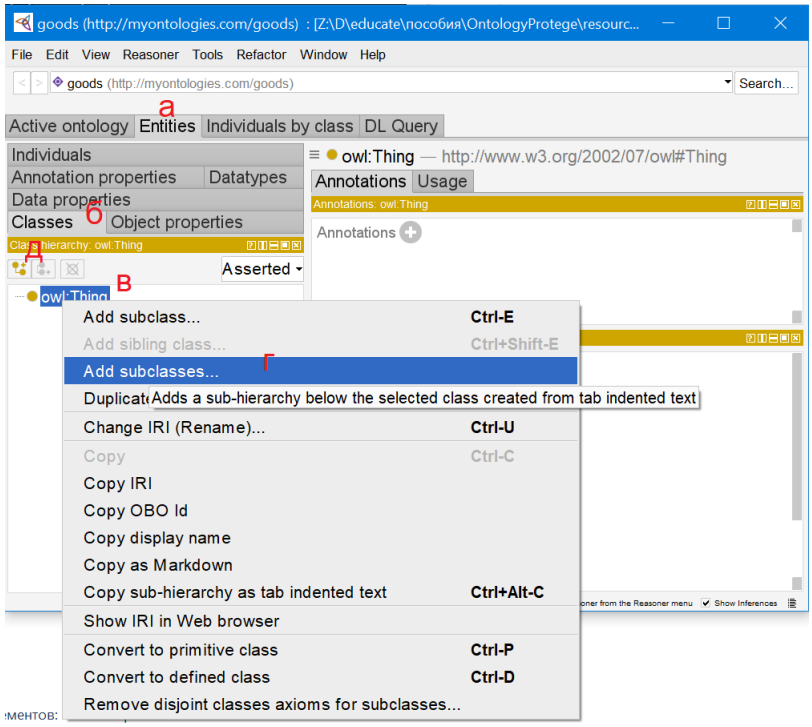

Рис. 5 Вкладка «Entities» в системе Protégé (а – панель в вкладок с активной вкладкой «Enitites»; б – панель вложенных вкладок вкладки «Entities» с активной вкладкой «Classes»; в – суперкласс owl:Thing; г, д – кнопки добавления иерархии подклассов/подкласса для выделенного класса)

Далее необходимо создать иерархию классов. Для этого необходимо выделить класс, подклассами которого будут верхние классы нашей иерархии (в нашем случае owl:Thing, рис. 5в). После этого вызывать контекстное меню (правой кнопкой мыши – ПКМ) и выбрать пункт «Add subclasses» (рис. 5г). Классы также могут добавляться по одному (рис. 5д).

При нажатии на кнопку «Add subclasses…» появляется окно добавления иерархии подклассов (рис. 6). Каждая введенная строка в данном окне создает отдельный класс (путем добавления в онтологию троек «x rdf:type owl:Class . x rdfs:subClassOf y»). Где x – IRI создаваемого класса, задаваемый как «{IRI онтологии}#{обработанное_имя_класса}» (в рамках обработки имени класса, происходит, например, замена пробела на «_»); y – IRI родительского класса, который задается как owl:Thing для классов, заданных без отступа и как IRI ближайшего верхнего класса с меньшим отступом для остальных (отступы задается как «Tab»).

После нажатия на «Continue» появляется окно, предлагающее создать для всех создаваемых классов одного уровня отношения «owl:disjointWith». Для этой задачи соглашаемся.

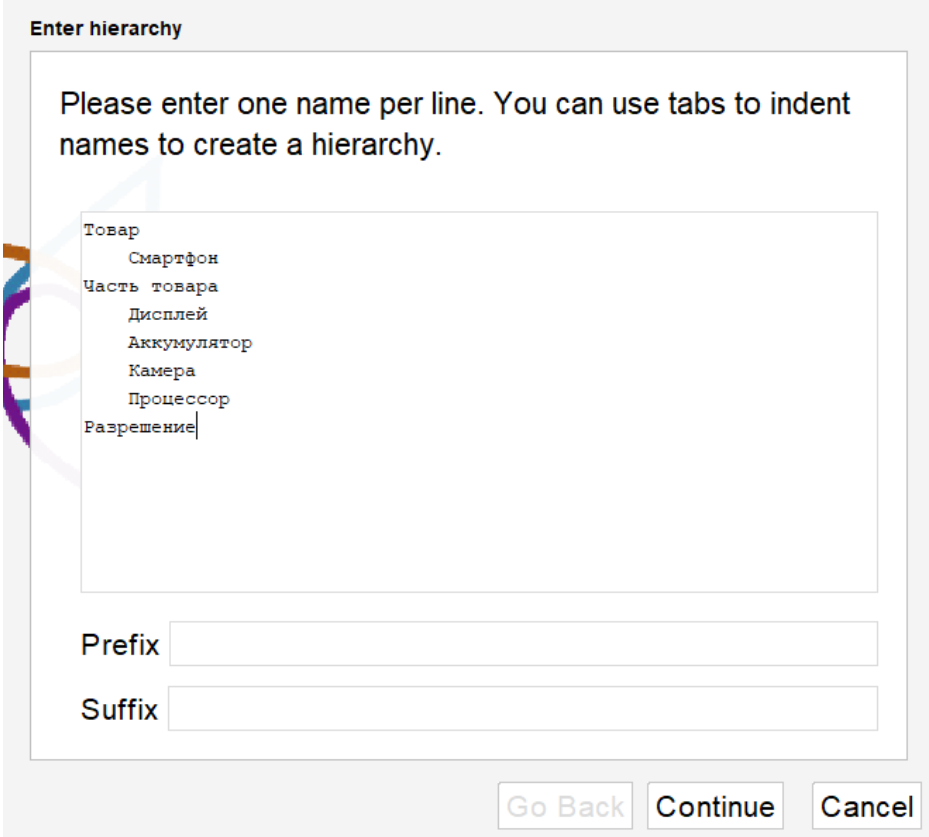

Рис. 6 Окно создания иерархии подклассов в системе Protégé

Пока БЗ является достаточно простой, рассмотрим ее графическое представление и один из синтаксисов OWL/RDF. Для вывода графического представления необходимо активировать вкладку «Window -> Tabs -> OntoGraf» и перейти на нее (рис. 7). Здесь будут видны классы, экземпляры и object property.

Рассмотрение текстового представления позволит нам увидеть какие именно тройки инструментальная система создала на данный момент и лучше понять как она работает (в некоторых случаях поведение системы может быть настроено, здесь рассматривается поведение по умолчанию). Для этого необходимо сохранить онтологию (File -> Save As). При сохранении на данном этапе необходимо выбрать синтаксис «Turtle Syntax».

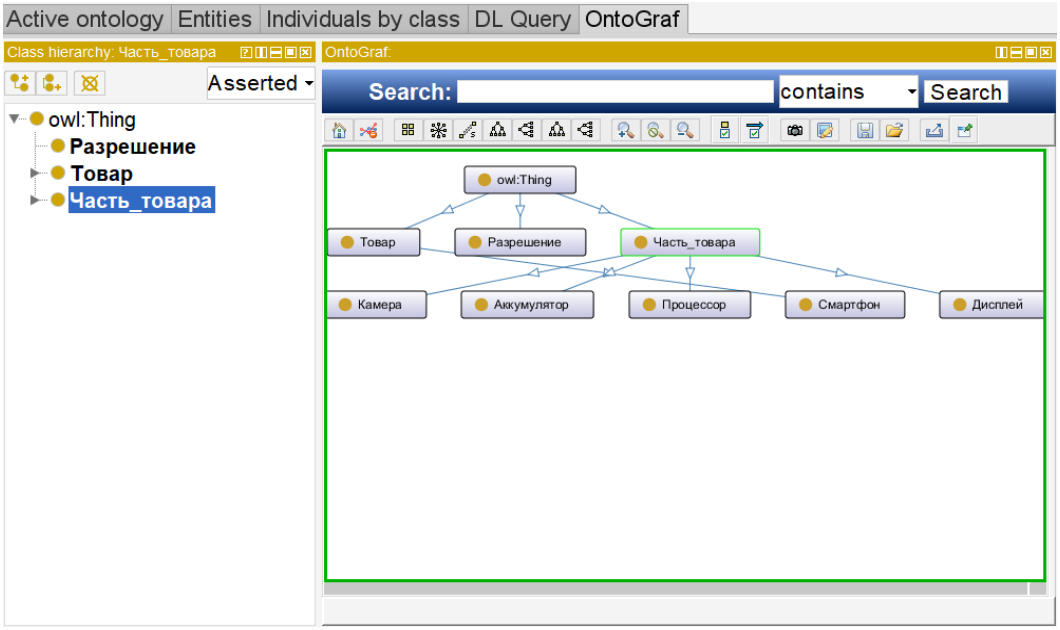

Рис. 7 Вкладка «OntoGraf» в системе Protégé

Открыв получившийся файл можно увидеть приведенный далее текст (приведены фрагменты с комментариями, объясняющими синтаксис и общую структуру онтологии. Комментарии начинаются с #).

#Описание начинается с объявления префиксов

#Определение общего префикса всех создаваемых сущностей (далее можно вместо него писать «:»

@prefix : <http://myontologies.com/goods#> .

#Определение префиксов owl, rdf, xml, xsd, rdfs

«.» является символом разделителем троек

@prefix owl: <http://www.w3.org/2002/07/owl#> .

...

#далее IRI нашей онтологии определяется как экземпляр онтологии

<http://myontologies.com/goods> rdf:type owl:Ontology .

#Далее следует описание созданных классов

#«;» является специальным разделителем троек и после него идет двойка «предикат – объект», а «субъект» берется из предыдущей тройки

:Аккумулятор rdf:type owl:Class ;

rdfs:subClassOf :Часть_товара .

#Т.е. это эквивалентно записи «:Аккумулятор rdf:type owl:Class . :Аккумулятор rdfs:subClassOf :Часть_товара»

#Для подклассов owl:Thing отношение rdfs:subClassOf может быть опущено при генерации файла

:Разрешение rdf:type owl:Class .

...

#Далее описываются общие аксиомы. В нашем случае это несколько условий на группы классов одного уровня, что никакие сущности не могут быть экземплярами одновременно нескольких классов из группы

[rdf:type owl:AllDisjointClasses ;
owl:members (:Аккумулятор :Дисплей :Камера :Процессор)] .

Преобразуем последнюю запись в вид троек для ее понимания, учитывая что: «[]» – позволяет задавать пустые узлы; «()» – позволяет задавать списки (для задания списков в RDF есть специальный класс rdf:List и отношения rdf:first, rdf:rest):

_x rdf:type owl:AllDisjointClasses .

_x owl:members _y .

_y rdf:type rdf:List . _y rdf:first :Аккумулятор . _y rdf:rest _e .

_e rdf:type rdf:List . _e rdf:first :Дисплей . _y rdf:rest _l .

_l rdf:type rdf:List . _l rdf:first :Камера . _y rdf:rest _k .

_k rdf:type rdf:List . _k rdf:first :Процессор . _y rdf:rest rdf:nill .

Это также может быть записано в функционально стиле: DisjointClasses(:Аккумулятор :Дисплей :Камера :Процессор)

Понимание стандартов и синтаксиса облегчает дальнейшую работу с онтологией (в особенности работу с запросами и редактирование онтологии из программного кода). Кроме того, система Protégé реализует на уровне интерфейса не все операции (в частности, нет возможности явного создания пустых узлов, но при этом система в состоянии их понять в большинстве случаев). Также стоит отметить, что многие описания так или иначе придется писать самостоятельно в соответствии с определенным синтаксисом (defined classes, сложные ограничения на поля и т.д. реализуются в системе с использованием Manchester Syntax [7]).

Далее необходимо задать Object Properties. Они задаются аналогично иерархии классов на вкладке «Object Properties». (рис. 8)

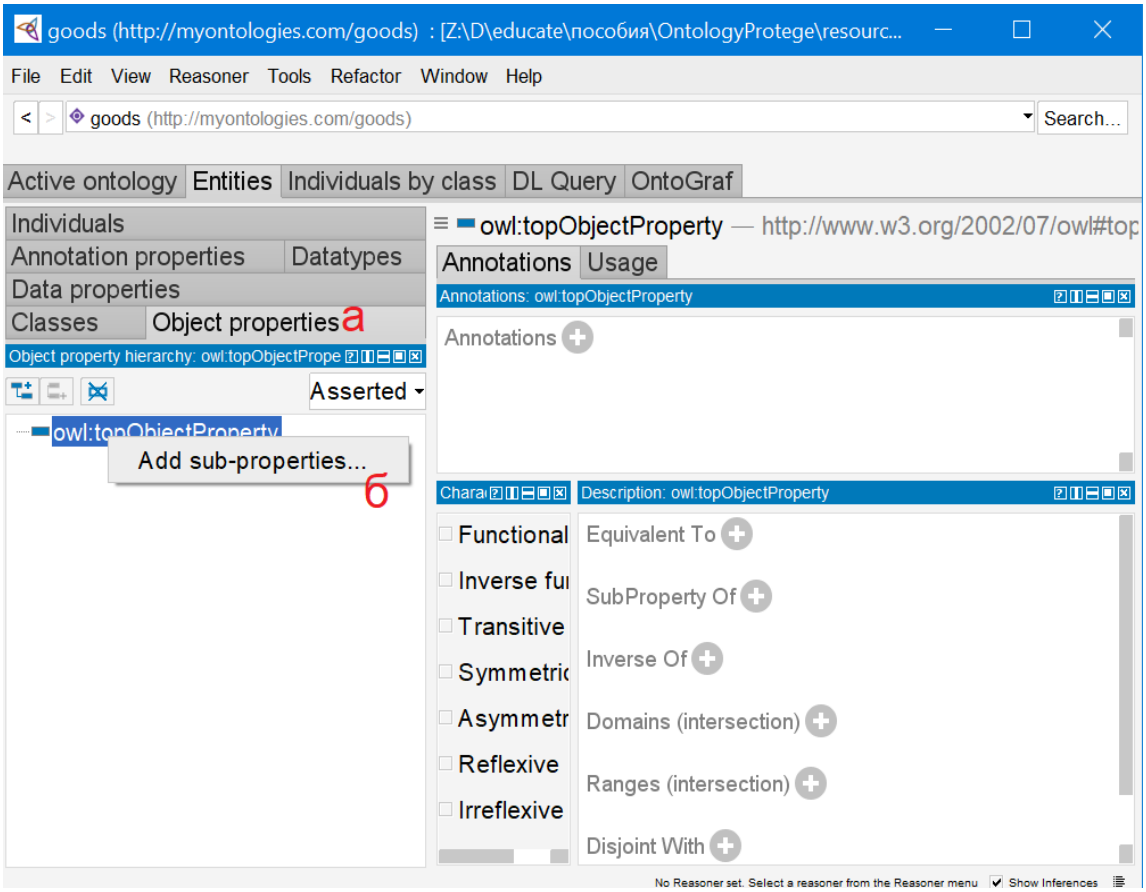

Рис. 8 Добавление Object Properties (а – активная вкладка Object Properties, б – кнопка добавления иерархии отношений)

На данном этапе необходимо задать свойства «часть» и «разрешение». Далее опишем эти свойства подробнее, а именно зададим область определения и область значения.

Для свойства «часть» область определения «Товар», область значения «Часть_товара». Для того, чтобы задать эту информацию, необходимо выбрать свойство (рис. 9а), найти раздел «Domains» в появившемся описании и добавить новую запись (рис. 9б), выбрав соответствующий класс сущностей (рис. 9г).

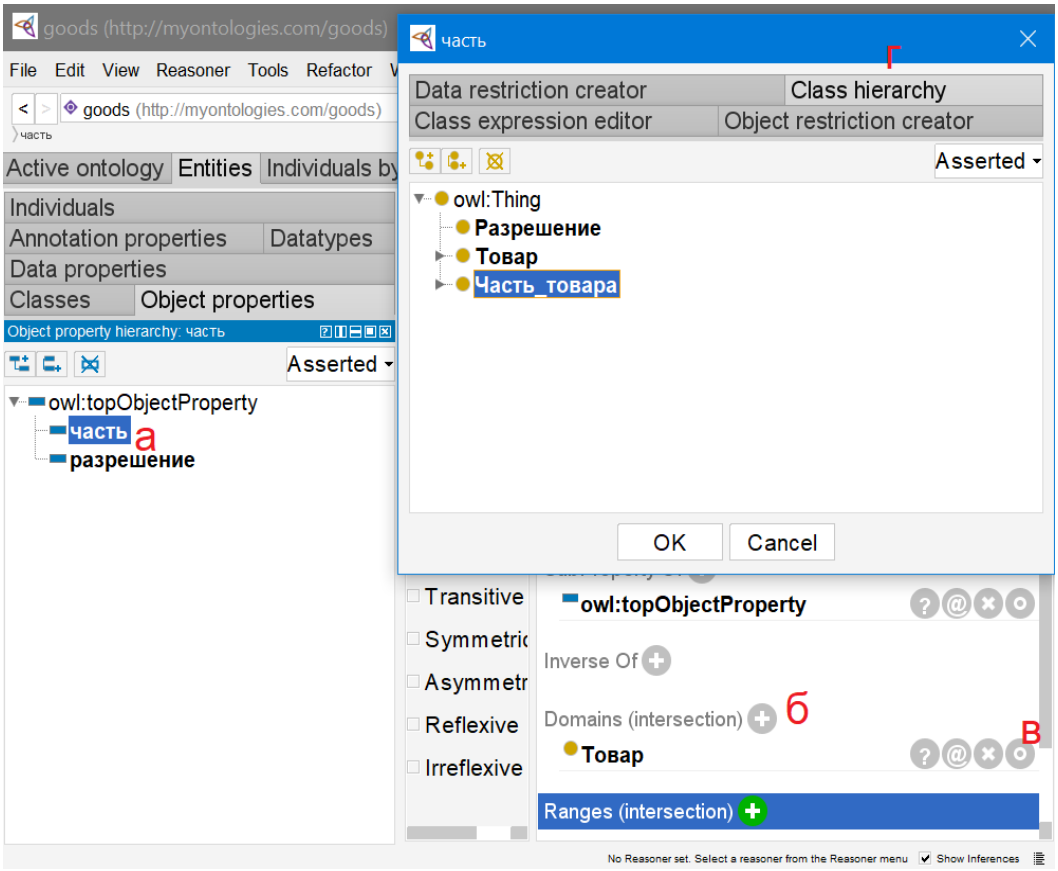

Рис. 9 Задание информации о Object Property в системе Protégé (а – выделенное свойство «часть», б – кнопка добавления новой области определения, в – кнопка редактирования существующей области определения, г – активная вкладка иерархии классов при задании области определения свойства)

При этом возможен не только выбор из существующих классов, но и задание через выражения на вкладках «Class expression editor», «Object restriction creator» (они являются одним из примеров, где придется использовать Manchester Syntax, а не просто графический интерфейс).

По аналогии создаются Data Properties. Полный список свойств, создаваемых в данной БЗ, приведен ниже (при создании необходимо заменить «;» на переносы строк и отступ)

текстовое

код производителя; цвет заявленный производителем; сеть 2g; сеть 3g; частота lte; формат sim карты; технология изготовления экрана; материал корпуса; защита корпуса; защита экрана; степень защиты; операционная система; производитель процессора; модель процессора; конфигурация процессора; графический ускоритель; тип вспышки основной камеры

числовое

год выпуска; число sim карт; диагональ экрана; разрешение горизонталь; разрешение вертикаль; разрешение кадр/сек; плотность пикселей; количество цветов экрана млн; количество ядер; частота процессора ггц; объем оперативной памяти гб; объем встроенной памяти гб; максимальный объем карты памяти гб; количество основных камер; количество мегапикселей основной камеры; апертура основной камеры; количество мегапикселей фронтальной камеры; количество фронтальных камер; емкость аккумулятора мА*ч; время работы в режиме разговора; время работы в режиме ожидания; ширина; высота; толщина; вес

логическое

автофокусировка основной камеры; оптическая стабилизация основной камеры; наличие вспышки фронтальной камеры; наличие беспроводной зарядки

свободное

конфигурация процессора; датчики; особенности основной камеры; комплектация; биометрическая защита; особенности

Для них также необходимо задать область определения и область значения. Стоит отметить, что для верхних свойств в иерархии будет задаваться только область значения, т.к. такие отношения как «числовое» могут связывать любой «субъект», но только с числом. Окно задание области значения для data properties отличается от аналогичного для object property (рис.10). Для свойств «числовое», «свободное», «текстовое», «логическое» область значения задается с использованием вкладки «Built in datatypes», на которой выбираются типы «xsd:double», «xsd:string», «xsd:boolean».

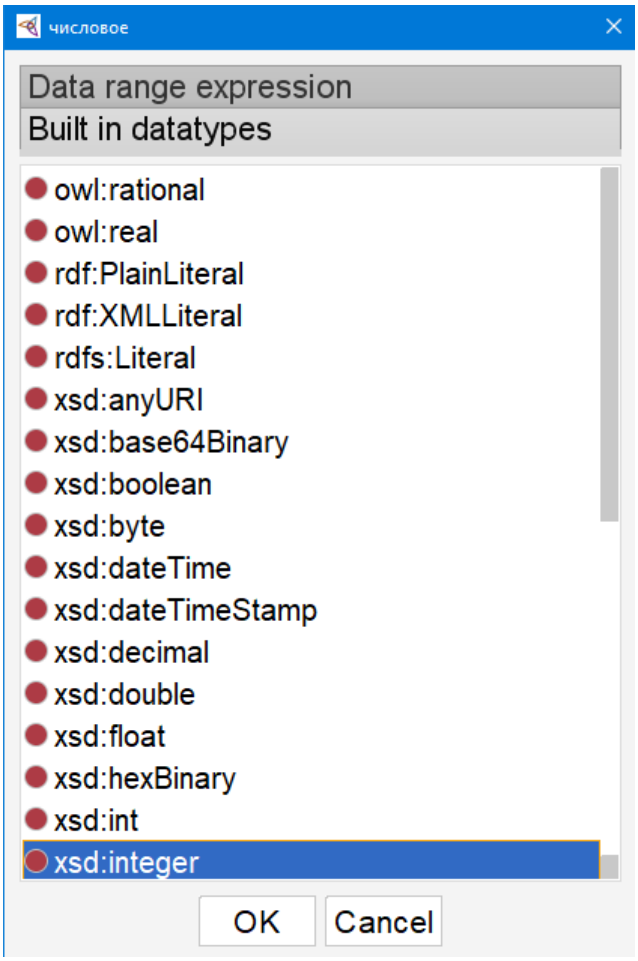

Рис. 10 Окно задания области значения для data property в системе Protégé

Для некоторых свойств необходимо задать дополнительные ограничения. Для этого используется вкладка «Data range expression» и Manchester Syntax. Разберем некоторые из свойств. Форматы сим карт стандартизированы: Полноразмерные (1FF), Mini-SIM (2FF), Micro-SIM (3FF), Nano-SIM (4FF), Встроенные (E-

SIM). Потому можно задать дополнительное ограничение на свойство «формат sim карты» (рис. 11).

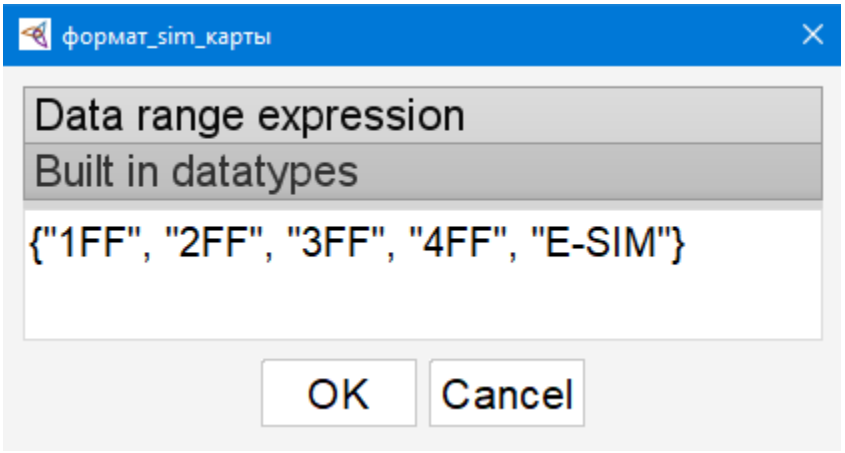

Рис. 11 Задание дополнительных ограничений на свойство «формат sim карты»

Рассмотрим, что при этом создается в файле онтологии:

:формат_sim_карты rdf:type owl:DatatypeProperty ;
rdfs:subPropertyOf :текстовое ;
rdfs:range [rdf:type rdfs:Datatype ; owl:oneOf ("1FF" "2FF" "3FF" "4FF" "E-SIM")] .

Как видно, для описания области значений создается пустой узел, который является типом данных (экземпляром rdfs:Datatype) и его экземпляры могут принимать значения из заданного списка (за счет использования отношения owl:oneOf). При использовании ограничений на data property пустой узел rdfs:Datatype создается всегда, дальнейшие конструкции зависят от введенного разработчиком. Рассмотрим возможные конструкции для этого поля подробнее:

1. {} – задает список значений, в рамках которых могут быть только скалярные значения (возможно использовать задание типа данных через «^^» и теги языков через «@»)
2. Любой тип данных (xsd:integer, xsd:string и т.д.)
3. Любое скалярное значение

4. [] – ограничение на значение, которые идут после типа значения. Например, xsd:interger[> 0] соответствует целым строго положительным числам. Ограничения прописываются в неограниченном количестве через запятую, например: xsd:integer[> 0 , < 100]. Ограничения, которые могут понадобиться в работах: все знаки сравнения (>, <, >=, <=), проверка строки по регулярному выражению (xsd:string[pattern “[0-9]{4}”]. Необходимо отметить, что регулярное выражение в таком виде проверяет соответствие всей строки, а не ищет подобные вхождения. Т.е. аналогично выражению «^[0-9]{4}$» в обычной ситуации).
5. Логические операторы: or, and. Например, «xsd:integer or xsd:double».

Рассмотрим свойство «год выпуска». Он должен быть целым четырехзначным числом не меньшим 1973 или целым не более чем двузначным числом (в случае, если учитываем возможность внесения сокращенной записи «20», «21», а не делаем преобразование заранее). Пользуясь новыми возможностями, изменим область значений свойства «числовое» на «xsd:integer or xsd:double or xsd:decimal». Далее зададим свойству «год выпуска» ограничение «xsd:integer[>1973] or xsd:integer[>=0 , <= 99]». Также рассмотрим, как это выражение преобразуется в RDF синтаксис (на примере Turtle):

```
:год_выпуска rdf:type owl:DatatypeProperty ;
  rdfs:subPropertyOf :числовое ;
  rdfs:range [ rdf:type rdfs:Datatype ;
    owl:unionOf ( [ rdf:type rdfs:Datatype ; owl:onDatatype xsd:integer ;
        owl:withRestrictions ( [ xsd:minInclusive 0 ] [ xsd:maxInclusive 99 ] ) ]
      [ rdf:type rdfs:Datatype ; owl:onDatatype xsd:integer ;
        owl:withRestrictions ( [ xsd:minExclusive 1973] ) ] )
  ] .
```

Приведенное выражение несколько сложнее Manchester Syntax, но позволяет получить понимание как работают выражения в терминах RDF и как они, соответственно, потом обрабатываются.

Создаются 2 пустых узла с отношениями xsd:minInclusive 0, xsd:maxInclusive 99, которые объединяются в RDF список (rdf:List через сокращенное описание «()»). Далее создается новый пустой узел – тип данных, основанный на xsd:integer (owl:onDatatype), которые соединяется отношением owl:withRestrictions с созданным ранее списком. Полностью аналогично создается еще один тип данных, основанный на integer, но с другими ограничениями. После этого создается еще один новый тип данных, которые определяется как объединение (owl:unionOf) созданных ранее. И именно он является областью определения нашего свойства. При этом по факту, созданные типы интересуют нас только с точки зрения возможных значений и могут рассматриваться как множества.

На остальные свойства ограничения могут быть наложены аналогично. Как минимум необходимо заполнить области определения и значения для свойств, связывающих части товара и конкретные характеристики («материал корпуса», «количество ядер» и т.д.). Остальное на данном этапе можно оставить как есть и переходить к экземплярам. Они создаются на вкладке «Individuals». (рис. 12)

На данной вкладке экземпляры создаются по одному. Для этого необходимо нажать на кнопку добавления (рис. 12б) и ввести локальное имя (на основании которого с использование IRI онтологии будет сгенерирован полный IRI), полный IRI или идентификатор с использованием префиксов. Создадим экземпляр рассматриваемого смартфона с идентификатором «SM-G965FZPHSER». После того как экземпляр создан, появляется окно его описания (рис. 13).

Нам необходимо задать экземпляром какого класса является созданная сущность (построить отношение rdf:type). Для этого необходимо добавить запись в раздел «Types» (рис 13а), где, использовав вкладку «Class Hierarchy», выбрать класс «Смартфон».

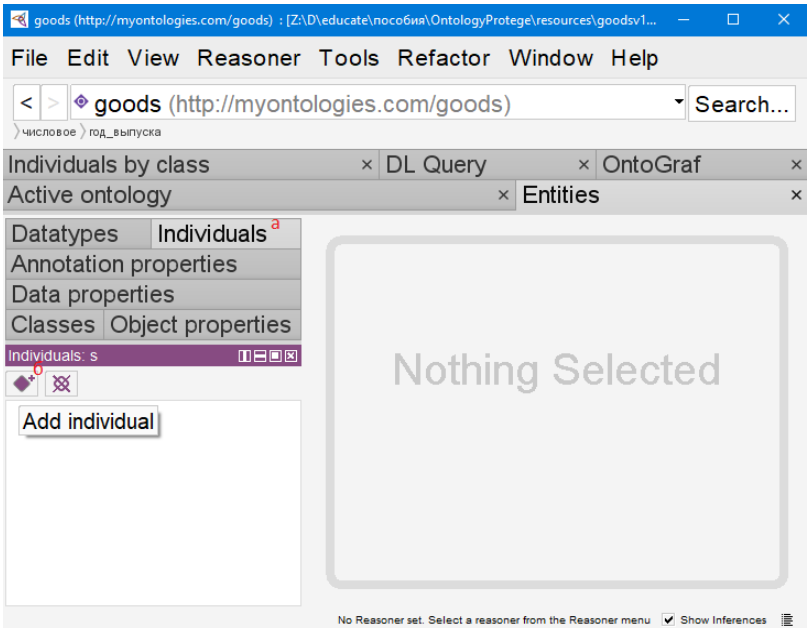

Рис. 12 Вкладка «Individuals» в системе Protégé (а – активная вкладка Individuals, б – кнопка создания нового экземпляра)

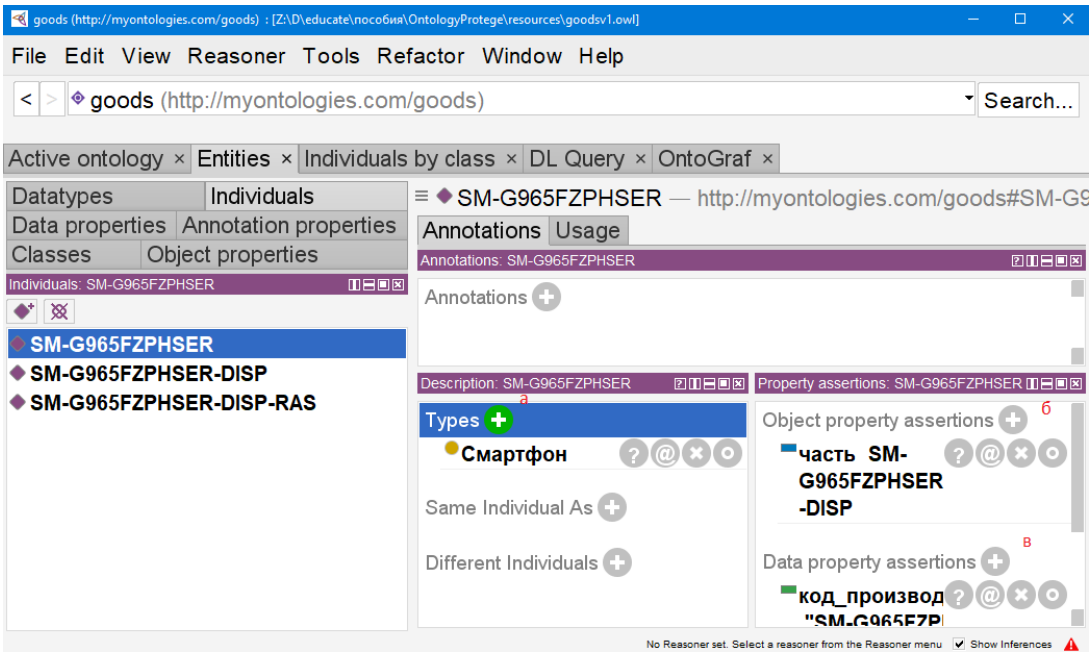

Рис. 13 Окно описания экземпляра в системе Protégé (а – задание отношения rdf:type, б – задание отношений с использование object property, в – задание отношений с использованием data property)

Далее необходимо задать отношения для рассматриваемого смартфона. Первым создадим отношение «часть» с его дисплеем. Для этого по аналогии создадим новый экземпляр класса «Дисплей». После чего для экземпляра смартфона добавим новую запись в раздел «Object property assertions» (рис. 13б). При добав-

лении необходимо ввести имя отношения и имя экземпляра, с которым строится отношение (рис. 14).

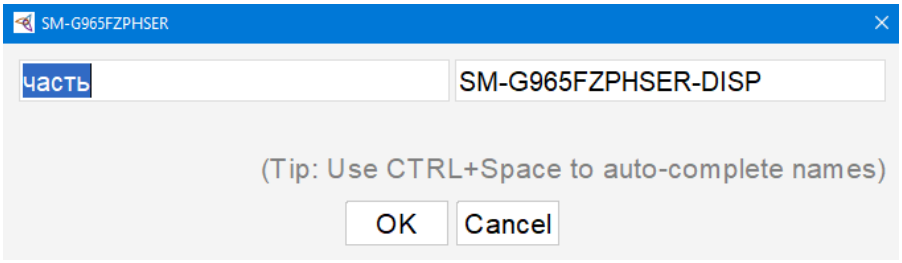

Рис. 14 Окно задания object property для экземпляра

После этого зададим отношение в разделе data properties на примере «код производителя» (рис. 15).

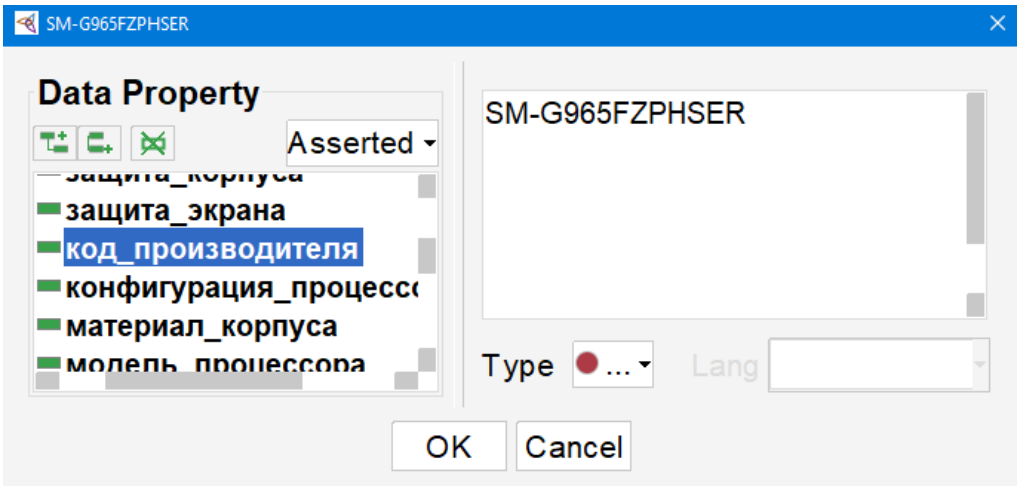

Рис. 15 Окно задания data property для экземпляра

Для этого в появившемся окне необходимо выбрать само свойство, ввести его значение и выбрать тип. Для определенных типов также активируется поле выбора языка.

По аналогии создаем экземпляр «Разрешение» и задаем дисплею разрешение. Рассмотрим текстовое представление созданных сущностей:

:SM-G965FZPHSER rdf:type owl:NamedIndividual , :Смартфон ;
:часть :SM-G965FZPHSER-DISP ;
:код_производителя "SM-G965FZPHSER"^^xsd:string .
:SM-G965FZPHSER-DISP rdf:type owl:NamedIndividual , :Дисплей ;

```
:разрешение :SM-G965FZPHSER-DISP-RAS .
:SM-G965FZPHSER-DISP-RAS rdf:type owl:NamedIndividual , :Разрешение ;
:разрешение_вертикаль 1920 ;
:разрешение_горизонталь 1080 .
```

Новая синтаксическая конструкция «,» означает, что после нее будет записан только «объект» из тройки, а «субъект – предикат» будут взяты из предыдущей. «owl:NamedIndividual» указывает, что сущность является экземпляром и не пустым узлом. Однако, на данном этапе, при удалении этого отношения, в системе Protégé не будет видна разница. Автор пособия считает, что дальнейшее описание достаточно большого количества экземпляров и их отношений проще сделать напрямую в файле онтологии с использованием любого текстового редактора. При этом можно было бы заменить некоторые экземпляры на пустые узлы:

```
:SM-G965FZPHSER-DISP rdf:type owl:NamedIndividual , :Дисплей ;
  :разрешение [rdf:type :Разрешение ; :разрешение_вертикаль 1920 ;
   :разрешение_горизонталь 1080] .
```

Система будет понимать такое описание, однако в интерфейсе не будет указываться сам экземпляр класса «Разрешение» и, как следствие, не будет возможности редактирования его описания. При этом все возможности обработки, в том числе вывод в рамках системы Protégé остаются возможными. Тем не менее, для большей наглядности при открытии онтологии в системе, далее будет приведен текст экземпляров и их отношений без использования таких пустых узлов.

```
:SM-G965FZPHSER rdf:type owl:NamedIndividual , :Смартфон ;
    :часть :SM-G965FZPHSER-DISP;
    :часть :Exynos_9810;
    :часть :SM-G965FZPHSER-CAM-BACK;
```

:часть :SM-G965FZPHSER-CAM-FRONT;
:часть :SM-G965FZPHSER-BAT;
:код_производителя "SM-G965FZPHSER";
:цвет_заявленный_производителем "ультрафиолет";
:сеть_2g "GSM 1900", "GSM 1800", "GSM 900";
:сеть_3g "WCDMA 900", "WCDMA 2100";
:частота_lte "LTE 1800 (B3)", "LTE 2100 (B1)", "LTE 2300 (B40)",
"LTE 2600 (B7)", "LTE 1900 (B39)", "LTE 2000 (B34)";
:формат_sim_карты "4FF";
:степень_защиты "IP68";
:операционная_система "Android 8.0 Oreo";
:год_выпуска 18;
:число_sim_карт 2;
:объем_оперативной_памяти_гб 6;
:объем_встроенной_памяти_гб 256;
:максимальный_объем_карты_памяти_гб 400;
:материал_корпуса "металл", "стекло";
:защита_корпуса "защита от пыли и влаги";
:ширина 73.8;
:высота 158.1;
:толщина 8.5;
:вес 189;
:датчики "гироскоп, датчик Холла, акселерометр, барометр, компас, датчик приближения, датчик освещения, датчик сердечного ритма";
:комплектация "комплект амбушюр, комплект переходников, документация, сетевой адаптер, скрепка для извлечения слота SIM-карты, наушники";
:биометрическая_защита "сканер лица, сканер радужной оболочки глаза, сканер отпечатков пальцев";
:особенности "ANT+, технология Dolby Atmos, камера Bixby, камера-переводчик, селфимоджи, стереодинамики от AKG, несъемный аккумулятор" .

:SM-G965FZPHSER-DISP rdf:type owl:NamedIndividual , :Дисплей ;
:разрешение :SM-G965FZPHSER-DISP-RAS;
:технология_изготовления_экрана "Super AMOLED";
:защита_экрана "Corning Gorilla Glass";
:диагональ_экрана 6.2;
:плотность_пикселей 529;
:количество_цветов_экрана_млн 16 .
:SM-G965FZPHSER-DISP-RAS rdf:type owl:NamedIndividual ;
:разрешение_вертикаль 1920 ;
:разрешение_горизонталь 1080 .
:Exynos_9810 rdf:type owl:NamedIndividual , :Процессор ;

:производитель_процессора "Samsung" ;
:модель_процессора "Exynos 9810" ;
:конфигурация_процессора "4x Custom 2.9 ГГц, 4x Cortex-A55 1.9 ГГц" ;
:количество_ядер 8 ;
:частота_процессора_ггц 2.9, 1.9 ;
:графический_ускоритель "Mali-G72 MP18" .
:SM-G965FZPHSER-CAM-BACK rdf:type owl:NamedIndividual , :Камера ;
:количество_основных_камер 2 ;
:количество_мегапикселей_основной_камеры 12, 12 ;
:апертура_основной_камеры 1.5, 2.4 ;
:тип_вспышки_основной_камеры "светодиодная" ;
:разрешение :SM-G965FZPHSER-DISP-CAM-BACK-RAS-1, :SM-G965FZPHSER-DISP-CAM-BACK-RAS-2, :SM-G965FZPHSER-DISP-CAM-BACK-RAS-3 ;
:автофокусировка_основной_камеры 1 ;
:оптическая_стабилизация_основной_камеры 1 ;
:особенности_основной_камеры "сверхбыстрый датчик изображения, двойная апертура объектива" .
:SM-G965FZPHSER-DISP-CAM-BACK-RAS-1 rdf:type owl:NamedIndividual ;
:разрешение_вертикаль 3840 ;
:разрешение_горизонталь 2160 ;
#полный IRI необходим из-за спец символа "/"
<http://myontologies.com/goods#разрешение_кадр/сек> 60 .
:SM-G965FZPHSER-DISP-CAM-BACK-RAS-2 rdf:type owl:NamedIndividual ;
:разрешение_вертикаль 1280 ;
:разрешение_горизонталь 720 ;
#полный IRI необходим из-за спец символа "/"
<http://myontologies.com/goods#разрешение_кадр/сек> 960 .
:SM-G965FZPHSER-DISP-CAM-BACK-RAS-3 rdf:type owl:NamedIndividual ;
:разрешение_вертикаль 1920 ;
:разрешение_горизонталь 1080 ;
#полный IRI необходим из-за спец символа "/"
<http://myontologies.com/goods#разрешение_кадр/сек> 240 .
:SM-G965FZPHSER-CAM-FRONT rdf:type owl:NamedIndividual , :Камера ;
:количество_мегапикселей_фронтальной_камеры 8;
:количество_фронтальных_камер 1 ;
:разрешение :SM-G965FZPHSER-DISP-CAM-FRONT-RAS-1 ;
:наличие_вспышки_фронтальной_камеры 0 .
:SM-G965FZPHSER-DISP-CAM-FRONT-RAS-1 rdf:type owl:NamedIndividual , :Разрешение ;
:разрешение_вертикаль 2560 ;
:разрешение_горизонталь 1440 ;

#полный IRI необходим из-за спец символа "/"
<http://myontologies.com/goods#разрешение_кадр/сек> 30 .
:SM-G965FZPHSER-BAT rdf:type owl:NamedIndividual , :Аккумулятор ;
#полный IRI необходим из-за спец символа "*"
<http://myontologies.com/goods#емкость_аккумулятора_мА*ч> 3500 ;
:время_работы_в_режиме_разговора 25 ;
:наличие_беспроводной_зарядки 1 .

Теперь основные данные внесены и можно проверить онтологию на непротиворечивость. Для этого необходимо выбрать механизм вывода HermiT Reasoner [5] (рис. 16б) и запустить его (рис. 16в).

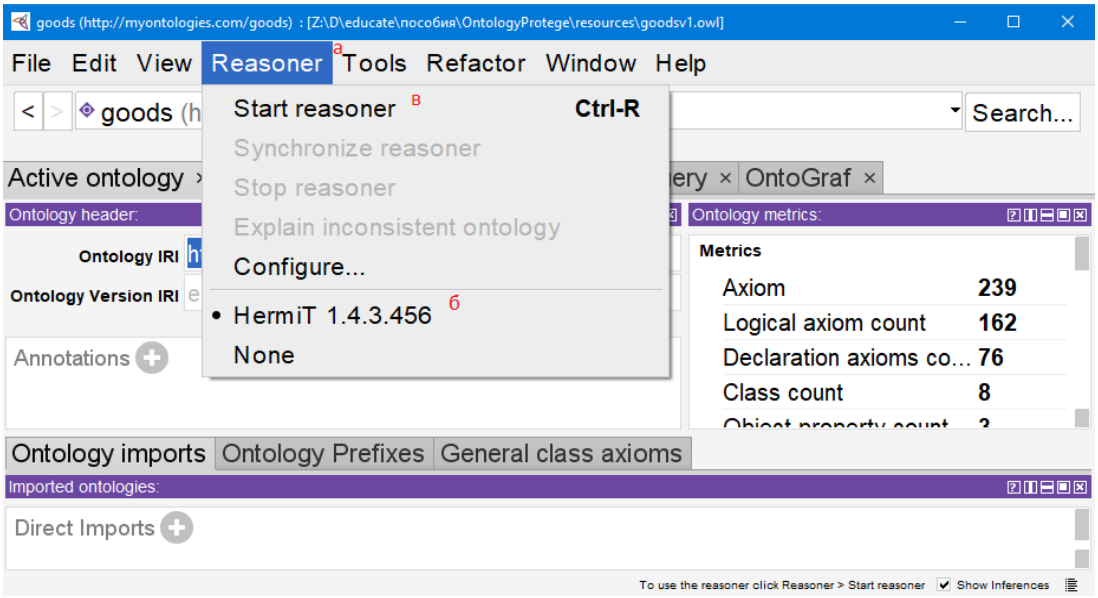

Рис. 16 Запуск механизма вывода в системе Protégé (а – пункт меню с действиями вывода, б – выбранный механизм HermiT, в – кнопка запуска вывода).

Индикацией того, что вывод запущен в данном случае будет то, что кнопка «Start reasoner» станет неактивной, а «Stop reasoner» наоборот активной. Если в процессе вывода были найдены противоречия, то система выведет сообщения об ошибках с их объяснением (рис. 17а) или выведет сообщение об ошибке запуска вывода (рис. 17в). При выполнении всех приведенных выше действий и рекомендаций ошибок быть не должно.

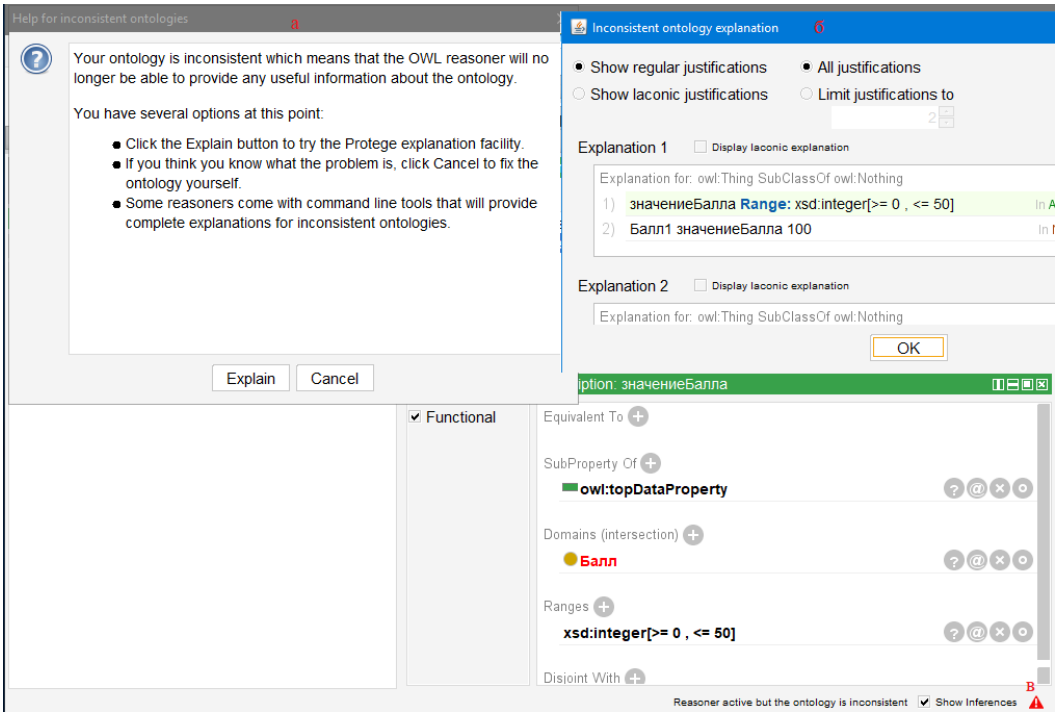

Рис. 17 Пример ошибки при запуске механизма рассуждений (а – сообщение о проблемах в онтологии с предложением объяснения; б – окно объяснений; в – индикатор ошибок в логах системы)

Также, если внимательно изучить приведенный выше фрагмент с экземплярами онтологии, то мы намеренно не задали тип для таких сущностей как «G965FZPHSER-DISP-CAM-BACK-RAS-2». Если при этом читатель выполнял все рекомендации и заполнил область определения для таких отношений как «разрешение_горизонталь», то можно увидеть, что тип был выведен автоматически (рис. 18). Выведенные (Inferred) понятия отмечаются в системе Protégé желтым цветом.

Для удобства дальнейшей обработки дополним онтологию новым object property: «является_частью». Это свойство является обратным к «часть», что необходимо отметить в его описании (рис. 19а). Прописывать его к созданным сущностям нет необходимости, т.к. оно будет выведено (рис. 19б).

В процессе этой работы было создана первая версия онтологии, декларативно описывающей предметную область «товар смартфон». В следующих работах в онтологию будут добавлены новые аксиомы, выражения и правила для вывода от-

ношений и проверки непротиворечивости БЗ, а также написаны запросы для выбора информации из нее по определенным критериям.

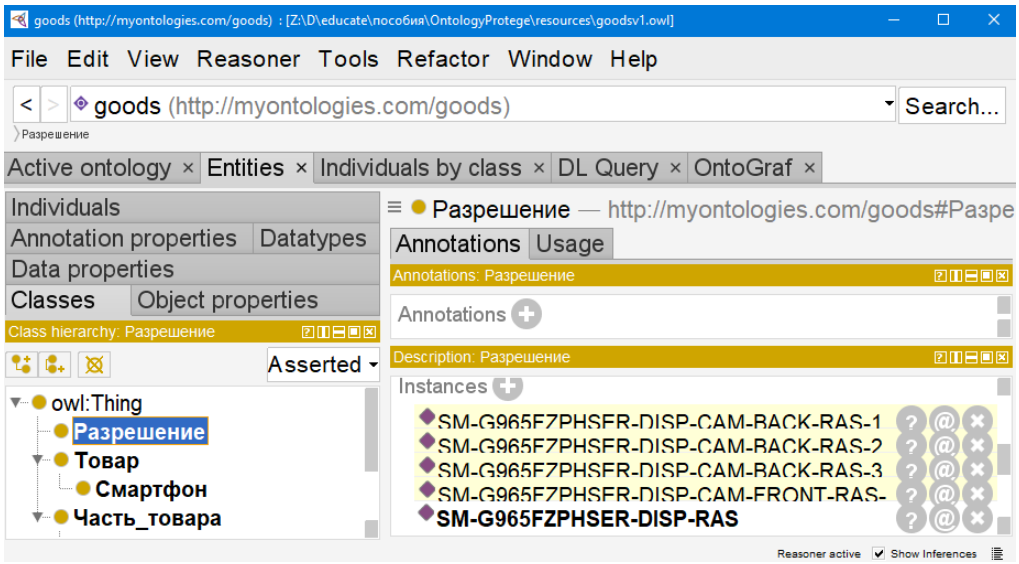

Рис. 18 Выведенные экземпляры у класса «Разрешение» в системе Protégé

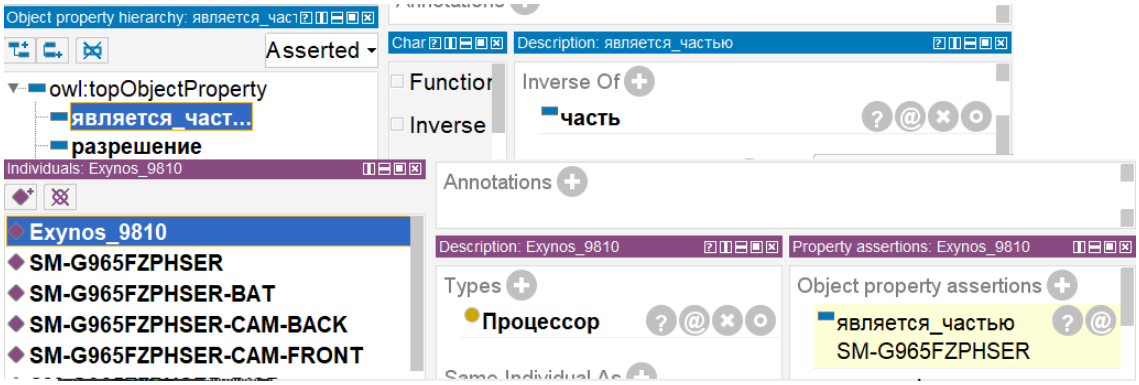

Рис. 19 Демонстрация вывода в соответствии с описанием inverseOf

Задания

1. Дополнить созданную базу знаний минимум 10 экземплярами других смартфонов
2. По аналогии, дополнить базу знаний для категорий «ноутбук» и «холодильник»

ЛАБОРАТОРНАЯ РАБОТА 2. СОЗДАНИЕ DEFINED CLASSES В СИСТЕМЕ PROTÉGÉ

В этой работе будут созданы классы, отношения «rdf:type» с которыми не будут указываться в явном виде, а будут строиться автоматически в процессе вывода. Такие классы в системе Protégé называются «Defined Classes» («обычные» классы называются «Primitive Classes»). Для примера будут реализованы следующие классы: «Новый смартфон», «Камерофон», «Игровой смартфон», «Смартфон для фильмов», «Смартфон для походов». Это позволит образовать в онтологии группы экземпляров в соответствии с заранее заданными критериями, которые будут доступны для дальнейшей обработки.

Первоначально сформируем критерии, по которым экземпляры будут относится к заявленным классам:

1. Новый смартфон. Выпущен после 18 года.
2. Камерофон. Минимум 2 основные камеры. Минимум 12 мегапикселей. Есть оптическая стабилизация. Съемка видео минимум Ultra HD 4K 60кадр./сек.
3. Игровой смартфон. Процессор Snapdragon или Exynos. Минимум 6 ядер. Частота работы процессора минимум 2.9ГГц. Объем оперативной памяти минимум 6гб. Графический ускоритель Mali или Adreno.
4. Смартфон для фильмов. Игровой смартфон. Соотношение сторон 21:9. Разрешение экрана от 4K.
5. Смартфон для походов. Емкость аккумулятора от 6000 мА*ч. Степень защиты выше чем IP68. Защита экрана «Gorilla Glass». Защита корпуса «укрепленный корпус». Материал корпуса включает металл.

Далее отладим запросы, в соответствии с которым будут формироваться экземпляры обозначенных классов. Такие запросы формируется с использованием

Manchester Syntax и называются в системе Protégé «DL Query». Для отладки таких запросов выделена отдельная вкладка (рис. 20).

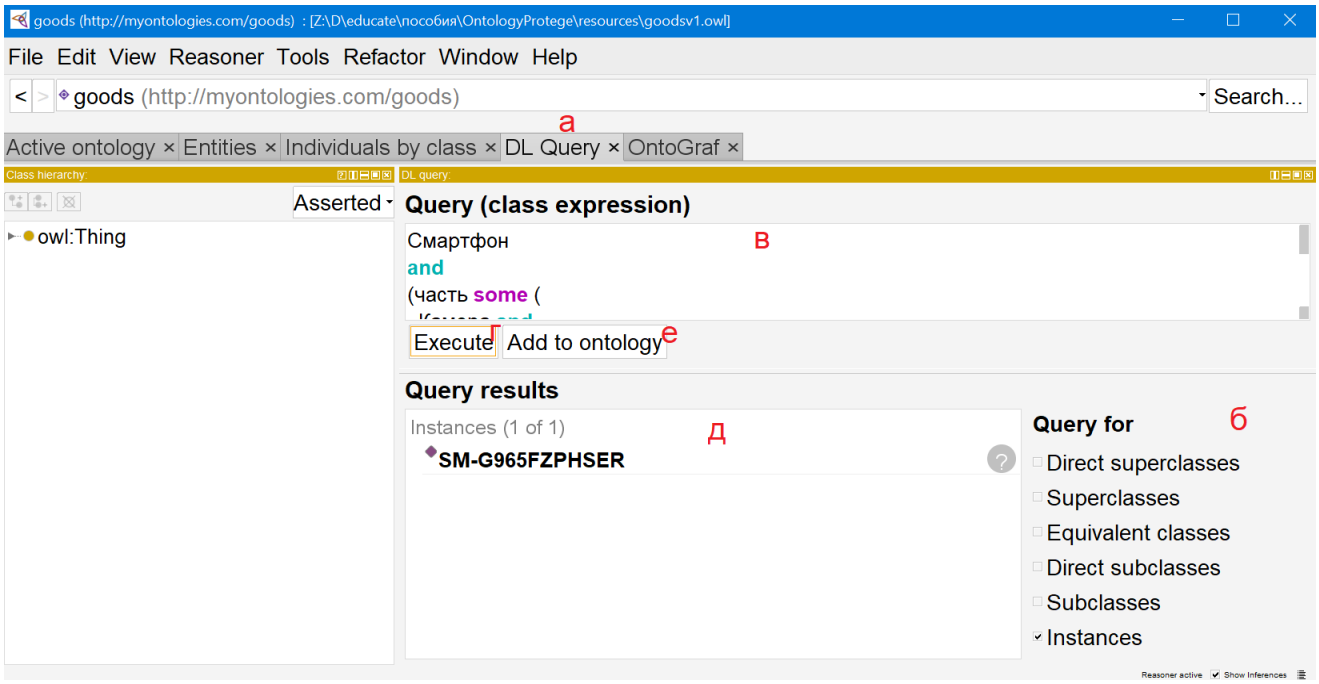

Рис. 20 Вкладка «DLQuery» в системе Protégé (а – панель вкладок с активной вкладкой «DLQuery»; б – панель выбора типа запрашиваемых сущностей; в – поле ввода запроса; г – кнопка выполнения запроса; д – результаты запроса; е – кнопка добавления запроса в онтологию)

Для освоения интерфейса выберем все экземпляры класса «Смартфон». Для этого необходимо в поле ввода запроса (рис. 20в) ввести запрос «Смартфон», выбрать ожидаемый тип результата – экземпляры (рис. 20б), запустить запрос (рис. 20г). После этого можно увидеть результаты (рис. 20д).

Перед написанием запросов необходимо определиться с возможными для использования конструкциями (объяснения будут приводится применительно к типу выбираемых сущностей – экземпляр):

1. Локальное имя класса. Позволяет выбрать экземпляры данного класса. Как явно заданные, так и выведенные. Например: «Смартфон».
2. <локальное_имя_свойства> some <выражение>. Позволяет выбрать все экземпляры, для которых задано как минимум одно отношение <локальное_имя_свойства> с сущностью, подходящей под <выражение>. В каче-

стве выражения могут выступать: вложенный «DL Query»; любое вариант ограничений, рассмотренный ранее при задании области значения data property. Например: «год_выпуска some xsd:integer»

3. <локальное_имя_свойства> only <выражение>. Все экземпляры, для которых отсутствуют заданные отношения или они все удовлетворяют выражению. Например: «год_выпуска only xsd:integer[>= 18].
4. <локальное_имя_свойства> exactly <количество>. Все экземпляры, для которых задано указанное количество указанных отношений. Например: «год_выпуска exactly 1»
5. <локальное_имя_свойства> exactly <количество> <выражение>. Все экземпляры, для которых задано указанное количество указанных отношений, удовлетворяющих выражению. При этом может быть сколько угодно таких же отношений, не удовлетворяющих выражению. Например: «год_выпуска exactly 1 xsd:integer[>= 18]»
6. <локальное_имя_свойства> min/max <количество> [<выражение>]. Аналогично exactly, но должно быть не менее/не более, чем указанное количество указанных отношений.
7. <локальное_имя_свойства> value <литерал>. Все экземпляры, которые связаны указанным отношением с указанным скалярным значением.
8. <выражение1> and/or <выражение2>. Пересечение/объединение множеств выбранных экземпляров в рамках выражения1 и выражения2. Например: «Дисплей or Процессор»
9. not <выражение>. Можно рассматривать как операция дополнения над множеством экземпляров, выбранных в рамках выражения. Например: «not Смартфон»

Определившись с множеством возможных конструкций, составим запрос для первого класса «Новый смартфон». У экземпляров такого класса не должно быть отношений «год выпуска», связывающих их с числом меньше 18 (для упро-

щения в первом варианте запроса рассмотрим год как двузначное число). Но, при этом должно быть хоть одно такое отношение (если таких отношений нет, то недостаточно информации для принятия решения). Также экземпляр должен быть типа «Смартфон». Соответственно, необходимый нам запрос должен описываться как пересечение 3 множеств: «Смартфон and (год_выпуска min 1) and (год_выпуска only xsd:integer[>= 18])».

Выполнив этот запрос на вкладке «DL Query», можно увидеть, что результатов не найдено. Вывод основывается на теории дескриптивных логик, рассмотрение которой выходит за рамки данного пособия. Однако, для понимания вывода и дальнейших действий, рассмотрим причины пустого результата в упрощенном варианте.

В онтологии относительно года выпуска есть как минимум утверждения «:SM-G965FZPHSER rdf:type Смартфон; :год_выпуска 18». Для того, чтобы экземпляр был выбран в результатах запроса, необходимо, чтобы он относился к каждому из трех множеств. Учитывая приведенные утверждения, можно сделать вывод, что «:SM-G965FZPHSER» относится к классу «Смартфон» и «год_выпуска min 1». Но, нельзя сделать никакие выводы относительно «год_выпуска only xsd:integer[>= 18]». У нас информация о том, что существует такое x («:SM-G965FZPHSER :год_выпуска x»), которое удовлетворяет ограничениям (x >= 18), но из этого нельзя сделать вывод, что для любого x это ограничение всегда выполняется. При этом в результаты запроса «not (Смартфон and (год_выпуска min 1) and (год_выпуска only xsd:integer[>= 18]))» рассматриваемый экземпляр также не войдет. Т.е. для «год_выпуска only xsd:integer[>= 18]» применительно к данного экземпляру не был сделан вывод ни о ложности, ни о истинности высказывания. В этом (предположение об открытости мира) состоит еще одна существенная разница с базами данных.

Одним из вариантов решения сложившейся проблемы является внесение в онтологию новых сведений. Так, если бы для рассматриваемого экземпляра вы-

полнялось ограничение «:год_выпуска exactly 1» (для рассматриваемого экземпляра существует ровно одно отношение «год выпуска»), то с учетом существования подходящего под критерии отношения уже можно было бы сделать вывод о том, что существуют только такие отношения. Эти сведения могут быть добавлены в разделе «Types» экземпляра с использованием вкладки «Class expression editor» (рис. 21).

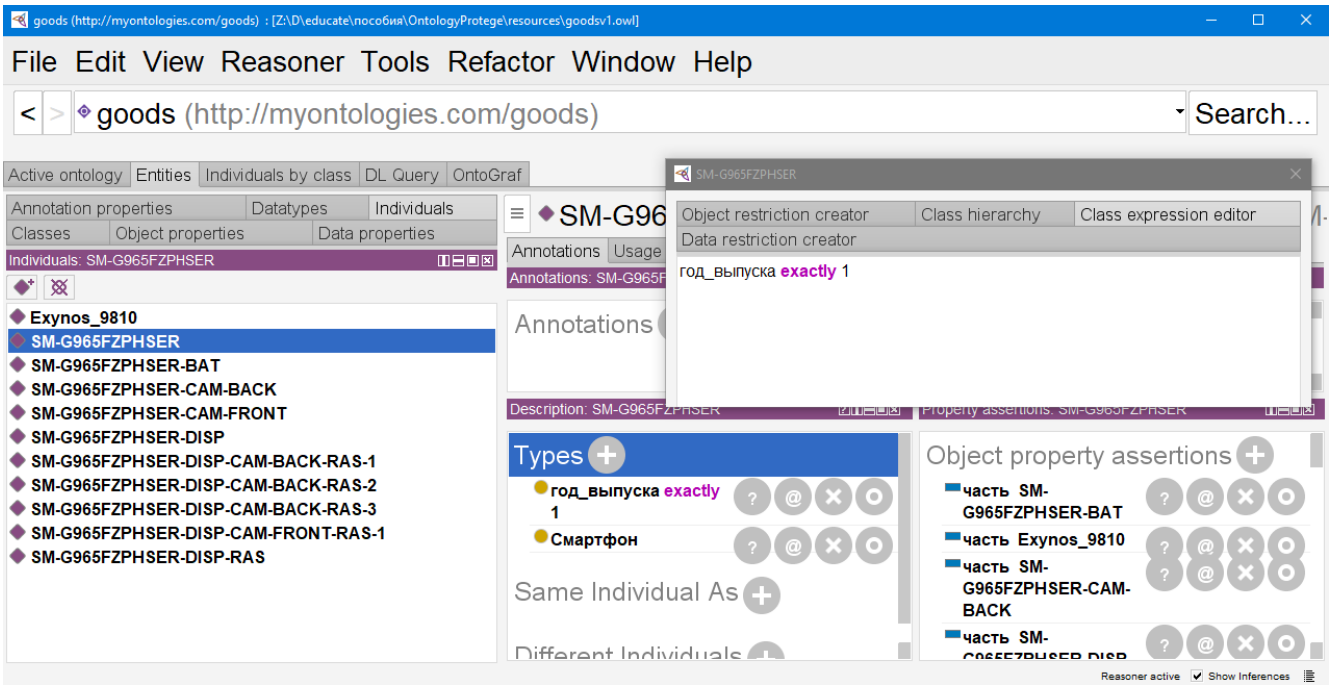

Рис. 21 Использование вкладки «Class expression editor» для добавления информации о экземпляре

После отладки запроса необходимо добавить класс. Это можно сделать с использованием кнопки «Add to ontlogy» на вкладке «DL Query» (рис. 20e) или создав класс и заполнив поле «Equivalent To» в его описании (рис. 22).

Рассмотрим, как данный класс описывается в тексте онтологии.

```
:Новый_смартфон rdf:type owl:Class ;
  owl:equivalentClass [ owl:intersectionOf ( :Смартфон
    [ rdf:type owl:Restriction ;
      owl:onProperty :год_выпуска ;
```

owl:allValuesFrom [rdf:type rdfs:Datatype ; owl:onDatatype xsd:integer ;
owl:withRestrictions ([xsd:minInclusive 18])]]
[rdf:type owl:Restriction ; owl:onProperty :год_выпуска ;
owl:minCardinality "1"^^xsd:nonNegativeInteger]); rdf:type owl:Class]

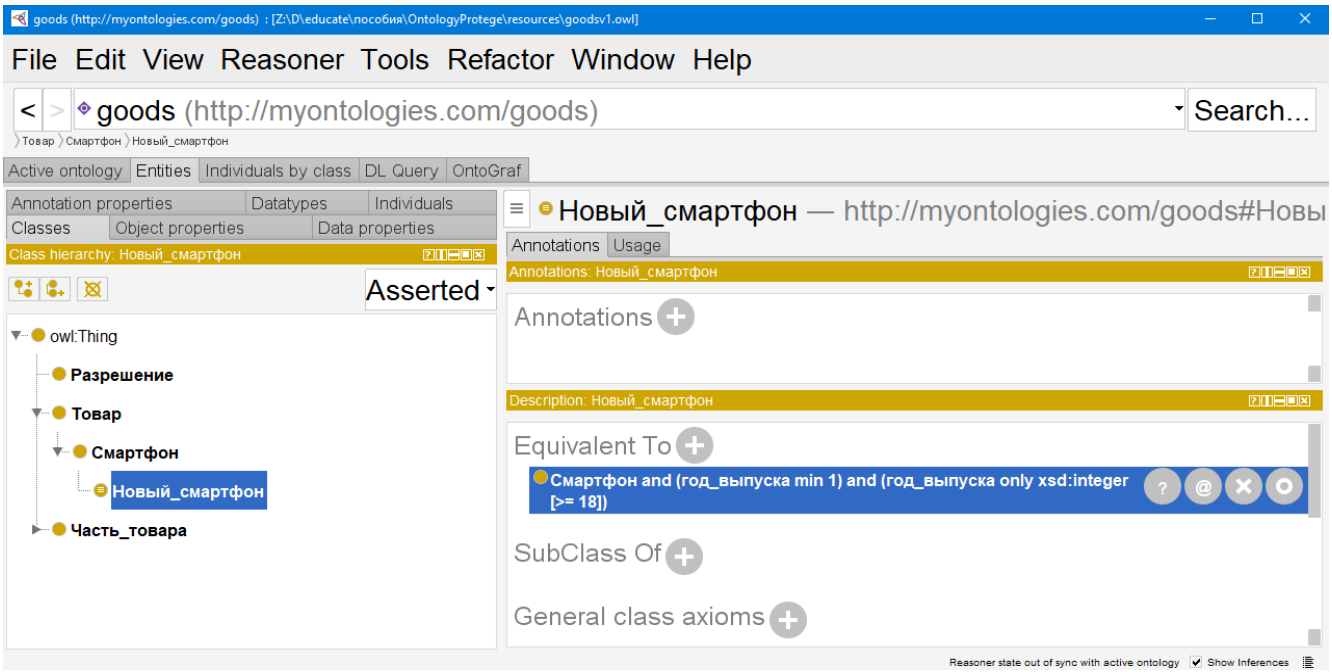

Рис. 22 Пример заполнения поля «Equivalent To» в описании класса

Как видно, используется отношение «owl:equivalentTo», которое позволяет сделать вывод, что если экземпляры относятся к «объекту» этого отношения, то относятся и к «субъекту». А в качестве «объекта» выступает пустой узел, являющийся пересечением 3 других классов, как и планировалось изначально.

Дополним написанный запрос для работы с годом, заданным в полном формате:

Смартфон and (год_выпуска min 1) and (год_выпуска only (xsd:integer[>= 18, <= 25] or xsd:integer[>= 2018]))

Также составим запрос для класса «Камерофон»:

```
Смартфон
and
(часть some ( Камера and
  (количество_основных_камер some xsd:integer[ >= 2 ])
  and
  (количество_мегапикселей_основной_камеры some xsd:integer[ >= 12 ])
  and
  (разрешение some (Разрешение and
    ((разрешение_горизонталь some xsd:integer[ >= 3840 ])
     or (разрешение_вертикаль some xsd:integer[ >= 3840 ]))
    and (разрешение_кадр/сек some xsd:integer[ >= 60 ])
  ))
))
```

Задания

1. Составить запросы для оставшихся групп: «Смартфон для фильмов», «Игровой смартфон», «Смартфон для походов». Создать defined class для каждой.
2. Сформировать минимум 5 групп (на основании имен подборок на ресурсе источнике) для категорий «Ноутбуки» и «Холодильники». Для каждой из групп сформировать критерии выбора экземпляров и создать defined class.

ЛАБОРАТОРНАЯ РАБОТА 3. НАПИСАНИЕ ЗАПРОСОВ НА ЯЗЫКЕ SPARQL

В предыдущей работе было произведено отнесение экземпляров к группам по определенным критериям. Для этого использовались defined classes и язык DL Query. Данные инструменты сильно расширяют возможности рассуждений в рамках онтологий. Однако, DL Query применяется именно в рамках рассуждений и имеет ряд недостатков, если использовать его только для получения доступа к явно заданным или уже выведенным сущностям онтологии: медленное выполнение (т.к. используется логический вывод); необходимость учитывать поведение при выводе (например, добавления в онтологию «exactly» в прошлой работы); невозможность описания некоторых условий (например, выбор отношений, а не экземпляров; выбор ряда характеристик смартфона). Поэтому для таких задач используется другой язык – SPARQL [8]. Данный язык позволяет выбирать и добавлять сущности в онтологию, формировать новый граф RDF. В данном пособии будут рассмотрены только возможности выбора. Язык синтаксически схож с SQL (и не только синтаксически. В частности, он также основан на предположении о закрытости мира, в отличие от DL Query), применяемым для аналогичных задач в базах данных. Поэтому (и потому что SQL должен быть уже знаком обучаемым) с ним будут проводиться аналогии.

Для работы с SPARQL в системе Protégé необходимо активировать вкладку «Window -> Tabs -> SPARQL Query» (рис. 23). Необходимо понимать, что SPARQL и механизм рассуждений в Protégé работают независимо и не влияют на результаты работы друг друга. Это означает, что нет возможности выбрать экземпляры класса «Новый смартфон», которые являются выведенными. При этом это возможно в рамках, например, библиотеки Apache Jena для языка Java.

Первоначально выберем первые 2 смартфона и характеристики их камер (количество камера; мегапиксели; апертура) и рассмотрим на примере этого запроса основные синтаксические конструкции.

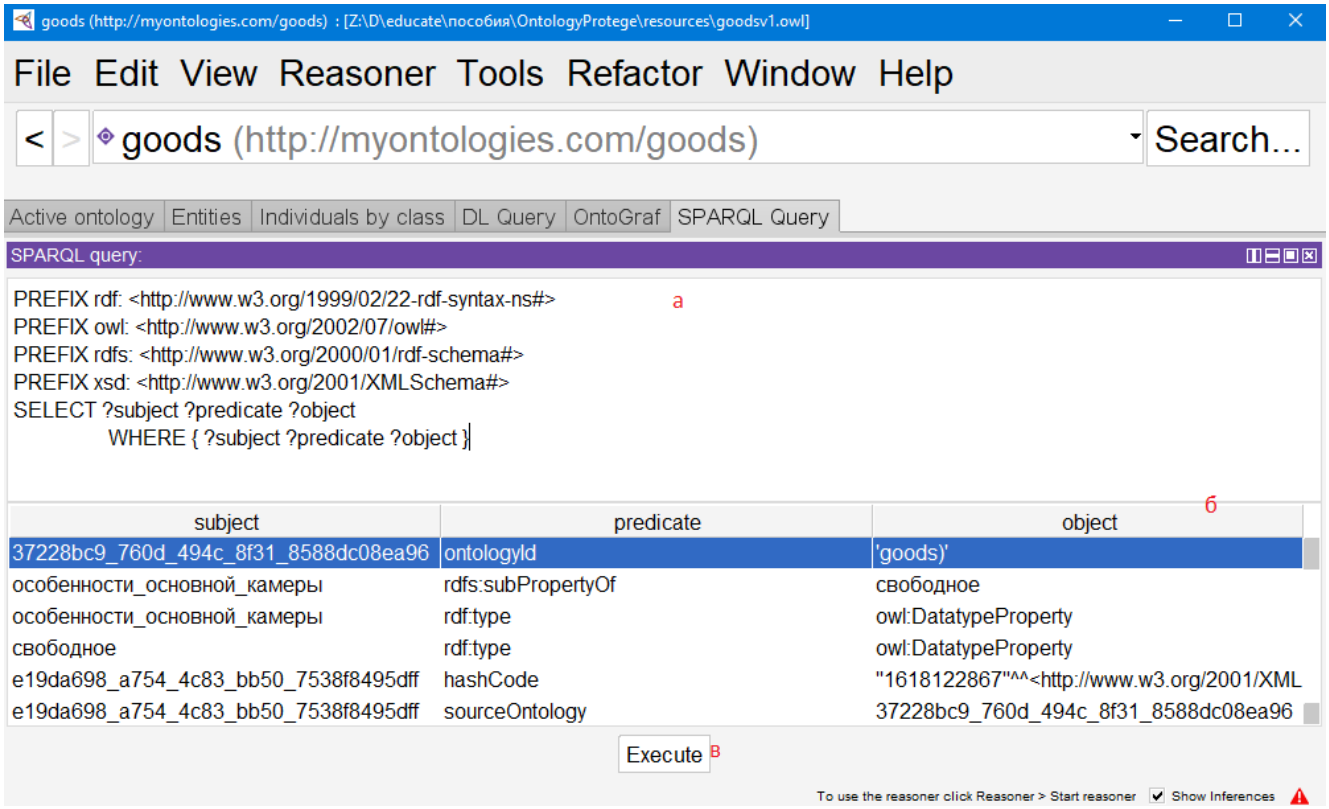

Рис. 23 Вкладка «SPARQL Query» в системе Protégé (а – поле ввода запроса; б – результаты запроса; в – кнопка выполнения запроса)

#Сначала приводится блок стандартных префиксов для того, чтобы не записывать громоздкие идентификаторы в рамках основной части запроса

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

#Также необходимо добавить префикс для созданной онтологии (ее идентификатор берется с вкладки «Active Ontology»)

PREFIX : <http://myontologies.com/goods#>

#После префиксов идет конструкция SELECT{переменные}, которая описывает что будет выбрано в качестве результата запроса

#Все переменные начинаются с «?» и перечисляются через пробел

SELECT ?смартфон ?камера_число ?камера_мп ?камера_ап

#Далее идет конструкция WHERE { {тройки} }. Тройки записываются аналогично синтаксису Turtle, что позволяет также использовать разделители «.», «;», «,».

WHERE {

#Вместо отдельных сущностей могут ставится переменные, начинающиеся с «?». На них могут накладываться условия, они могут выбираться в рамках списка полей в «SELECT».

?смартфон :часть ?камера .

?камера rdf:type :Камера ;

:количество_основных_камер ?камера_число ;

:количество_мегапикселей_основной_камеры ?камера_мп ;

:апертура_основной_камеры ?камера_ап }

#Далее идет конструкция ORDER BY, позволяющая задать сортировку результатов

ORDER BY DESC(?камера_ап)

#Далее идет конструкция LIMIT, OFFSET позволяющая задать количество пропускаемых записей и количество выбираемых

LIMIT 2 OFFSET 0

Как видно, структура запроса схожа с языком SQL (в том числе доступны конструкции GROUP BY, HAVING, объяснение которых выходят за рамки данного пособия), однако содержимое блоков различается. Основная часть – WHERE, разберем подробнее ее вложенные конструкции, которые будут использоваться в рамках работы:

1. Тройка «субъект предикат объект». На месте каждого из элементов тройки может стоять конкретный IRI или литерал или переменная. Позволяет выбрать все тройки онтологии, подходящие под заданную (заданная тройка используется как маска). При выборе: на месте переменных в заданной тройке может стоять любая сущность в выбираемой тройке; литералы должны совпадать по значению, типу, языку; IRI на соответствующих местах должны совпадать.
2. Разделители «.» между тройками можно интерпретировать как декартового произведение, но с учетом равенства соответствующих переменных. Например, «?смартфон :код_производителя ?код» позволяет выбрать все тройки, в которых присутствует отношение «код производителя». Если в онтологии 2 смартфона и только у одного из них есть это отношение, то будет выбрана 1 тройка.

   «?смартфон :часть ?часть» позволяет выбрать все тройки, в которых присутствует отношение «часть». Если в онтологии 2 смартфона и у каждого из них по 5 частей, то будет выбрано 10 троек.

   Если упрощенно рассмотреть порядок выбора в рамках «?смартфон :код_производителя ?код . ?смартфон :часть ?часть», то будет сначала выбрана 1 запись по первой тройке, 10 записей по второй. Далее будет сформировано 10 записей как все варианты совмещений данных записей. После чего будут удалены 5 записей, где в рамках одноименных переменных «смартфон» не совпадают значения.
3. В рамках троек также могут использоваться пустые узлы, описываемые аналогично синтаксису Turtle с использованием «[]».
4. OPTIONAL { {вложенная часть where} }. Позволяет задать вложенный запрос, который будет использован только при наличии результатов. Например, «?смартфон :код_производителя ?код . ?смартфон :часть ?часть» выберет 10 записей, если в онтологии будет 2 смартфона и 5 ча-

стей у каждого. Но, если будет 2 смартфона и 0 частей у каждого, то никаких результатов выбрано не будет. При этом «?смартфон :код_производителя ?код OPTIONAL { ?смартфон :часть ?часть }» выберет также 10 записей в первом случае, но 2 записи во втором (часть в рамках OPTIONAL учитываться не будет).

5. FILTER({условие}). Позволяет оставить в рамках выбранных записей только те, которые подходят под условие. Например, «?смартфон :год_выпуска ?год FILTER(?год > 18)» позволяет выбрать только те результаты, где год выпуска позднее 18 (стоит учитывать, что если у смартфона 2 таких отношения и у одного значение больше 18, а у другого меньше, то сам смартфон все равно будет выбран, т.к. одна из записей с ним будет подходить под условия). Т.к. FILTER не является тройкой, то после него нет необходимости использовать разделители. Таких конструкций в запросе может быть множество. В рамках условий FILTER могут использоваться операции: &&, ||, <, >, <=, >=, =, !=. Также могут использоваться встроенные функции, например, regex, isLiteral (список всех функций здесь приводиться не будет и может быть найден в документации).
6. FILTER EXISTS { { вложенная часть where} }, FILTER NOT EXISTS { { вложенная часть where} }. Специальные виды FILTER, которые оставляют записи, для которых вложенный запрос вернул/не вернул результаты.

Зная основные конструкции, составим следующий запрос: выбор информации (идентификатор, код производителя, конфигурация процессора, емкость аккумулятора, количество мегапикселей основной камеры, диагональ экрана) о всех смартфонах не старее 2018 года с «Super AMOLED» экраном.

Необходимо учитывать, что часть информации может отсутствовать, но при этом результат все равно должен быть представлен, значит необходимо использовать множество конструкций OPTIONAL. Условия на год и экран являются обяза-

тельными и сущности, не имеющие этой информации, не должны выбираться. Условие на экран накладывается через механизм троек. Условие на год должно учитывать, что могло быть переиздание смартфона и несколько таких отношений. Значит необходимо или работать с минимальным годом или использовать конструкцию FILTER NOT EXISTS.

```
SELECT ?смартфон ?код ?конфигурация ?диагональ ?емкость ?мп
WHERE {
  #с помощью пустого узла определяем условие, что выбираемый идентификатор должен быть "Смартфон" или его подкласс
  ?смартфон rdf:type [ rdfs:subClassOf* :Смартфон ] .
  #условие, что связь "год выпуска" должна существовать. Реализовано вложенным условием, чтобы в случае нескольких годов не выбралось несколько записей
  FILTER EXISTS {?смартфон :год_выпуска ?год}
  #условие на "новизну" года выпуска
  FILTER NOT EXISTS {?смартфон :год_выпуска ?год2 FILTER(?год2 < 18 || (?год2 > 1000 && ?год2 < 2018))}
  #считаем, что дисплей один. Для нескольких выберется информация только о подходящих
  ?смартфон :часть ?дисплей . ?дисплей rdf:type :Дисплей .
  #условие на технологию изготовления экрана
  ?дисплей :технология_изготовления_экрана ?технология
FILTER(regex(?технология, "super amoled", "i"))
  #дополнительная выбираемая информация (при наличии)
  OPTIONAL {?смартфон :код_производителя ?код}
  OPTIONAL {?смартфон :часть ?проц . ?проц rdf:type :Процессор . ?проц :конфигурация_процессора ?конфигурация}
```

OPTIONAL {?дисплей :диагональ_экрана ?диагональ}

OPTIONAL {?смартфон :часть ?аккум . ?аккум rdf:type :Аккумулятор . ?аккум <http://myontologies.com/goods#емкость_аккумулятора_мА*ч> ?емкость }

OPTIONAL {?смартфон :часть ?камера . ?камера rdf:type :Камера . ?камера :количество_мегапикселей_основной_камеры ?мп } }

Задания

Сформировать запросы:

1. выбор всех смартфонов и информации о них (аналогичной описываемой в работе) с аккумулятором не менее 500 мА*ч
2. выбор всех вариантов блоков камер смартфонов с их информацией (апертура, количество камер, производитель, мегапиксели, разрешение видеосъемки)
3. выбор всех вариантов модулей экранов с их информацией (диагональ, разрешение, плотность пикселей, технология изготовления экрана)
4. выбор первых 10 ноутбуков с их характеристиками: конфигурация процессора, тактовая частота, диагональ экрана, разрешение экрана, размер оперативной памяти, размер носителя информации, тип носителя информации, список разъемов

ЛАБОРАТОРНАЯ РАБОТА 4. НАПИСАНИЕ ПРАВИЛ С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА SWRL

В рамках последней работы необходимо сформировать в онтологии правила, которые позволят, учитывая особенности предметной области, формировать выведенные отношения. Для примера рассмотрим правило, которое будет отмечать сведения об аккумуляторе смартфона как сомнительные, если заявленное время работы в режиме разговора превышает «емкость аккумулятора / 250».

SWRL [9, 10] позволяет составлять правила, являющиеся импликацией между предпосылкой и следствием. После запуска механизма рассуждений онтология будет изменяться в соответствии с этими правилами. Предпосылками и следствиями являются выражения на основе предикатов (в том числе созданных в рамках онтологий). Рассмотрим синтаксис на примерах:

1. Смартфон(?x). Означает ?x rdf:type :Смартфон (?x аналогичен переменным в языке SPARQL)
2. часть(?x, ?p) – означает ?x :часть ?p. Таким образом можно записать предикат на основании любого созданного data или object property.
3. Смартфон(?x) ^ часть(?x, ?p). операция «and» записывается как «^».
4. Смартфон(?x) ^ часть(?x, ?p) -> является_частью(?p, ?x). Операция импликации записывается как «->». Данное правило определяет аксиому о том, что если ?x является смартфоном и содержит часть ?p, то также можно сказать, что ?p является частью ?x. Подобное правило генерируется автоматически в соответствии с полем «inverseOf», которое мы указывали в прошлых работах.

Из встроенных предикатов [9] будут необходимы: swrlb:lessThanOrEqual(?x, ?y) (?x меньше или равно ?y); swrlb:lessThan(?x, ?y) (?x строго меньше ?y); swrlb:greaterThanOrEqual(?x, ?y) (?x больше или равно ?y; swrlb:greaterThan(?x, ?y) (?x строго больше ?y); swrlb:divide(?x, ?y, ?z) (в ?x результат операции ?y/?z).

Для отладки условных частей правил может использоваться язык SQWRL [11]. В рамках него будет использоваться только предикат sqwrl:select(?x, ?y, ?z, …). Он позволяет отобразить пользователю все перечисленные сущности, для которых активировались предпосылки. Таким образом можно преобразовать правило в аналог запроса и проверить, что оно применяется для запланированных сущностей.

Сформируем SQWRL правило по заданным в работе критериям: «Аккумулятор(?a) ^ емкость_аккумулятор_мА_ч(?a, ?v) ^ время_работы_в_режиме_разговора(?a, ?t) ^ swrlb:divide(?tr, ?v, 250) ^ swrlb:greaterThan(?t, ?tr) -> sqwrl:select(?a, ?t, ?tr)» Именно такие записи о аккумуляторах будут отмечаться сомнительными. Чтобы проверить корректность условий данного правила следует рассмотреть для каких аккумуляторов это правило будет срабатывать (выбрать идентификатор аккумулятора, заявленное время работы и вычисленное время работы).

Для доступа к SQWRL необходимо активировать вкладку «Window -> Tabs -> SQWRL Tab» (рис. 24). Далее необходимо создать правило (рис. 24б) и ввести его тело (рис. 25в). Далее необходимо выделить правило и нажать кнопку запуска (рис. 24а, рис. 24в). После этого результат работы можно увидеть на вкладке, соответствующей имени правила (рис. 25г).

Убедившись в корректности условной части правила, можно перенести его в список SWRL правил (Window -> Tabs -> SWRL Tab) и изменить его заключение. На данной вкладке уже отображаются все SQWRL правила, однако они отмечены как неактивные. Можно или изменить правило и отметить его активным или создать новое. Условная часть правила остается прежней, а заключением будет являться «Сомнительно(?a)» (также необходимо создать соответствующий класс).

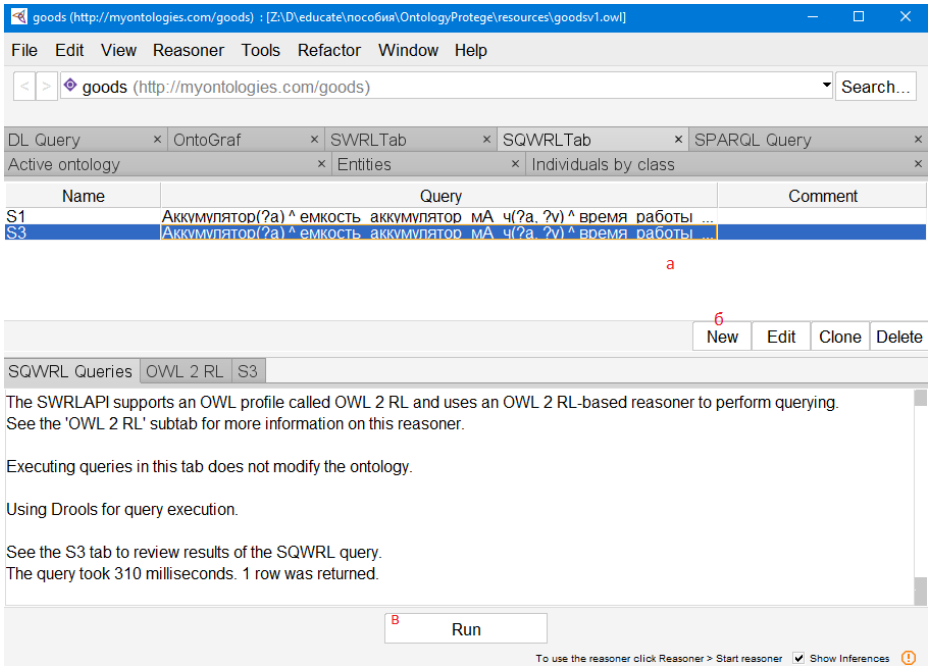

Рис. 24 Вкладка SQWRL в системе Protégé (а – список созданных правил; б – кнопки создания нового правила и редактирования выделенного; в – кнопка запуска выбранного правила)

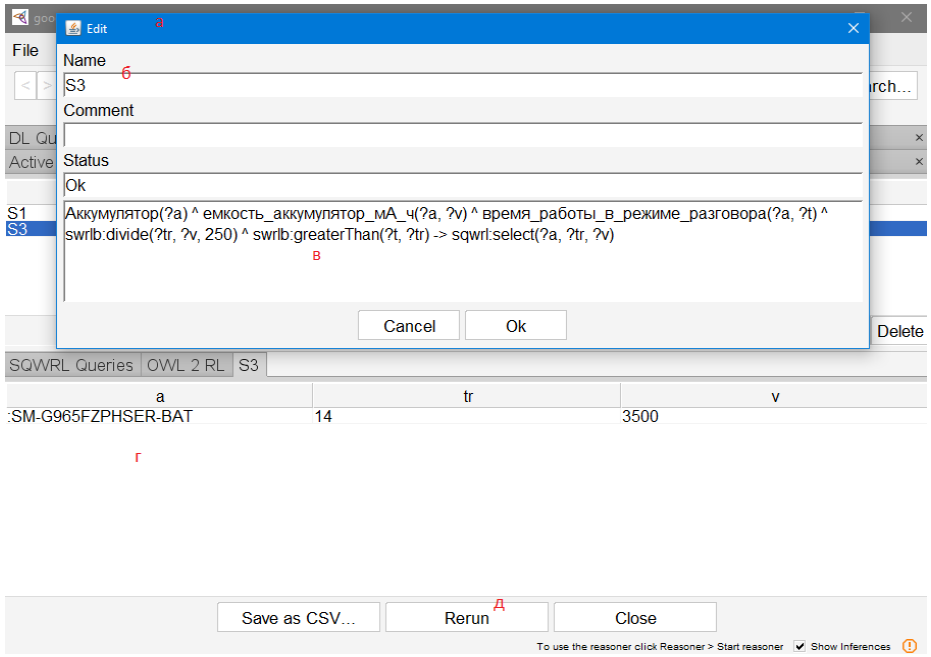

Рис. 25 вкладка SQWRL в системе Protégé (а – окно редактирования правила; б – имя правила; в – тело правила; г – результаты выполнения правила; д – кнопка перезапуска правила)

Теперь необходимо запустить правило, предварительно загрузив понятия из онтологии (рис. 26б, рис. 26в).

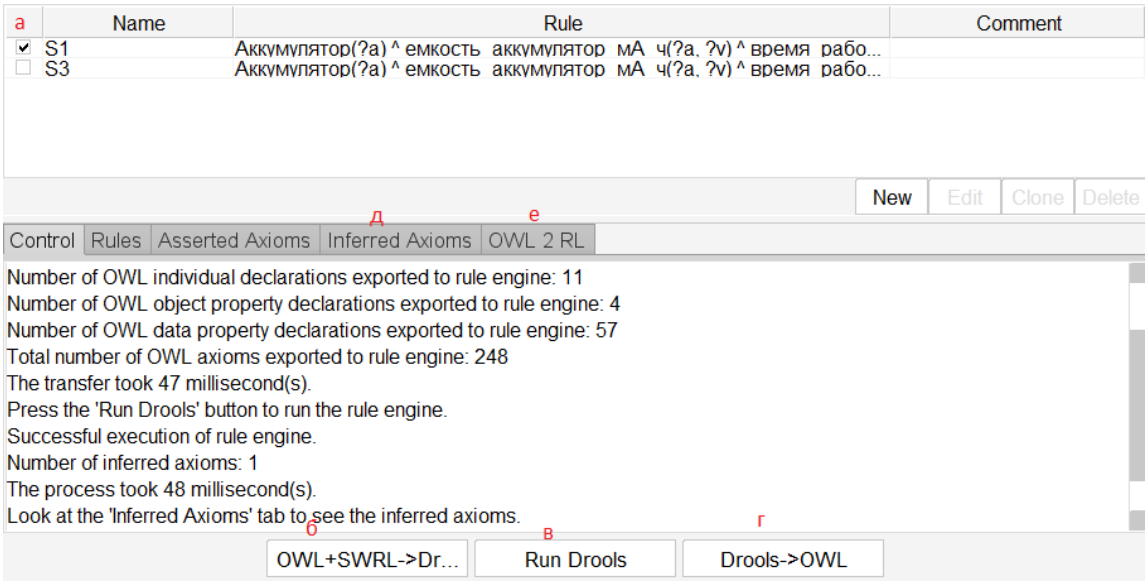

Рис. 26 Вкладка SWRL в системе Protégé (а – активность правила; б – кнопка загрузки понятий/аксиом из онтологии и списка правил; в – запуск вывода на основе правил; г – внесение выведенных понятий в онтологию; д – список выведенных понятий; е – настройки вывода)

После этого можно ознакомиться с выведенными аксиомами (рис. 26.д). В случае настроек по умолчанию их будет больше 100. Это связано с тем, что выполняется множество стандартных правил (например, формирование новых отношений в соответствии с описанием «inverseOf»). Для нашего примера это необязательно. Поэтому можно на вкладке «OWL 2 RL -> OWL2RL Control» выключить все таблицы стандартных правил. После этого в выведенных аксиомах останутся только аксиомы, полученные с использованием созданного правила.

Задания

1. Сформировать минимум 5 правил, выявляющие сомнительные описания различных характеристик.
2. Сформировать правило, которое в зависимости от емкости аккумулятора, тактовой частоты процессора, разрешения экрана и его диагонали будет формировать примерное время автономной работы ноутбука в режимах: игра, работа, ожидание.

СПИСОК ЛИТЕРАТУРЫ

1. Cyganiak R., Lanthaler M., Wood D. RDF 1.1 Concepts and Abstract Syntax // World Wide Web Consortium (W3C). – 2014. – 25 февраля [Электронный ресурс]. URL: http://www.w3.org/TR/rdf11-concepts/ (дата обращения: 20.05.2020).
2. Hayes P. J., Patel-Schneider P. F. RDF 1.1 Semantics // World Wide Web Consortium (W3C). – 2014. – 25 февраля [Электронный ресурс]. URL: https://www.w3.org/TR/rdf11-mt/ (дата обращения: 20.05.2020).
3. Brickley D., Guha R. V., McBride B. RDF Schema 1.1 // World Wide Web Consortium (W3C). – 2014. – 25 февраля [Электронный ресурс]. URL: https://www.w3.org/TR/rdf-schema/ (дата обращения: 20.05.2020).
4. W3C OWL Working Group OWL 2 Web Ontology Language Document Overview (Second Edition) // World Wide Web Consortium (W3C). – 2012. – 11 декабря [Электронный ресурс]. URL: https://www.w3.org/TR/owl2-overview/ (дата обращения: 20.05.2020).
5. Glimm, Birte & Horrocks, Ian & Motik, Boris & Stoilos, Giorgos & Wang, Zhe. (2014). Hermit: An Owl 2 Reasoner. Journal of Automated Reasoning. 53. 10.1007/s10817-014-9305-1.
6. Protégé 5 Documentation [Электронный ресурс]. URL: http://protegeproject.github.io/protege/ (дата обращения: 20.05.2020).
7. Horridge M., Patel-Schneider P. F. OWL 2 Web Ontology Language Manchester Syntax (Second Edition) // World Wide Web Consortium (W3C). – 2012. – 11 декабря [Электронный ресурс]. URL: https://www.w3.org/TR/owl2-manchester-syntax/ (дата обращения: 20.05.2020).
8. Harris S., Seaborne A., Prud'hommeaux E. SPARQL 1.1 Query Language // World Wide Web Consortium (W3C). – 2013. – 21 марта [Электронный ресурс]. URL: https://www.w3.org/TR/sparql11-query/
9. Horrocks I., Patel-Schneider P. F., Boley H., Tabet S., Grosof B., Dean M. SWRL: A Semantic Web Rule Language Combining OWL and RuleML // Member Submis-

sions W3C. – 2004. – 21 мая [Электронный ресурс]. URL: https://www.w3.org/Submission/SWRL/

10. Hassanpour S., O'Connor M.J., Das A.K. (2009) Exploration of SWRL Rule Bases through Visualization, Paraphrasing, and Categorization of Rules. In: Governatori G., Hall J., Paschke A. (eds) Rule Interchange and Applications. RuleML 2009. Lecture Notes in Computer Science, vol 5858. Springer, Berlin, Heidelberg
11. O'Connor, Martin & Das, Amar. (2009). SQWRL: a query language for OWL.