УДК 004.822
DOI: 10.15827/0236-235X.122.303-310

Дата подачи статьи: 18.10.17
2018. Т. 31. № 2. С. 303–310

ИЗВЛЕЧЕНИЕ СХЕМЫ ДАННЫХ ИЗ КОНЕЧНОЙ ТОЧКИ ДОСТУПА SPARQL

А.О. Мочалов [1], *аспирант, aomochalov@corp.ifmo.ru*
Д.И. Муромцев [1], *к.т.н., зав. кафедрой, d.muromtsev@gmail.com*

[1] *Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики (Университет ИТМО), Кронверкский просп., 49, г. Санкт-Петербург, 197101, Россия*

Статья посвящена проблеме извлечения схемы данных из конечной точки доступа SPARQL. Схема данных необходима для написания запросов и поиска информации, а также для оптимизации выполнения запросов.

В данной работе рассмотрены существующие методы для извлечения схем, отмечены плюсы и минусы каждого из них. Для разработки собственного метода дано определение схемы данных, под которой в данной работе принято считать словарь всех триплетов, где предикатами являются все фиксированные предикаты из пространства имен RDFS, а также триплеты, которые логически следуют из датасета в соответствии с семантикой RDFS, кроме являющихся элементами известных словарей (RDF, RDFS, Void, OWL, XML Schema, XSD). Элементы схемы из конечной точки доступа SPARQL будут извлекаться с помощью правил RDFS.

В работе используются не все правила семантики RDFS, так как, согласно принятому определению схемы данных, не все правила возвращают элементы схемы.

В статье подробно описан разработанный метод, представлена его архитектура. Для работы с правилами логического вывода используется система управления бизнес-правилами Drools. В работе отмечены плюсы и минусы созданного метода, показавшего ожидаемые результаты тестирования. Отмечено, что количество классов и свойств может быть различным у опубликованной и полученной разработанным методом схем. Это обусловлено лишь тем, что разработанный метод использует правила RDF.

Тестирование показало, что авторский метод не полностью покрывает СД, но вполне работоспособен.

Ключевые слова: *Semantic Web, linked data, sparql endpoint, owl, схема данных, набор данных, датасет*.

На данный момент существует большое количество датасетов (http://www.meloda.org/dataset-definition/), пригодных для повторного использования. Однако информация о том, какая используется *схема данных* (СД), не всегда доступна. Эта информация важна для разработчиков, так как СД отражает структуру данных, знание которой необходимо для написания запросов и поиска информации.

Большинство хранилищ датасетов (триплсторов) хранят СД для оптимизации выполнения запросов, однако на данный момент не существует универсального инструмента и средств для ее извлечения с помощью стандартных программных интерфейсов SPARQL endpoint (http://semanticweb.org/wiki/SPARQL_endpoint.html). Многие разработчики не поддерживают СД в актуальном состоянии, а это приводит к тому, что не все классы и свойства описаны в СД. Указанная проблема и будет решаться в данной работе.

С помощью разработанного авторами метода будут извлекаться из датасета все классы и свойства, которые должны принадлежать схеме RDFS [1]. Метод для извлечения схемы опирается на стандарт RDF 1.1 Semantics и представленные в нем стандартные правила для извлечения схемы. В данной работе не используется семантика OWL. СД извлекается из конечной точки SPARQL (SPARQL endpoint) только посредством SPARQL-запросов [2].

Обзор существующих методов

Исследования в области извлечения СД можно разделить на две категории. К первой относятся работы, которые используют датасеты, доступные локально в виде RDF-графа, а затем применяют разработанные алгоритмы и проводят анализ. Первый продукт, относящийся к данной работе, – RDF Digest [3]. Это инструмент для автоматического создания высказываний о RDF/S базы знаний. В качестве оценки результатов используются свойства релевантности и покрытия. Минусом данной работы является невозможность использования конечной точки доступа SPARQL.

Следующий инструмент первой категории – Loupe [4]. Это онлайн-инструмент, помогающий исследовать набор данных (датасеты) и узнать, какие словари (классы и свойства) используются в наборе данных. На данный момент представлена только демоверсия инструмента. В будущем планируется добавить способы обнаружения и отслеживания изменений в наборе данных и обновление только части индексов.

К первой категории также отнесем проект под названием DB2-RDF [5] – СУБД, создающую представление и механизм запросов к RDF-данным поверх реляционной БД. В данной работе интересен подход к проблеме извлечения СД для эффективного отображения схемы графа на реляционную модель.

Также следует упомянуть метод для параллельного извлечения OWL 2 EL-онтологий из связанных наборов данных. Разработчиками данного метода являются Huiying Li и Qiang Sima из Southeast University. Они предоставляют два метода извлечения данных: параллельный и непараллельный [6].

К первой категории также относится фреймворк ABSTAT [7], предназначенный для исследования и понимания больших наборов данных. Ключевой особенностью работы является использование минимального количества типов моделей для представления абстракции набора данных (http://abstat.disco.unimib.it:8880/about). В отличие от него описанный в статье метод извлекает СД из точки доступа SPARQL.

Ко второй категории относятся работы, в которых для извлечения СД используют SPARQL-запросы. Отличительной особенностью данного способа является то, что для каждого датасета пишутся свои запросы. К данной категории относится проект BaseKB (http://basekb.com/) – первый корректный преобразователь из Freebase в RDF. Автор работы [8] ставит перед собой задачу понимания содержания большой базы знаний, например, имеющей 800 миллионов фактов и 60 тысяч свойств. Для решения данной проблемы он предлагает воспользоваться языком SPARQL-запросов, чтобы задать вопросы о СД.

Представим сводную таблицу рассмотренных проектов (табл. 1).

Укажем отличительные особенности данной работы от двух ранее рассмотренных категорий. От первой категории работ она отличается тем, что вместо локальной загрузки дампа используется SPARQL endpoint, что позволяет исследовать СД датасетов, локальная загрузка которых невозможна по ряду причин (размер, частые изменения, приватность). От второй категории работ предлагаемая работа отличается тем, что предоставляет универсальный инструмент, предназначенный для пользователей, ничего не знающих о содержимом датасета, то есть о том, какие данные скрываются за точкой доступа SPARQL. Еще одной особенностью авторской работы в отличие от данной категории работ является формализованный подход: предлагается свое определение понятия СД и исследуются методы, подходящие для ее извлечения.

Описание разработанного метода

Анализ существующих методов для извлечения СД позволяет сделать вывод, что все методы загружают либо локально датасет, либо определенный набор высказываний через конечную точку SPARQL, а затем применяют разработанные алгоритмы. По результатам аналитического обзора была уточнена постановка задачи: необходимо разработать алгоритм для извлечения СД из связанного набора данных (https://www.w3.org/standards/semanticweb/data) через публичную точку доступа SPARQL, используя только язык запросов SPARQL.

Таблица 1

Существующие реализации методов по извлечению СД

Table 1

Existing implementations of methods for extracting data schemes

Проект	Стадия исследования
Loupe	Представлена демоверсия инструмента: предоставляет статистические данные (классы, свойства, тройки, пространства имен). В будущем планируется придумать способы обнаружения и отслеживания изменений в датасете и реализовать обновление только части индексов
MonetDB/ RDF	Инструмент для преобразование RDF-дампа в реляционную БД. В дальнейшем планируется улучшить алгоритмы, масштабирование, поиск по схеме, обновление наборов данных
Web Data Commons	Проект извлечения структурированных данных из Common Crawl, предоставляет извлеченные данные для загрузки в исследовательских целях
R2D	Готовая JDBC-оболочка для RDF-хранилищ, которая предоставляет инструмент для визуализации, а также инструмент для реляционного представления данных
DB2RDF (NoSQL Graph Support)	Представление и механизм запросов к RDF-данным поверх реляционной БД. В дальнейшем планируются улучшение производительности, поддержка стандарта языка SPARQL 1.1
RDF-Digest	Представлена готовая реализация продукта для автоматического создания высказываний о RDF/S базы знаний
ABSTAT	Представлен готовый фреймворк, помогающий исследовать большие связанные наборы данных

Для разработки метода было введено следующее определение СД.

СД – это словарь (https://www.w3.org/standards/semanticweb/ontology) всех триплетов, где предикатами являются все фиксированные предикаты из пространства имен RDFS [9], а также триплеты, которые логически следуют из датасета в соответствии с семантикой RDFS, кроме триплетов, являющихся элементами известных словарей (RDF, RDFS, Void, OWL, XML Schema, XSD). Правила логического вывода, следующие из семантики RDFS, представлены в таблице 2.

В третьей колонке для правил, которые определяют элементы схемы, представлены эквивалентные SPARQL-запросы. Для правил, которые не определяют элементы схемы, третья колонка пустая. Данные аксиомы (правила) применяются для конкретного датасета. В результате на выходе получаем словарь триплетов, который уникальным образом определяет СД для каждого датасета.

Таблица 2

Правила логического вывода, соответствующие семантике RDFS

Table 2

Inference rules corresponding to RDFS semantics

Правило	Описание	SPARQL-запрос
ICEXT[1](y) is defined to be { x : < x, y > is in IEXT(I(rdf:type)) }	Если есть такой **y**, что **x** является типом **y**, то **y** является классом	construct {?o rdf:type rdfs:Class} where {?s rdf:type ?o}
IC is defined to be ICEXT(I(rdfs:Class))	Интерпретация множества всех классов определена как расширение интерпретации множества классов в интерпретации rdfs:Class	construct where {?s rdf:type rdfs:Class}
LV is defined to be ICEXT(I(rdfs:Literal))	Множество литералов определены как расширение интерпретации множества классов в интерпретации rdfs:Literal	construct {?s rdf:type rdfs:Literal . ?s rdf:type rdfs:Class} where {?s rdf:type rdfs:Literal}
ICEXT(I(rdfs:Resource)) = IR	Расширение интерпретации множества всех классов в интерпретации rdfs:Resource равно интерпретации множества ресурсов	
ICEXT(I(rdf:langString)) is the set {I(E) : E a language-tagged string }	Множество всех классов в интерпретации rdf:langString является множеством интерпретации E, где E – language-tagged string	
for every other IRI aaa in D, ICEXT(I(aaa)) is the value space of I(aaa)	Для каждого IRI aaa из D множество всех классов в интерпретации aaa является значением пространства интерпретации aaa	
for every IRI aaa in D, I(aaa) is in ICEXT(I(rdfs:Datatype))	Для каждого IRI aaa в D интерпретация aaa принадлежит расширению интерпретации классов в интерпретации rdfs:Datatype	construct {?s rdf:type rdfs:Datatype . ?s rdf:type rdfs:Class} where {?s rdf:type rdfs:Datatype}
If < x,y > is in IEXT(I(rdfs:domain)) and < u,v > is in IEXT(x) then u is in ICEXT(y)	Если есть пара **x** и **y**, имеющих отношение rdfs:domain, и пара **u** и **v**, имеющих отношение **x**, тогда **u** является типом **y**	construct {?s rdfs:domain ?o . ?o rdf:type rdfs:Class . ?s rdf:type rdfs:Property} where {?s rdfs:domain ?o}
If < x,y > is in IEXT(I(rdfs:range)) and < u,v > is in IEXT(x) then v is in ICEXT(y)	Если есть пара **x** и **y**, имеющих отношение rdfs:range, и пара **u** и **v**, имеющих отношение **x**, тогда **v** является типом **y**	construct {?s rdfs:range ?o . ?o rdf:type rdfs:Class . ?s rdf:type rdfs:Property} where {?s rdfs:range ?o}
IEXT(I(rdfs:subPropertyOf)) is transitive and reflexive on IP	Расширение интерпретации rdfs:subPropertyOf транзитивно и рефлексивно на интерпретацию множества свойств	construct {?s rdfs:subPropertyOf ?o . ?s rdf:type rdfs:Property . ?o rdf:type rdfs:Property} where {?s rdfs:subPropertyOf ?o}
If <x,y> is in IEXT(I(rdfs:subPropertyOf)) then x and y are in IP and IEXT(x) is a subset of IEXT(y)	Если есть пара **x** и **y**, имеющих отношение rdfs:subPropertyOf, то **x** и **y** являются свойствами и **x** является подмножеством **y**	
If x is in IC then < x, I(rdfs:Resource) > is in IEXT(I(rdfs:subClassOf))	Если **x** является элементом множества классов, тогда пара <x, I(rdfs:Resource)> имеет расширение интерпретации rdfs:SubClassOf	
IEXT(I(rdfs:subClassOf)) is transitive and reflexive on IC	Расширение интерпретации rdfs:subClassOf транзитивно и рефлексивно на интерпретацию множества классов	construct {?s rdfs:subClassOf ?o . ?s rdf:type rdfs:Class . ?o rdf:type rdfs:Class} where {?s rdfs:subClassOf ?o}
If < x,y > is in IEXT(I(rdfs:subClassOf)) then x and y are in IC and ICEXT(x) is a subset of ICEXT(y)	Если есть пара **x** и **y**, имеющих отношение rdfs:subClassOf, то **x** и **y** являются классами и **x** является подмножеством **y**	
If x is in ICEXT(I(rdfs:ContainerMembershipProperty)) then: < x, I(rdfs:member) > is in IEXT(I(rdfs:subPropertyOf))	Если **x** является расширением интерпретации классов и интерпретации rdfs:ContainerMembershipProperty, то пара **x** и **rdfs:member** имеет отношение rdfs:subPropertyOf	
If x is in ICEXT(I(rdfs:Datatype)) then < x, I(rdfs:Literal) > is in IEXT(I(rdfs:subClassOf))	Если **x** принадлежит расширению интерпретации множества классов в интерпретации rdfs:Datatype, тогда любая пара <x, I(rdfs:Literal)> принадлежит расширению интерпретации множества классов в интерпретации rdfs:subClassOf	construct {?p rdf:type owl:DatatypeProperty} where {?s ?p ?o FILTER (isLiteral(?o))}

Каждый SPARQL-запрос в третьей колонке таблицы 2 в качестве формы результата имеет construct. Это обусловлено тем, что construct возвращает RDF-граф, который задается специальным

шаблоном, то есть для каждого очередного решения в construct-шаблон вместо переменных подставляются uri, промежуточные узлы, или литералы.

В данной работе будут использоваться не все правила семантики RDFS, так как, согласно определению СД, не все они возвращают элементы схемы. В таблице 3 представлены SPARQL-запросы, которые будут использоваться в данной работе, и соответствующие им алиасы.

Таблица 3

SPARQL-запросы, выполняемые над точкой доступа

Table 3

SPARQL queries performed on the access point

Алиас	SPARQL-запрос
rdfs-1	construct {?o rdf:type rdfs:Class} where {?s rdf:type ?o}
rdfs-2	construct where {?s rdf:type rdfs:Class}
rdfs-3	construct {?s rdf:type rdfs:Literal . ?s rdf:type rdfs:Class} where {?s rdf:type rdfs:Literal}
rdfs-4	construct {?s rdf:type rdfs:Datatype . ?s rdf:type rdfs:Class} where {?s rdf:type rdfs:Datatype}
rdfs-5	construct {?s rdfs:subPropertyOf ?o . ?s rdf:type rdf:Property . ?o rdf:type rdf:Property} where {?s rdfs:subPropertyOf ?o}
rdfs-6	construct {?s rdfs:subClassOf ?o . ?s rdf:type rdfs:Class . ?o rdf:type rdfs:Class} where {?s rdfs:subClassOf ?o}
rdfs-7	construct {?s rdfs:domain ?o . ?o rdf:type rdfs:Class . ?s rdf:type rdf:Property} where {?s rdfs:domain ?o}
rdfs-8	construct {?s rdfs:range ?o . ?o rdf:type rdfs:Class . ?s rdf:type rdf:Property} where {?s rdfs:range ?o}
rdfs-9	construct {?p rdf:type rdf:Property} where {?s ?p ?o}

Согласно определению СД, из точки доступа извлекаются элементы схемы, которые логически следуют из семантики RDFS, кроме элементов, описанных в известных схемах, поэтому к каждому construct-запросу из таблицы 3 добавляется фильтр. Например, запрос rdfs-1 будет выглядеть следующим образом:

```
construct {?o rdf:type rdfs:Class} where {?s rdf:type ?o
FILTER(!STRSTARTS(STR(?o),
'http://www.w3.org/2000/01/rdf-schema#') &&
!STRSTARTS(STR(?o),
'http://www.w3.org/2002/07/owl#') &&
!STRSTARTS(STR(?o), 'http://www.w3.org/1999/02/22-
rdf-syntax-ns#') && !STRSTARTS(STR(?o),
'http://www.w3.org/XML/1998/namespace') &&
!STRSTARTS(STR(?o),
'http://www.w3.org/2001/XMLSchema#') &&
!STRSTARTS(STR(?o), 'http://rdfs.org/ns/void#') &&
!isBlank(?o))}
```

Необходимо также отметить, что запросы, определенные с точки зрения семантики RDFS, и запросы, которые непосредственно будут выполняться над точкой доступа, в результате могут оказаться не одними и теми же. Это обусловлено тем, что запрос может выполняться долго над точкой доступа и в результате или не будет получен ответ от точки доступа, или получены неполные результаты. Эта проблема стоит особенно остро, так как заранее сложно сказать, какой запрос окажется проблемным для конкретной точки доступа. В качестве решения данной проблемы предлагается извлекать данные из точки доступа по частям с помощью команд LIMIT и OFFSET. В данной работе не извлекаются классы и свойства, принадлежащие известным схемам, таким как RDFS, Void, XML Schema, RDF, OWL. Из выборки игнорируются blank nodes, которые не являются частью данных, а выполняют служебную функцию для представления OWL-аксиом в виде RDF. Данные элементы планируется использовать в следующей работе.

В таблице 4 указаны плюсы и минусы разработанного метода.

Таблица 4

Плюсы и минусы разработанного метода

Table 4

Advantages and disadvantages of the developed method

Свойство метода	Плюс	Минус
Локальная загрузка дампа	Не требуется	Невозможно применять алгоритмы к локальному датасету
Способ извлечения СД	Простой инструмент для извлечения СД (SPARQL-запросы)	Анализ каждого запроса

Главным плюсом, а также отличительной особенностью метода является то, что не требуется загружать дамп локально на компьютер для извлечения схемы, необходимо только предоставить URL с конечной точкой SPARQL. Из плюса вытекает отрицательная сторона метода. Минусом является то, что необходимо анализировать каждый запрос: строить быстрые производительные запросы. Запросов не должно быть слишком много, они не должны быть очень сложными. Также запрос не должен передавать огромное количество информации, что может привести к его неуспешному выполнению. Под неуспешным запросом понимаем возникновение ситуации, когда метод не обеспечивает требуемую производительность, а также выдачу требуемых результатов обработки запроса.

Архитектура метода

Опишем общую архитектуру разработанного метода. На начальном этапе имеются правила логического вывода RDFS, представленные в таблице 2. Для редактирования и создания новых правил необходимо использовать систему управления биз-

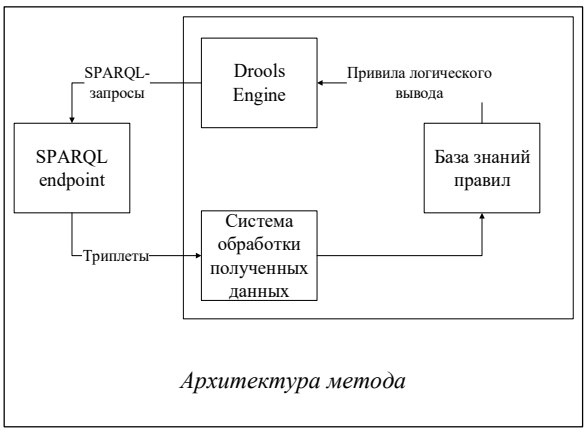
нес-правилами. В данной работе используется система Drools. Каждое правило логического вывода из таблицы 2 было переписано на естественном языке с использованием Drools. Например, первое правило из таблицы 2:

ICEXT(y) is defined to be { x : < x,y > is in IEXT(I(rdf:type)) }

будет описано в Drools соответствующим образом:

```
rule "Predicate is rdf:type"
when
   There is a triple with predicate rdf:type
then
   add all object in set classes;
end
```

Главные сущности, используемые в работе (Subject, Predicate, Object), описаны с помощью Java-классов с необходимыми параметрами. Наконец, каждое правило будет переведено в SPARQL-запрос, с помощью которого извлекаются данные из точки доступа SPARQL endpoint. Общая архитектура разработанного метода представлена на рисунке.



Архитектура метода

Реализация метода

Для реализации метода необходимо выделить сущности и представить их в виде классов. Было создано три класса: Subject, Predicate, Object, каждый из которых имеет строковый параметр value. На данный момент классы отличаются лишь выводом. Для human-readable (http://opendatahandbook.org/glossary/en/terms/human-readable/) представления в Drools (http://www.drools.org/) используется DSL (Domain-specific language (http://searchmicroservices.techtarget.com/definition/domain-specific-language-DSL)) – предметно-ориентированный язык. Так как необходимо разработать инструмент для специалиста в области баз знаний (http://searchcrm.techtarget.com/definition/knowledge-base) и семантического веба (https://www.w3.org/standards/semanticweb/), а не для программиста или технического специалиста, выбран был именно DSL. Для написания правил в Drools были подготовлены два файла, в которых описаны правила логического вывода RDFS. В первом файле (DSLR-файл (https://docs.jboss.org/drools/release/5.2.0.Final/drools-expert-docs/html/ch05.html)) описаны правила на естественном языке. Например,

```
rule "Predicate is rdf:type"
when
   There is a triple with predicate rdf:type
then
   add all object in set classes;
end
```

Правило описывает, что у всех троек с предикатом rdf:type (https://www.w3.org/TR/rdf-schema/#ch_type) все объекты принадлежат множеству классов, что соответствует RDFS-правилам. Во втором файле (DSL-файл) описана логика перевода Drools-правила в SPARQL-запрос. Например, первому правилу из DSLR в файле DSL соответствует запись

```
[when] There is a triple with predicate {predicate} =
reader.HttpReader( addAllObjectInSetClass("?s {predicate}
?o BIND({predicate} as ?p)") != 0)
[then] add all object in set classes = Sys-
tem.out.println("Matched rule: If there is a triple with predi-
cate {predicate} Then add all object in set classes")
```

При выполнении данного правила будет создан объект класса Predicate со значением переменной value, равным rdf:type. Далее будет вызван метод addAllObjectInSetClass, который принимает на вход строку, содержащую условие для SPARQL-запроса. В рассматриваемом примере это будет “?s rdf:type ?o BIND({predicate as ?p})”. Полный SPARQL-запрос будет выглядеть следующим образом: SELECT * WHERE {?s rdf:type ?o BIND({predicate as ?p})}. Данный запрос вернет все тройки, у которых объект будет являться типом rdfs:Class (https://www.w3.org/TR/rdf-schema/#ch_class).

Представим правила в объектном виде:

```
rule "Predicate is rdf:type"
when
      $res :
   reader.HttpReader(addAllObjectInSetClass("?s
   rdf:type ?o BIND({predicate} as ?p)") != 0)
then
      System.out.println("Matched rule: If there is a
   triple with predicate rdf:type Then add all
   object in set classes");
end
```

Так как используемые объекты представлены объектами Java-классов, логику и реализацию можно менять в самих этих классах. Для оптимизации запросов изменения вносятся в Drools-правила.

Берется каждое из правил, реализующих семантику RDFS, и выполняется над точкой доступа с помощью SPARQL-запросов. Правила выводят выражения (триплеты) (https://www.w3.org/TR/rdf-schema/#ch_class), которые помещаются в множество. Каждый новый выведенный триплет проверяется на уникальность, то есть один и тот же триплет может быть выведен несколько раз, но правило применяется единожды. В результате получается множество триплетов, которые однозначно определяют СД.

Тестирование реализованного метода и анализ результатов

Для тестирования готового метода будут использоваться точки доступа с известной схемой данных. Для примера возьмем одни из самых популярных публичных точек доступа: Semantic Web Dog Food (SWDF) (http://www.scholarlydata.org/) со схемой conference-ontology (http://www.scholarlydata.org/ontology/), The Listening Experience Database (LED) (http://led.kmi.open.ac.uk/rdf/export/led-SNAPSHOT.nt.bz2) со схемой данных led ontology (http://led.kmi.open.ac.uk/ontology), lexvo.org (http://led.kmi.open.ac.uk/ontology) со схемой lexvo ontology (http://lexvo.org/ontology), open foodfacts.org (http://fr.openfoodfacts.org/data/fr.openfoodfacts.org.products.rdf) со схемой данных food ontology (http://data.lirmm.fr/ontologies/food), Talis Aspire – Roehampton University (http://resourcelists.roehampton.ac.uk/lists/CB05FC97-2D5B-C51A-E8D1-CAAC70BDF2D8.rdf), который реализует схему schema.org (http://schema.org/docs/developers.html#defs). После проведения эксперимента необходимо проанализировать полученные результаты и сделать необходимые выводы. Особенно следует подчеркнуть, насколько полно разработанный метод выделил множество триплетов, которые должны соответствовать схеме, и как долго выполнялись SPARQL-запросы, то есть в качестве критериев оценки используются время выполнения SPARQL-запроса и покрытие авторского метода.

В таблицах 5 и 6 представлены время выполнения каждого запроса для тестовых точек доступа и сравнение количества классов и свойств у полученной и оригинальной схем данных соответственно.

Таблица 5

Время выполнения каждого запроса для тестовых точек доступа

Table 5

The execution time of each query for test access points

Запрос	Время выполнения запроса, мс					
	SWDF	LED	lexvo.org	openfoodfacts.org	Talis Aspire - Roehampton University	Dbpedia
rdfs-1	13	46	50	31	63	606
rdfs-2	12	14	25	14	19	20
rdfs-3	9					
rdfs-4	14					
rdfs-5	13				54	
rdfs-6	10	13		23	89	204
rdfs-7	10	26		41		361
rdfs-8	10	23		22		121
rdfs-9	7	22	21	26	29	466

Таблица 6

Таблица результатов тестирования

Table 6

A test results table

Датасет	Опубликованная схема	Полученная схема	Среднее время выполнения запросов, мс
SWDF	Classes: 34 Properties: 49	Classes: 72 Properties: 64	11
LED	Classes: 53 Properties: 1	Classes: 44 Properties: 73	24
lexvo.org	Classes: 9 Properties: 4	Classes: 5 Properties: 15	32
Open-foodfacts.org	Classes: 10 Properties: 8	Classes: 9 Properties: 94	26
Talis Aspire – Roehampton University	Classes: 599 Properties: 136	Classes: 592 Properties: 111	51
Dbpedia	Classes: 303 Properties: 2689	Classes: 303 Properties: 9081	296

Анализ результатов тестирования

Полученные результаты можно разбить на несколько групп:

– количество классов и свойств в полученной СД больше, чем в опубликованной;

– количество классов в полученной СД больше, чем в опубликованной; количество свойств в полученной меньше или равно, чем в опубликованной;

– количество классов в полученной СД меньше или равно, чем в опубликованной; количество свойств в полученной СД больше, чем в опубликованной;

– количество классов и свойств в полученной СД меньше или равно количеству классов и свойств в опубликованной.

Большее количество классов и свойств в полученной СД свидетельствует о том, что опубликованные СД неполные, не содержат все элементы схемы. Данная ситуация для Linked Data обусловлена тем, что разработчики датасетов не поддерживают в актуальном состоянии СД [10]. Когда же количество классов и свойств в опубликованной СД больше, чем в полученной, опубликованные СД содержат blank node, которые игнорируются в данной работе.

К первой группе полученных результатов относится тестирование для точек доступа SWDF. К третьей группе – тестирование для датасета LED, lexvo.org, openfoodfacts.org. К четвертой группе – тестирование датасета Talis Aspire – Roehampton University. Во вторую группу ни один датасет при тестировании не попал.

Выясним, почему количество классов и свойств в полученной СД меньше, чем в опубликованной. Для примера возьмем датасет openfoodfacts.org. Посмотрим, почему количество классов в опубликованной схеме больше на один, чем в полученной схеме. Класс, который не был излечен правилами http://data.lirmm.fr/ontologies/food#Recipe. Алгоритмом не был обработан триплет <http://data.lirmm.fr/ontologies/food#Recipe> rdfs:subClassOf owl:Thing. Данный класс не был излечен, так как не использовалась семантика OWL.

Эти же выводы результатов тестирования применимы для всех остальных датасетов.

Заключение

Перед началом работы перед авторами стояла проблема, как по имеющемуся датасету получить полную схему данных. В результате был разработан инструмент для извлечения СД из конечной точки доступа SPARQL. Инструмент позволяет получить схему датасета для дальнейшей работы: правильное построение запросов, понимание построения логики датасета. Разработанный метод является универсальным, так как для извлечения СД используется только язык запросов SPARQL. Предлагаемый метод для извлечения схемы опирается на стандарт RDF 1.1 Semantics и представленные в нем стандартные правила извлечения. Семантику OWL можно реализовывать в дальнейшей работе. Тестирование показало, что авторский метод не полностью покрывает СД, но вполне работоспособен. В дальнейшем планируется добавить необходимые запросы и оптимизировать их для извлечения недостающих элементов схемы.

Литература

1. RDF 1.1 Semantics. URL: https://www.w3.org/TR/rdf11-mt/ (дата обращения: 28.09.2017).
2. SPARQL Query Language for RDF. URL: https://www.w3.org/TR/rdf-sparql-query/ (дата обращения: 28.12.2017).
3. TroullinouAffiliated G., Kondylakis H., Daskalaki E., Plexousakis D. RDFDigest: Efficient Summarization of RDF/S KBs. URL: https://link.springer.com/chapter/10.1007/978-3-319-18818-8_8 (дата обращения: 28.09.2017).
4. Mihindukulasooriya N., Poveda-Villalion M., Garcia-Castro R., Gomez-Perez A. Loupe – An Online Tool for Inspecting Datasets in the Linked Data Cloud. URL: https://www.researchgate.net/publication/283048477_Loupe_-_An_Online_Tool_for_Inspecting_Datasets_in_the_Linked_Data_Cloud (дата обращения: 28.09.2017).
5. Bornea A., Dolby J., Kementsietsidis A., Srinivas K., Dantressangle P., Udrea O., Bhattacharjee B. Building an Efficient RDF Store Over a Relational Database. URL: https://www.researchgate.net/publication/262162010_Building_an_efficient_RDF_store_over_a_relational_database (дата обращения: 28.09.2017).
6. Li H. Parallel mining of OWL 2 EL ontology from large linked datasets. URL: https://www.researchgate.net/publication/276106914_Parallel_mining_of_OWL_2_EL_ontology_from_large_linked_datasets (дата обращения: 28.09.2017).
7. Spahiu B., Porrini R., Palmonari M., Rula A., Maurino A. ABSTAT: Ontology-driven Linked Data Summaries with Pattern Minimalization. URL: http://km.aifb.kit.edu/ws/sumpre2016/paper_3.pdf (дата обращения: 28.12.2017).
8. Houle P. How to introspect the Freebase schema with SPARQL. URL: http://blog.databaseanimals.com/how-to-introspect-the-freebase-schema-with-sparql (дата обращения: 28.09.2017).
9. RDF Schema 1.1. URL: https://www.w3.org/TR/2014/PER-rdf-schema-20140109/ (дата обращения: 28.09.2017).
10. Miličić V. Problems of Linked Data. URL: http://milicicvuk.com/blog/2011/07/26/problems-of-linked-data-14-identity/ (дата обращения: 28.09.2017).

Software & Systems
DOI: 10.15827/0236-235X.122.303-310

Received 18.10.17
2018, vol. 31, no. 2, pp. 303–310

EXTRACTING A DATA SCHEMA FROM THE SPARQL ENDPOINT

A.O. Mochalov [1], *Postgraduate Student, aomochalov@corp.ifmo.ru*
D.I. Muromtsev [1], *Ph.D. (Engineering), Head of Chair, d.muromtsev@gmail.com*

[1] *The National Research University of Information Technologies, Mechanics and Optics, Kronverksky Ave. 49, St. Petersburg, 197101, Russian Federation*

Abstract. The paper is devoted to the problem of extracting a data schema from the SPARQL endpoint. The data schema is necessary for writing queries and searching information, as well as for optimizing query execution.

The paper considers the existing methods for extracting schemes, notes minuses and pluses of each method. It defines the notion of a data schema to develop own method. A data schema is a dictionary of all triplets, where the predicates are all fixed predicates from the RDFS namespace, as well as triplets that logically follow from the dataset according to RDFS semantics, except for triplets that are elements of well-known dictionaries (RDF, RDFS, Void, OWL, XML Schema, XSD). The elements from the SPARQL endpoint schema will be retrieved using RDFS rules. The work does not use all RDFS semantics rules as not all rules return the elements of the schema according to the definition of the data scheme adopted in this paper.

The paper describes the developed method in detail and presents its architecture. It uses the business rules management system Drools to work with inference rules. The paper presents the advantages and disadvantages of the developed method that showed the expected test results.

It it noted that the number of classes and properties can be different for the schemes developed and obtained by the developed method. This is only due to the fact that the developed method uses RDF rules.

Keywords: Semantic Web, linked data, SPARQL endpoint, OWL, data schema, dataset.

References

1. *RDF 1.1 Semantics*. Available at: https://www.w3.org/TR/rdf11-mt/ (accessed September 28, 2017).
2. *SPARQL Query Language for RDF*. Available at: https://www.w3.org/TR/rdf-sparql-query/ (accessed September 28, 2017).
3. Troullinou G., Kondylakis H., Daskalaki E., Plexousakis D. *RDFDigest: Efficient Summarization of RDF/S KBs*. Available at: https://link.springer.com/chapter/10.1007/978-3-319-18818-8_8 (accessed September 28, 2017).
4. Mihindukulasooriya N., Poveda-Villalion M., Garcia-Castro R., Gomez-Perez A. *Loupe - An Online Tool for Inspecting Datasets in the Linked Data Cloud*. Available at: https://www.researchgate.net/publication/283048477_Loupe_-_An_Online_Tool_for_Inspecting_Datasets_in_the_Linked_Data_Cloud (accessed September 28, 2017).
5. Bornea A., Dolby J., Kementsietsidis A., Srinivas K., Dantressangle P., Udrea O., Bhattacharjee B. *Building an Efficient RDF Store Over a Relational Database*. Available at: https://www.researchgate.net/publication/262162010_Building_an_efficient_RDF_store_over_a_relational_database (accessed September 28, 2017).
6. Li H. *Parallel mining of OWL 2 EL ontology from large linked datasets*. Available at: https://www.researchgate.net/publication/276106914_Parallel_mining_of_OWL_2_EL_ontology_from_large_linked_datasets (accessed September 28, 2017).
7. Spahiu B., Porrini R., Palmonari M., Rula A., Maurino A. *ABSTAT: Ontology-driven Linked Data Summaries with Pattern Minimalization*. Available at: http://km.aifb.kit.edu/ws/sumpre2016/paper_3.pdf (accessed September 28, 2017).
8. Houle P. *How to introspect the Freebase schema with SPARQL*. Available at: http://blog.databaseanimals.com/how-to-introspect-the-freebase-schema-with-sparql (accessed September 28, 2017).
9. *RDF Schema 1.1*. Available at: https://www.w3.org/TR/2014/PER-rdf-schema-20140109/ (accessed September 28, 2017).
10. Miličić V. *Problems of Linked Data*. Available at: http://milicicvuk.com/blog/2011/07/26/problems-of-linked-data-14-identity/ (accessed September 28, 2017).

Примеры библиографического описания статьи

1. Мочалов А.О., Муромцев Д.И. Извлечение схемы данных из конечной точки доступа SPARQL // Программные продукты и системы. 2018. Т. 31. № 2. С. 303–310. DOI: 10.15827/0236-235X.122.303-310.

2. Mochalov A.O., Muromtsev D.I. Extracting a data schema from the SPARQL endpoint *Programmnye produkty i sistemy* [Software & Systems]. 2018, vol. 31, no. 2, pp. 303–310 (in Russ.). DOI: 10.15827/0236-235X.122.303-310.