УДК 004.896, 004.414.23, 004.4'242

МОДЕЛЬ-ОРИЕНТИРОВАННЫЙ ПОДХОД К ПОСТРОЕНИЮ СВЯЗАННЫХ ДАННЫХ НА ОСНОВЕ РАЗНОРОДНЫХ ИСТОЧНИКОВ

С.В. Лебедев

Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина) и ООО «Датафабрик», Санкт-Петербург, Россия
lebedev.sv.etu@gmail.com

Аннотация

Сегодня доступно большое число различных источников данных. Многие источники представляют значительную ценность для принятия обоснованных решений в различных предметных областях. Простота и эффективность использования данных зависят от возможности их интеграции в единую модель, последовательно и всесторонне описывающую интересующий предмет. Получению такой модели препятствует то, что элементы разных источников не связаны друг с другом, а сами данные представлены в различных форматах. Несмотря на то, что консорциум W3C предложил языки единообразного и связанного описания данных, до сих пор существует значительное число издателей, публикующих несвязанные данные. В настоящей работе предлагается подход, на основе которого может быть создан эффективный инструмент для обработки больших объёмов несвязанных данных. Новым является автоматизация построения онтологии процесса связывания данных на основе отображения множества элементов онтологии, описывающей источник данных, во множество элементов онтологии предметной области. Полученная онтология процесса связывания может служить для специализаций этого процесса пользователем, для дальнейшей его оптимизации и реализации на различных вычислительных платформах. В частности, в работе продемонстрирована возможность генерации программного кода процесса связывания, исполняемого на высокопроизводительной масштабируемой платформе. Полученные результаты позволяют говорить о целесообразности развития предложенного подхода.

Ключевые слова: *связанные данные, модель процесса связывания, автоматизация построения модели, генерация программного кода, масштабируемые решения.*

Цитирование: *Лебедев, С.В.* Модель-ориентированный подход к построению связанных данных на основе разнородных источников / С.В. Лебедев // Онтология проектирования. – 2019. – Т.9, №1(31). – С.101-116. – DOI: 10.18287/2223-9537-2019-9-1-101-116.

Введение

Консорциум W3C (*World Wide Web Consortium*) разрабатывает и внедряет стандарты Всемирной Паутины (*World Wide We*b). Деятельность консорциума включает технологическое поддержание процессов построения глобальной сети данных. Некоторое время эта деятельность развивалась в рамках построения так называемой Семантической Паутины (*Semantic Web*) [1], а позднее была поглощена деятельностью (W3C *Data Activity*) по созданию Паутины Данных (*Web of Data*) [2]. Одна из основных задач этой деятельности заключается в том, чтобы перейти от паутины документов к паутине данных и сделать данные доступными для машинной обработки.

Для решения этой задачи несколькими рабочими группами были созданы стандарты языков представления данных (RDF/RDFs, OWL) и написания запросов к данным (SPARQL). Одной из отличительных черт этих языков является ориентация на использование междуна-

родных идентификаторов ресурсов (IRI, *International Resource Identifiers*) [3], позволяющих однозначно идентифицировать элемент данных на множестве наборов данных. Перечисленные стандарты составляют технический аспект решения.

Организационный аспект представлен требованиями к публикуемым данным. Первая группа требований определяет так называемые Связанные Данные (*Linked Data*) [4]. Требования сформулированы в виде правил, включающих требование использовать языки RDF и SPARQL. Вторая группа требований сформулирована в виде рейтинга открытости связанных данных. Наивысший рейтинг в пять звёзд присваивается данным, которые распространяются по открытой лицензии, используют стандарты W3C (RDF и SPARQL), имеют связи с данными, предоставляемыми третьими лицами [4]. Чем большему числу требований соответствуют данные, тем они более доступны для машинной обработки.

Общее количество наборов *Открытых Связанных Данных* растёт [5], но не охватывает некоторые важные источники. Одним из значимых поставщиков наборов данных является правительство: данные поставляются различными государственными органами. Если обратиться к сайту Реестра наборов открытых данных, то окажется, что из 21468 наборов только 5 представлено в формате RDF [6]. Большинство же данных представлено в форматах CSV, XML и JSON. Основным назначением форматов CSV, XML и JSON является представление данных в виде, не зависящем от приложения, в котором данные были созданы или использованы. Основное их назначение – это *сериализация* данных для хранения или передачи (RDF может быть также записан в XML) [7]. Предназначение RDF/OWL – это формальное и целостное описание предметной области (ПрО). Отсюда, RDF/OWL можно рассматривать как средство представления единой модели данных разделяемого экспертами описания ПрО.

Таким образом, требуется решить задачу построения связанных данных на основе данных, записанных в форматах сериализации (CSV, XML и JSON).

1 Задача построения связанных данных

Преимущество использования связанных данных (помимо машинной обработки) состоит в том, что они содержат больше сведений о рассматриваемом предмете, и, следовательно, позволяют принять относительно него более обоснованное решение. Использование единых семантических форматов (RDF/RDFs, OWL, SPARQL) позволяет унифицировать и тем самым упростить доступ к данным. Но большинство поставщиков публикует несвязанные данные, и задача их связывания ложится на плечи потребителя.

Концептуально связывание данных предполагает переход от множества наборов данных, описанных с помощью разрозненных схем, к набору, описанному с помощью одной или множества связанных схем. В рамках рассматриваемых технологий целевая схема представлена в форме онтологии ПрО. Связывание включает решение минимум трёх подзадач:

1) построение экземпляров классов ПрО;
2) инициализацию примитивных свойств экземпляров;
3) связывание экземпляров через объектные свойства.

Примером связывания может служить построение расширенного реестра коммерческих компаний на основе разнородных источников. В качестве источников могут выступать:

- сведения из Единого государственного реестра юридических лиц (ЕГРЮЛ) [8] (основной источник; полный набор является платным, представлен в формате XML);
- открытые данные Федеральной налоговой службы (ФНС) [9], например, Реестр дисквалифицированных лиц (формат CSV), Единый реестр субъектов малого и среднего предпринимательства (формат XML), Сведения о среднесписочной численности работников организации (формат XML) и др.;

- сведения по заключенным контрактам (формат JSON) [10] и т.п.

Реестр, созданный на основе множества перечисленных источников, позволит получать детальный «портрет» каждой компании и принимать более обоснованные (чем в случае использования только ЕГРЮЛ) решения, например, в рамках мероприятия по проверке контрагента и соблюдения должной осмотрительности.

Схематично система связывания данных представлена на рисунке 1. Модель ПрО готовится экспертом. Содержание и структура модели определяются предметной логикой.

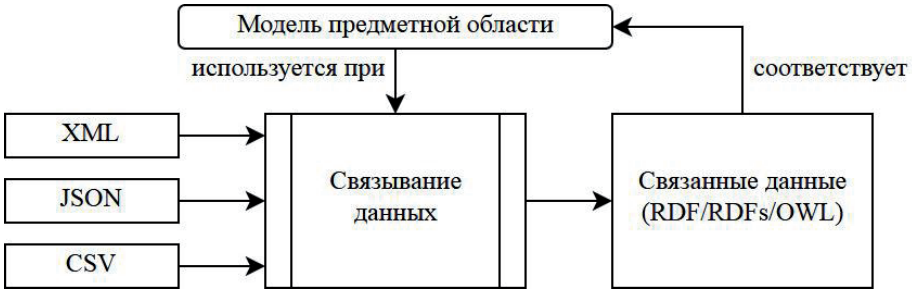

Рисунок 1 – Схема связывания данных из разнородных источников

На основе анализа работ [11-13], описывающих требования к инструментам построения связанных данных, и представленного варианта связывания данных сформулированы функциональные и нефункциональные требования к системе.

Основными функциональными требованиями, которые можно рассматривать как обязательный минимум для средства связывания данных, являются:

- возможность связывания разнородных документов: CSV (строки таблицы), XML (иерархия элементов, атрибуты элементов), JSON (массивы, структуры, пары имя-значение), онтологии в формате RDF/OWL (тройки субъект-предикат-объект);
- возможность использования нескольких источников для построения единой модели ПрО;
- поддержка функций, аргументами которых являются значения элементов источника данных, а значениями функций – элементы RDF тройки (субъект, объект, предикат).

К нефункциональным требованиям относятся:

1) простота подключения нового источника данных (подключение источника предполагает связывание модели источника данных и модели ПрО);
2) простота изменения модели ПрО;
3) возможность обработки больших объемов данных за ограниченное время;
4) возможность встраивания в имеющуюся вычислительную инфраструктуру.

Первое нефункциональное требование актуально в случае периодического появления новых источников и пересмотра схем данных уже подключенных источников.

Появление нового источника потребует расширения текущей модели ПрО. Требование актуально и в случае периодического пересмотра экспертного взгляда на ПрО. Инструмент изменения модели ПрО должен предоставлять абстракции уровня эксперта ПрО.

Размер данных и ограничения на время их обработки определяются ПрО. Следовательно, наиболее универсальным будет масштабируемое решение.

Четвёртое требование обусловлено тем, что задача связывания не является конечной: на этом этапе готовятся данные для последующего анализа и принятия решений.

Первые два нефункциональных требования определяют аспект взаимодействия пользователя с программным средством, третье и четвёртое требование – аспекты реализации.

Сформулированные требования позволяют сравнить известные инструменты связывания данных.

2 Инструменты связывания данных

Множество доступных инструментов можно условно разделить на две группы.

Первую составляют конвейеры обработки данных, реализующие принципы потокового программирования, изложенные, например, в [14]. Конвейер обработки данных представляет собой граф узлов-обработчиков, соединённых каналами передачи данных. Отдельно выделяют инструменты загрузки-преобразования-выгрузки данных (ETL, *Extract-Transform-Load*) [15]. Среди ETL-инструментов, поддерживающих работу с семантическими стандартами, можно выделить *UnifiedViews*[1] [16], *LinkedPipes*[2] [17], *Karma*[3] [18]. К преимуществам таких платформ можно отнести наличие расширенного функционала:

- извлечения (адаптеры к различным источникам данных, например, к базам данных, брокерам сообщений, веб-сервисам и т.п.);
- преобразования (фильтрация, переход от одной схемы к другой и т.п.);
- выгрузки данных (схожие с адаптерами для извлечения, но работающие на запись);
- наличие визуальных средств описания конвейера;
- предоставление (некоторыми платформами) возможности масштабирования.

К недостаткам подобных платформ можно отнести:

- невозможность глубокой интеграции в имеющийся вычислительный процесс;
- ограничения по масштабированию;
- отсутствие возможности интегрировать данные из нескольких источников в рамках одной операции (для преодоления этой трудности предлагается сначала обрабатывать каждый источник в отдельности, а затем связывать полученные результаты).

Вторую группу составляют узкоспециализированные программные решения, оформленные в виде Java-библиотек. К наиболее развитым инструментам относятся RML-*Mapper*[4] [12, 19] и SPARQL-*Generate*[5] [13, 20]. К достоинствам данных решений можно отнести более широкие возможности для связывания, включая возможность создания элементов узлов ETL платформ на их основе. Основной недостаток названных инструментов – это отсутствие возможностей для масштабирования.

3 Прототип платформы связывания данных на основе моделей

Исходным пунктом связывания данных является построение отображения элементов источника данных во множество элементов ПрО. В первой группе инструментов для этого используется графический интерфейс, во второй – специализированный язык. На основе этого отображения реализуется вычислительный процесс. В [12] выделены следующие варианты реализации процесса:

- процесс, управляемый отображениями (*mapping-driven*);
- процесс, управляемый данными (*data-driven*);
- смешанное управление.

Реализация, управляемая отображениями, наиболее прямолинейна, но может потребовать многократное прохождение элементов источника данных. Это объясняется тем, что отображения обрабатываются независимо друг от друга. Так, например, отображение одного и того же элемента данных на два разных элемента ПрО рассматриваются как независимые и обрабатываются за два прохода по одним и тем же элементам данных.

[1] https://github.com/UnifiedViews.
[2] https://github.com/linkedpipes.
[3] http://usc-isi-i2.github.io/karma/.
[4] https://github.com/RMLio/RML-Mapper.
[5] https://github.com/sparql-generate/sparql-generate.

Реализация, управляемая данными, учитывает связи между отображениями и сокращает число проходов до минимума. Для примера из предыдущего абзаца потребуется один проход по элементам данных, так как учитывается, что разные элементы ПрО строятся на основе одного и того же элемента источника данных. Но такая реализация более сложна, так как требует анализа зависимостей между отображениями и соответствующей оптимизации процесса.

3.1 Подход к связыванию данных на основе моделей

В работе предлагается подход, который можно отнести к смешанным реализациям:

- процесс, управляемый отображениями, строит *модель* процесса связывания данных;
- полученная модель описывает процесс связывания данных, управляемый данными.

Полученная модель может быть использована как для непосредственной реализации, так и для дальнейшей оптимизации и/или настройки пользователем.

Общая схема описанного подхода представлена на рисунке 2.

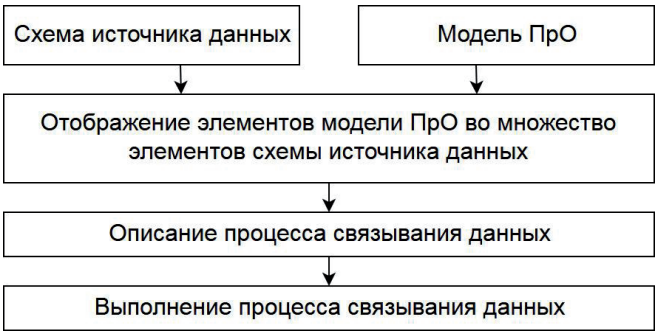

Рисунок 2 – Схема предложенного подхода к связыванию данных

3.2 Апробация подхода

Апробация подхода (*proof-of-concept*) имеет ограниченный функционал:

- работает с данными, которые хранятся в виде файлов в формате JSON;
- нет возможности подключения нескольких источников данных;
- нет возможности применения функций.

В качестве примера рассмотрено связывание данных о государственных контрактах (Федеральный закон о закупках 44-ФЗ). Данные по контрактам могут быть получены по адресу *https://clearspending.ru/opendata/* в разделе *Contracts/44 Federal Law*. Описание контракта включает данные по самому контракту, по заказчику (*customer*), поставщикам (*suppliers*) и поставляемым продуктам (*products*). Для описания контракта используются регистрационный номер (*regNum*) и дата публикации (*publishDate*). Для описания заказчика – имя (*fullName*), ИНН (*inn*), адрес (*postalAddress*). Для описания поставщика – имя (*organizationName*), ИНН (*inn*), адрес (*postAddress*). Для описания продукта – имя (*name*) и цена (*price*).

Реализация предложенного подхода связывания данных представлена на рисунке 3 (детализирует схему на рисунке 2).

3.2.1 Онтология абстрактной схемы данных

Онтология абстрактной схемы включает элементы, общие для любой схемы данных:

$$AbstractScheme = <Atom, Collection, Container>,$$

где:

- *Atom* используется для представления атомарного элемента данных (пара имя-значение);
- *Collection* – для представления коллекции (массива) однотипных элементов данных;
- *Container* – для представления множества разнотипных элементов.

Элементы онтологии позволят обращаться к различным элементам исходных данных с применением подходящего метода. Например, для коллекции таким методом будет перебор элементов.

Рисунок 3 – Схема реализация предложенного подхода к связыванию данных

3.2.2 Онтология схемы данных заданного формата

Онтология абстрактной JSON-схемы наследует элементы онтологии абстрактной схемы и фиксирует элементы, характерные именно для данного формата:

JsonAbstractScheme = <JSONDataElement, JSONArrayElement, JSONRecordElement>.

Элементы данной онтологии непосредственно используются для описания источника данных. Указание на используемый формат данных (т.е. JSON) позволяет на этапе реализации процесса отображения выбрать инструменты для работы именно с данным форматом (например, для чтения данных из файла).

3.2.3 Отология источника данных

Онтология источника данных описывает его структуру. На рисунке 4 представлен фрагмент онтологии источника данных для рассматриваемого примера. Все дальнейшие визуализации фрагментов онтологии выполнены с использованьем инструмента *Ontodia* (http://ontodia.org/).

Онтология источника данных строится по следующему принципу. Имена элементов источника данных используется в качестве имён соответствующих классов. Например, класс *product* представляет одноимённый элемент JSON-документа. Те же имена используются в качестве имён свойств, которые связывают родительские элементы с дочерними элементами. Например, класс *product* связан с классом *name* свойством *name*, что отражает вложенность JSON-элемента *name* в JSON-элемент *product*.

Описание источника данных играет вспомогательную роль и является вторичным по отношению к самим данным. Поэтому к способу записи такого описания можно предъявлять требования, руководствуясь единственно удобством использования описания для построения модели процесса связывания данных.

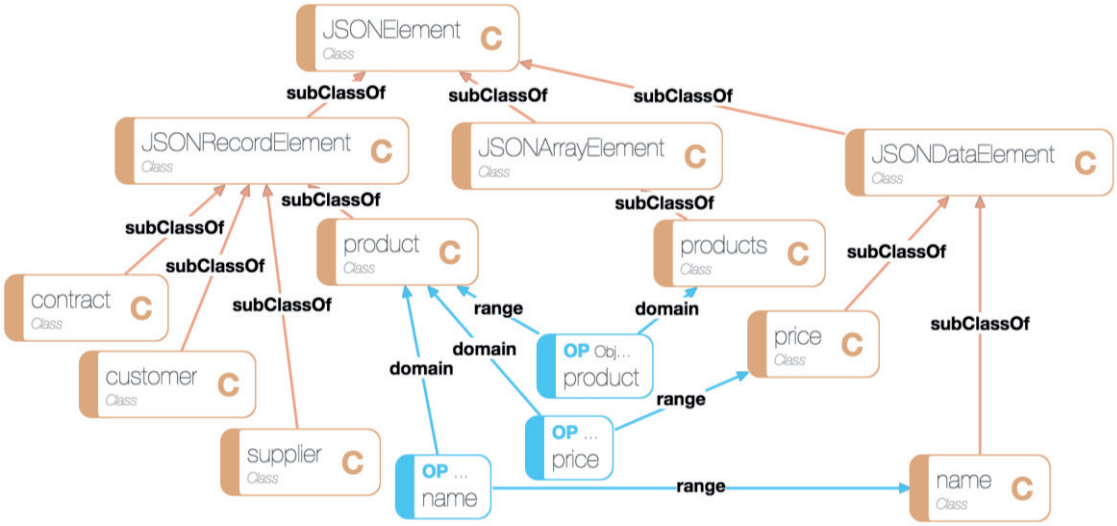

Рисунок 4 – Фрагмент онтологии источника данных

3.2.4 Онтология ПрО

Онтология ПрО фиксирует знание экспертов о некотором предмете. Онтология должна быть составлена, исходя из логики рассматриваемой ПрО. В рамках текущей реализации подхода единственным предъявляемым требованием к онтологии ПрО является указание области определения и области значения для всех свойств, которые должны быть включены в результаты отображения. Очевидно, что выполнение этого требования не предполагает понимания специфики процесса связывания данных. На рисунке 5 представлен фрагмент онтологии ПрО.

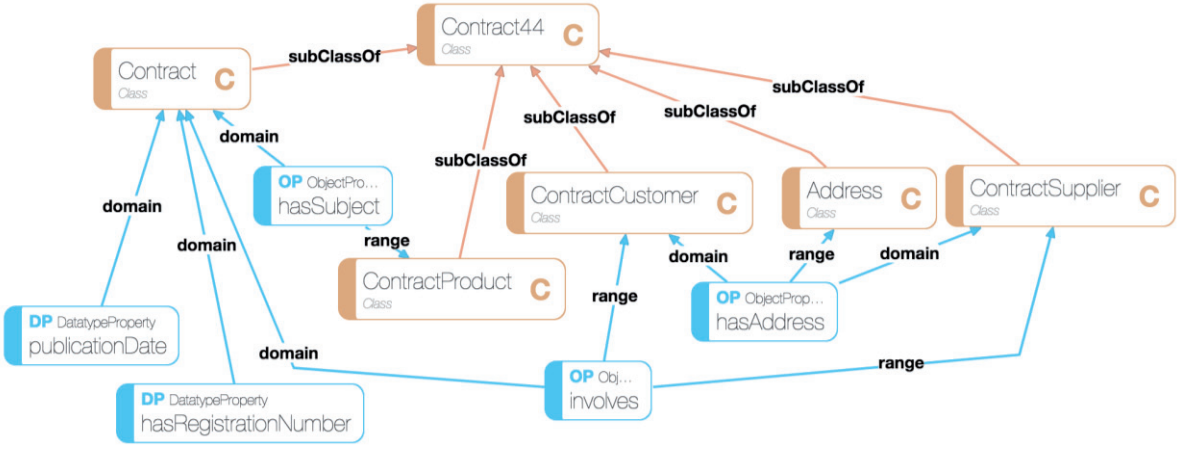

Рисунок 5 – Фрагмент онтологии ПрО

3.2.5 Онтология отображения

Онтология отображения включает только два элемента: *DataModel* используется в качестве родительского класса для всех классов источника данных, а *KnowledgeModel* использу-

ется в качестве родительского класса для всех классов ПрО. Наследование от перечисленных классов позволяет упростить разграничение элементов источника данных и ПрО на этапе работы с онтологией отображения.

3.2.6 Отображение элементов ПрО во множество элементов источника данных

Отображение элементов онтологии ПрО во множество элементов онтологии источника данных позволяет зафиксировать, какие элементы данных используются для порождения элементов онтологии ПрО. Отображения строятся пользователем. Результатом является онтология. Использование онтологии для фиксации конкретного отображения позволяет изолировать устанавливаемые отображения как от описания источника данных, так и от описания ПрО, и, соответственно, не ограничивает их повторное использование.

В текущей версии для установления отображения используются свойства *equivalProperty* и *equivalentClass*, которые являются частью спецификации языка OWL.

На рисунках 6 и 7 показаны примеры таких отображений. При соблюдении требования указания областей определения и значения свойств отображение на рисунке 6 позволяет:

1) однозначно определить, на основе какого элемента данных должен строиться экземпляр класса *ContractProduct*;
2) какой элемент данных использовать для заполнения свойства *hasCost;* для этого сопоставляются области определений и значений эквивалентных свойств.

Свойство *equivalentClass* позволяет разрешить неоднозначность, возникающую в том случае, когда область определения свойства ПрО включает множество классов. В таком случае явно указывается, какой класс онтологии источника данных соответствует классу онтологии ПрО.

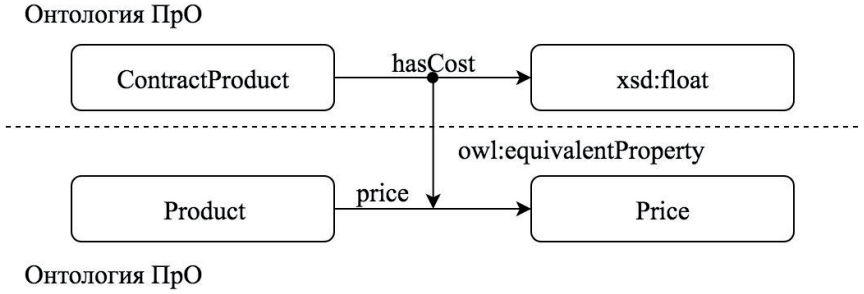

Рисунок 6 – Отображение на основе *equivalProperty*

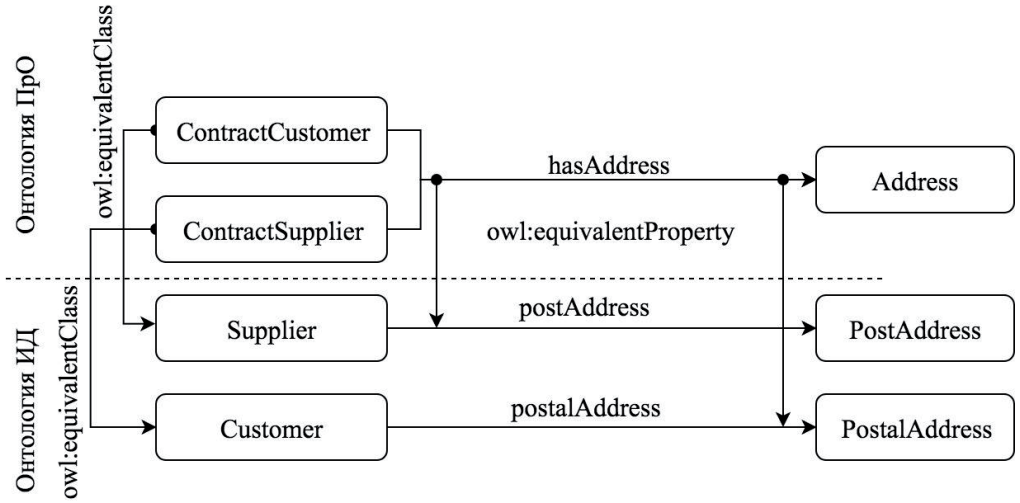

Рисунок 7 – Отображение на основе *equivalProperty* и *equivalentClass*

3.2.7 Онтология «Лабиринт». Этапы построения модели процесса связывания

Модель процесса связывания данных строится на основе онтологии отображения элементов ПрО во множество элементов источника данных. Для реализации построения модели процесса связывания создана онтология «Лабиринт». Данная онтология включает как элементы, из которых строится модель процесса связывания, так и правила построения этой модели. Модель процесса связывания основана на модели потока данных (см. рисунок 8).

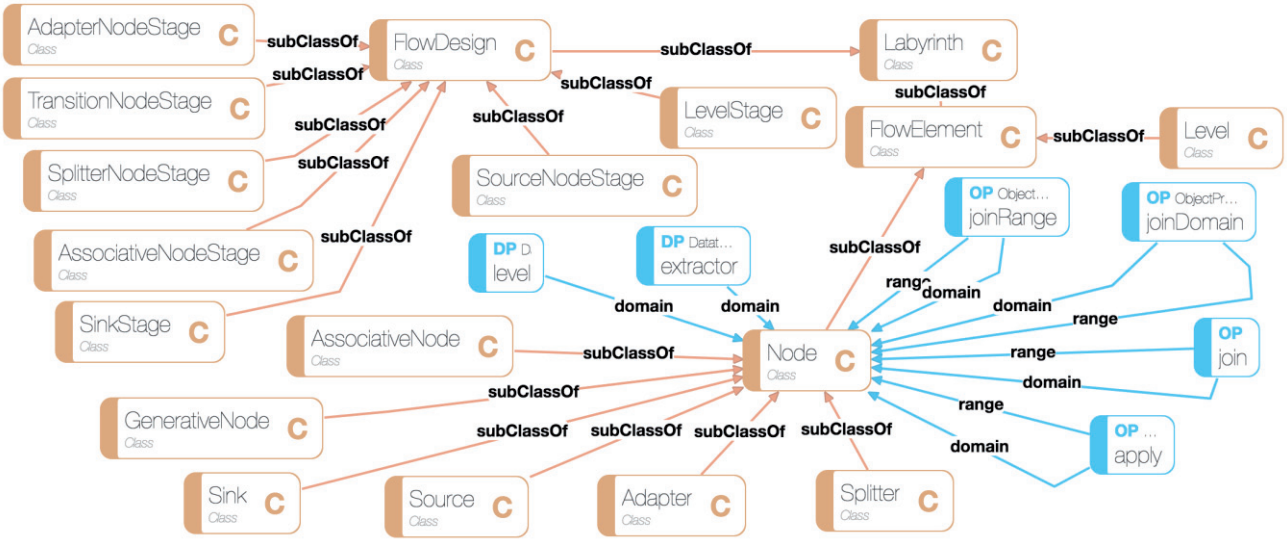

Рисунок 8 – Онтология «Лабиринт»: модель потока данных

Поток данных представляет собой граф, состоящий из узлов вычисления, которые сгруппированы по уровням. Каждый узел получает данные от узлов, лежащих на нижних уровнях. Узлы первого уровня получают данные из указанного источника данных. Онтология условно разделена на два подграфа: *FlowDesign* включает множество подклассов, представляющих этапы построения модели процесса связывания данных, а *FlowElement* включает подклассы, из экземпляров которых состоит модель процесса связывания. Основным классом модели процесса связывания данных является класс *Node* – узел потока данных, реализующий некоторую функцию. Узлы разделены на следующие типы:

- *Adapter*, *Splitter* – обеспечивают доступ к элементам источника данных: *Adapter* – это узел, который обеспечивает доступ к отдельным элементам источника данных, *Splitter* – к элементам массива;
- *GenerativeNode* и *AssociativeNode* – обеспечивают построение элементов ПрО: *GenerativeNode* отвечает за генерацию экземпляров классов, *AssociativeNode* – за генерацию отношений между экземплярами классов;
- *Source* обеспечивает доступ непосредственно к источнику данных;
- *Sink* обеспечивает запись результатов, полученных в отдельных узлах, в результирующую онтологию.

Каждый узел (т.е. экземпляр класса *Node*) описывается следующими свойствами:

- *объектные* свойства: *apply* указывает на узел, которому будут передаваться данные после обработки; *dataIn*/*dataOut* указывают на входные/выходные экземпляры классов онтологии источника данных; *join* указывает на узел типа *Sink*; *joinDomain*/*joinRange* указывают на узел типа *AssociativeNode*;
- *примитивные* свойства: *level* – номер уровня, к которому принадлежит узел; *extractor* – название элемента источника данных, который доступен для данного узла.

Таким образом, описываемая модель потока данных – это последовательность узлов, решающих следующие задачи:

1) чтение источника данных;
2) получение доступа к элементам иерархической структуры данных, полученной из источника данных;
3) порождение экземпляров классов ПрО на основе извлечённых данных;
4) связывание экземпляров классов отношениями;
5) запись полученных результатов.

Процесс связывания данных, описываемый предлагаемой моделью, можно представить как расщепление исходного источника данных на части и формирование на их основе множества потоков. Каждый поток несёт часть исходных данных. В соответствующих узлах эти данные преобразуются в элементы онтологии ПрО – экземпляры классов и отношения между экземплярами классов. В конце эти части собираются в единую структуру, формируя онтологическое описание ПрО.

3.2.8 Онтология «Лабиринт». Правила для построения модели процесса связывания данных

Классы подграфа *FlowDesign* (см. рисунок 8) представляют собой элементы системы правил, которая используется для автоматического построения модели процесса связывания данных на основе онтологии отображения. Для этого онтология «Лабиринт» импортируется в построенную пользователем онтологию отображения.

Правила заданы в виде SPARQL-запросов: *WHERE*-часть SPARQL-запроса – антецедент, *CONSTRUCT*-часть – консеквент. Запросы связаны с подклассами класса *FlowDesign* с помощью элементов языка SPIN[6]. TopBraid IDE[7] предоставляет удобный пользовательский интерфейс для построения и запуска таких правил.

На рисунке 9 представлен фрагмент интерфейса редактора и правило для построения уровней процесса связывания данных.

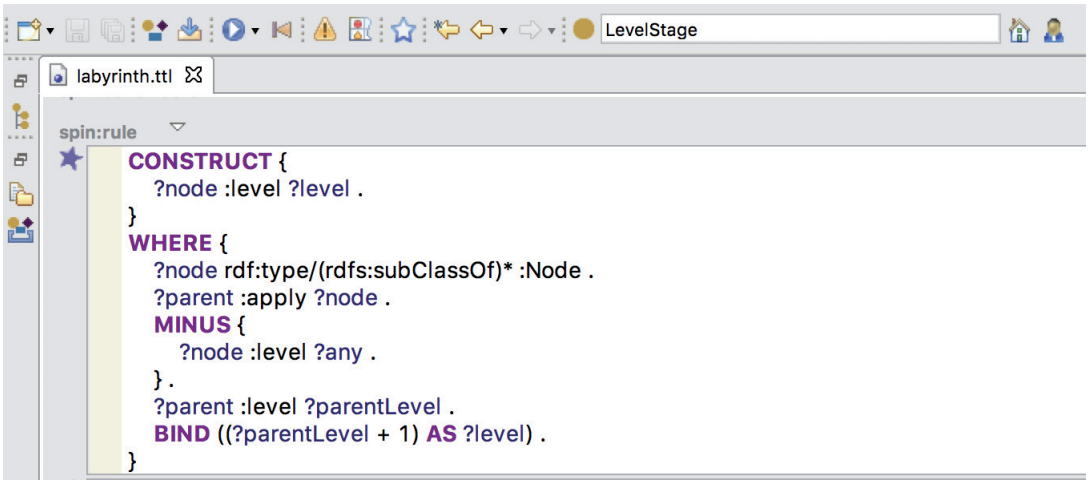

Рисунок 9 – Фрагмент интерфейса редактора с правилом построения уровней процесса связывания данных

О роли запроса каждого из подклассов *FlowDesign* (см. рисунок 8) можно судить по его названию. Например, класс *AssociativeNodeStage* отвечает за создание узлов, генерирующих отношения между двумя экземплярами.

[6] http://spinrdf.org/.
[7] https://www.topquadrant.com/tools/modeling-topbraid-composer-standard-edition/.

3.2.9 Модель процесса связывания данных

На рисунке 10 представлена онтология процесса связывания данных – результат выполнения правил-запросов для онтологии отображения рассматриваемого примера (модель развернута сверху-вниз).

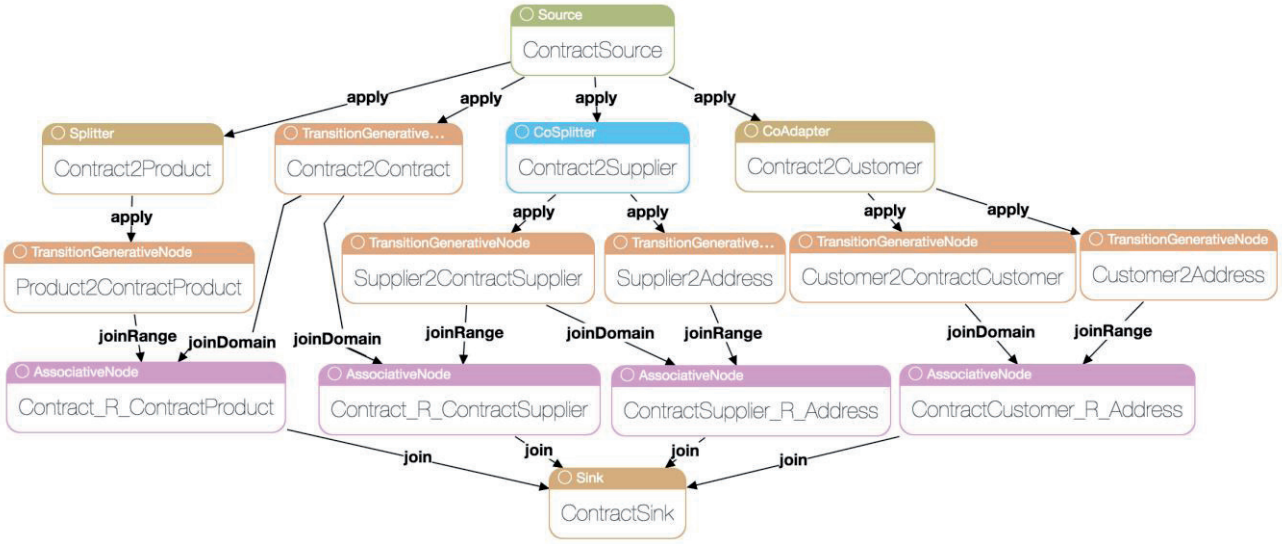

Рисунок 10 – Онтология процесса связывания данных

3.2.10 Генерация программного кода

На основе онтологии процесса связывания данных генерируется исполняемый код процесса. В качестве платформы для реализации выбрана библиотека *Apache Beam*[8]. Данную библиотеку можно рассматривать как своего рода общий интерфейс к различным реализациям высокопроизводительных вычислений на кластерах, таких как *Apache Spark*[9], *Google Cloud Dataflow*[10], *Apache Flink*[11] и др. *Apache Beam* на данный момент наиболее универсальное решение для реализации высокопроизводительных вычислений.

Процесс вычисления реализуется на языке программирования общего назначения *Kotlin*[12]. Для генерации кода по модели вычисления используется библиотека *KotlinPoet*[13]. Процесс генерации представляет собой итеративный процесс, состоящий из двух базовых действий: выполнение SPARQL-запроса к построенным онтологиям и генерация фрагмента программного кода на основе результатов запроса.

На рисунке 11 показана реализация процесса связывания на базе платформы *Google Cloud Dataflow*.

3.3 Оценка подхода

Предложенный подход сравнивался с инструментами RML-*Mapper* и SPARQL-*Generate*. Для приблизительной оценки того, как соотносится предлагаемое решение с иными решениями той же задачи, можно обратиться к [21].

[8] https://beam.apache.org/.
[9] https://spark.apache.org/.
[10] https://cloud.google.com/dataflow/.
[11] https://flink.apache.org/.
[12] https://kotlinlang.org/.
[13] https://github.com/square/kotlinpoet.

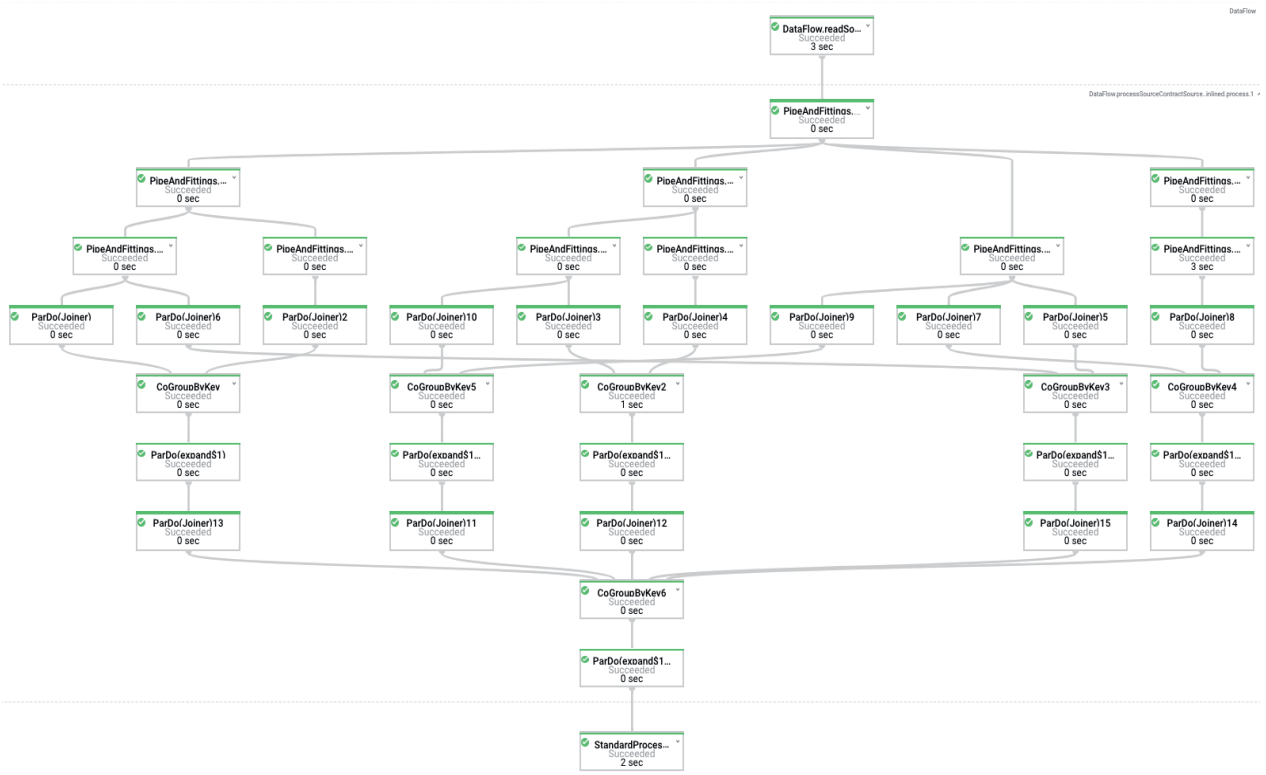

Рисунок 11 – Реализация процесса связывания данных на базе *Google Cloud Dataflow*

На рисунке 12 представлены результаты сравнения решений. Ось абсцисс фиксирует количество элементов в источнике данных. Под элементом понимается описание одного контракта. На оси ординат указано время выполнения запроса. DF (*Data Flow*) соответствует предлагаемому решению, запущенному на *Google Cloud Dataflow*; DF Elapsed Time – общее время обработки данных на базе *Google Cloud Dataflow*, включая время на построение кластера; DF Wall Time – процессорное время связывания данных; RML GCE (*Google Computation Engine*) Real Time – время связывания данных с помощью RML-*Mapper* на удалённой виртуальной машине; SG GCE Real Time – время связывания данных с помощью SPARQL-*Generate* на удалённой виртуальной машине.

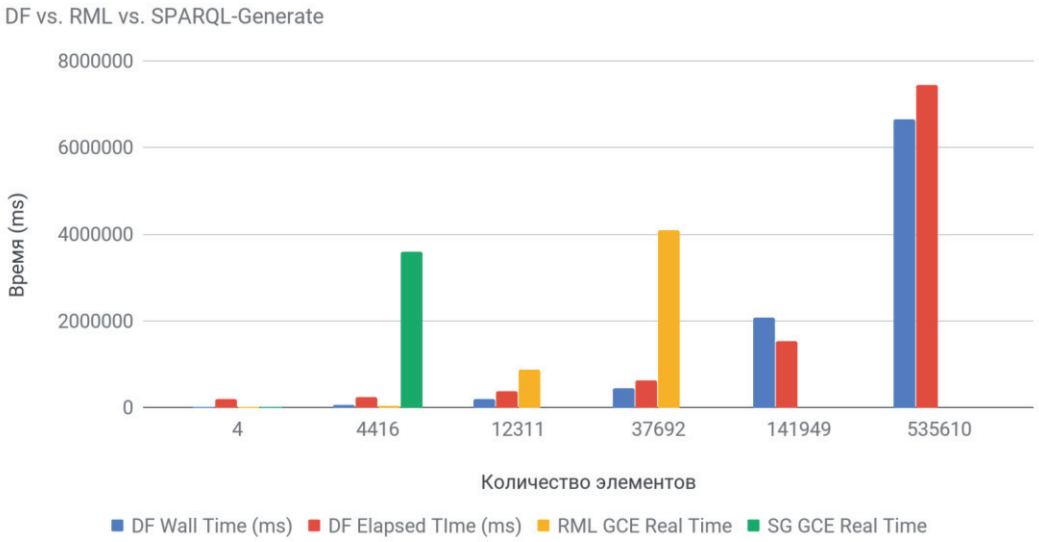

Рисунок 12 – Сравнение полученных результатов DF с RML-*Mapper* и SPARQL-*Generate*

Время для DF взято из пользовательского интерфейса, предоставляемого платформой *Google Cloud Dataflow*. Время для RML-*Mapper* и SPARQL-*Generate* измерялось с помощью команды *time* ОС Ubuntu. Команда выдаёт три значения времени – *real*, *user* и *sys*. В графике указано время *real*, т.е. время общей работы программы. Эквивалентным для него является время Wall Time.

Выполнение DF, кроме обработки 141949 элементов, было ограничено одной машиной типа *n*1-*standard*-1 (по номенклатуре *Google Cloud Platform*): 1 процессор и 3.75 GB ОЗУ. При обработке 141949 использовалось 2 машины типа *n*1-*standard*-1, что видно из графика: процессорное время больше общего времени, т.к. обработка шла параллельно. Данный запуск был проведён без ограничений, масштабирование производилось автоматически.

В качестве GCE использовалась виртуальная машина типа *n*1-*standard*-2 со следующими параметрами: 2 процессора и 7.5 GB ОЗУ.

Для 141949 и 535610 элементов не указано время преобразования для RML-*Mapper* и SPARQL-*Generate*, т.к. работа RML была прервана после нескольких часов ожидания, а SPARQL-*Generate* завершался с ошибкой из-за превышения доступной памяти.

По результатам сравнения можно сделать следующие выводы:

- при небольшом объёме данных предлагаемый подход проигрывает RML-*Mapper*, но при увеличении объёма превосходит его;
- реализация предложенного подхода на базе платформ высокопроизводительных вычислений позволяет воспользоваться реализованными в них средствами масштабирования вычислений (было продемонстрировано на наборе, состоящем из 141949 элементов).

4 Обсуждение

Простота рассмотренного сценария и функциональная ограниченность реализации не позволяют однозначно заявить о его преимуществах. Для полноценного сравнения решение должно реализовывать минимальный набор функциональных требований (см. первый раздел). Дополнительного обсуждения требует вопрос использования моделей для описания источника данных и процесса их связывания. Создание этих моделей требует определённых затрат, которые необходимо обосновать.

RML-*Mapper* и SPARQL-*Generate* используют специальные языки для фиксации отображений множества элементов источника данных во множество элементов ПрО: RML-*Mapper* использует язык RML [22], SPARQL-*Generate* [23] – расширение языка SPARQL. В указанных языках источник данных в явном виде не описывается. Для обращения к элементам источника данных используются строковые выражения. При этом если источник данных записан в формате JSON или XML, используются выражения на специальном языке. Например, для XML – это XPath. Такая строка не позволяет проверить тип хранимых данных или зафиксировать ограничения на возможные значения.

Напротив, явная онтологическая модель источника данных позволяет:

- документировать схему источника, что особенно полезно в тех случаях, когда элементы не имеют “дружественных” пользователю названий;
- добавлять ограничения (owl:Restrictions), например, на диапазон принимаемых элементов, и использовать эти знания для фильтрации неподходящих элементов данных;
- упростить повторное использование отображений (если известно, как источник данных связан с $ПрО_1$ и как $ПрО_1$ связана с $ПрО_2$, то можно построить $ПрО_2$ на основе указанного источника).

Таким образом, при долговременном системном использовании разнородных источников этот подход будет более эффективным при создании средств, упрощающих построение и отображение онтологий.

Заключение

В работе предложен подход для построения процесса связывания данных слабоструктурированных форматов в онтологическое представление, включающий:

- построение явных моделей источника данных и процесса преобразования;
- возможности автоматизации построения модели связывания данных и реализации процесса связывания данных на основе доступных масштабируемых высокопроизводительных платформ.

Развитие подхода должно быть направлено на более полное соответствие предложенным требованиям. Выполнение функциональных требований включает добавление возможности работы с несколькими источниками данных, возможности использовать функции для связывания данных, возможности работы с различными форматами. В части нефункциональных требований основная работа должна быть направлена на создание инструмента, упрощающего задачу построения онтологии источника данных и задачу построения отображения.

Список источников

[1] W3C Semantic Web Activity. – https://www.w3.org/2001/sw/.

[2] W3C Data Activity – Building the Web of Data. – https://www.w3.org/2013/data/.

[3] ***Ishida, R***. An Introduction to Multilingual Web Addresses / R. Ishida. - https://www.w3.org/International/articles/idn-and-iri/#problem.

[4] ***Berners-Lee, T.J.*** Linked Data – Design Issues / T.J. Berners-Lee. – https://www.w3.org/DesignIssues/LinkedData.

[5] The Linked Open Data Cloud. – https://lod-cloud.net/.

[6] Реестр наборов открытых данных. – https://data.gov.ru/opendata.

[7] ***Gonzalez, R.*** RDF vs. XML / R. Gonzalez. – https://www.cambridgesemantics.com/blog/semantic-university/learn-rdf/rdf-vs-xml/.

[8] Сведения о государственной регистрации юридических лиц, индивидуальных предпринимателей, крестьянских (фермерских) хозяйств. - https://egrul.nalog.ru/.

[9] Открытые данные Федеральной налоговой службы Российской Федерации. - https://www.nalog.ru/opendata/.

[10] Clearspending: Open data. - https://clearspending.ru/opendata/.

[11] ***Dimou, A.*** What factors influence the design of a linked data generation algorithm? / A. Dimou, P. Heyvaert, B. De Meester, R. Verborgh // Proceedings of the 11th Workshop on Linked Data on the Web, LDOW (23 April, 2018, Lyon, France) – CEUR-WS.org, 2018. – 6 p.

[12] ***Dimou, A.*** RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data / A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, R. de Walle // Proceedings of the 7th Workshop on Linked Data on the Web, LDOW2014 (08 April, 2014, Seoul, Korea). – CEUR-WS.org, 2014. – Vol. 1184. – 5 p.

[13] ***Lefrançois, M.*** A SPARQL Extension for Generating RDF from Heterogeneous Formats / M. Lefrançois, A. Zimmermann, N. Bakerally // 14th International Conference, ESWC (28 May – 1 June, 2017, Portorož, Slovenia) Springer, 2017. – P. 35–50.

[14] ***Morrison, J.P.*** Flow-Based Programming: A New Approach to Application Development / J.P. Morrison □2nd ed. – Charleston: Createspace Independent, 2010. – 370 p.

[15] ***Vassiliadis, P.*** A survey of extract–transform–load technology / P. Vassiliadis // International Journal of Data Warehousing and Mining (IJDWM). – Clayton: IGI Global, 2009. – Vol. 5, № 3. – P. 1–27.

[16] ***Knap, T.*** UnifiedViews: An ETL Framework for Sustainable RDF Data Processing / T. Knap, M. Kukhar, B. Macháč, P. Škoda, J. Tomeš, J. Vojt // European Semantic Web Conference, ESWC (25-29 May, 2014, Anissaras, Crete, Greece) – Cham: Springer, Cham, 2014. – Vol. 475. – P. 379–383.

[17] ***Klímek, J.*** LinkedPipes ETL in use / J. Klímek , P. Škoda // Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services, iiWAS '17 (4-6 December, Salzburg, Austria). – ACM Press, 2017. – P. 441–445.

[18] ***Gupta,* S.** Karma: A System for Mapping Structured Sources into the Semantic Web / S. Gupta, P. Szekely, A.C. Knoblock, A. Goel, M. Taheriyan, M. Muslea // Extended Semantic Web Conference, ESWC (27-31 May, 2012, Heraklion, Crete, Greece). – Springer, 2015. – P. 430–434.
[19] ***Dimou, A***. Extending R2RML to a Source-independent Mapping Language for RDF / A. Dimou, M. Vander Sande, P. Colpaert, E. Mannens, R. de Walle // International Semantic Web Conference, Posters & Demos (21-25 October 2013, Sydney, Australia). – CEUR-WS.org, 2013 – Vol. 1035. – P. 237–240.
[20] ***Lefrançois, M.*** Flexible RDF Generation from RDF and Heterogeneous Data Sources with SPARQL-Generate / M. Lefrançois, A. Zimmermann, N. Bakerally // European Knowledge Acquisition Workshop, EKAW (19–23 November, 2016, Bologna, Italy). – Springer, 2016. – P. 131–135.
[21] ***Kolchin, M.*** A practical review of non-RDF to RDF converters / M. Kolchin. – https://medium.com/datafabric/a-practical-review-of-non-rdf-to-rdf-converters-51686338927f.
[22] ***Dimou, A.*** RDF Mapping Language. Unofficial Draft / A. Dimou A., M. Vander Sande. – http://rml.io/spec.html.
[23] SPARQL-Generate - Language Overview. – https://ci.mines-stetienne.fr/sparql-generate/language.html.

MODEL-DRIVEN TRANSFORMATION OF HETEROGENEOUS SOURCES INTO LINKED DATA

S.V. Lebedev

Saint Petersburg Electrotechnical University "LETI" and DataFabric Ltd., Saint Petersburg, Russia
lebedev.sv.etu@gmail.com

Abstract

There are a lot of data sources that can be used to make a reasonable decision in the diversity of fields. But the linking of data from these sources into a solid view must come first. To represent linked data RDF language is used. Unfortunately, there are lots of those who publish data in non-RDF formats. Obviously, considering current data volumes integration task cannot be done in analytics' heads. A number of instruments to link heterogeneous data were proposed. The most flexible ones are based on special mapping languages. However, these solutions lack scaling and may not be effective for processing large data. In the presented paper, an approach for a possible solution is proposed. The idea of the approach is to abstract data linking process model separating it from a mapping as well as from a code implementation. The model is automatically generated based on a mapping of a set of source ontology elements into a set of domain ontology elements. The model of a process becomes available for separate manipulations relatively independent of other activities. The model can be used to optimize or to custom the linking process or can be treated as a specification for different implementations. As it based on the data-flow concepts it can be naturally translated into contemporary high-performance computation concepts. In other words, it is demonstrated how a process model can be implemented on one of such platforms. The obtained results make it reasonable to continue investigating and developing the proposed approach.

Key words: *linked data, linking process model, automation of model generation, program code generation, scalable solutions.*

Citation: *Lebedev SV.* Model-driven transformation of heterogeneous sources into linked data [In Russian]. *Ontology of designing*. 2019. 9(1): 101-116. – DOI: 10.18287/2223-9537-2019-9-1-101-116.

References

[1] W3C Semantic Web Activity. - https://www.w3.org/2001/sw/.
[2] W3C Data Activity – Building the Web of Data. - https://www.w3.org/2013/data/.
[3] ***Ishida R***. An Introduction to Multilingual Web Addresses. - https://www.w3.org/International/articles/idn-and-iri/#problem.
[4] ***Berners-Lee TJ.*** Linked Data – Design. - https://www.w3.org/DesignIssues/LinkedData.
[5] The Linked Open Data Cloud. - https://lod-cloud.net/.
[6] The register of sets of the open data [In Russian]. - https://data.gov.ru/opendata.

[7] ***Gonzalez R.*** RDF vs. XML. - https://www.cambridgesemantics.com/blog/semantic-university/learn-rdf/rdf-vs-xml/.
[8] Information about the state registration of legal entities, individual entrepreneurs, farms [In Russian]. - https://egrul.nalog.ru/.
[9] Open data of the Federal Tax Service of the Russian Federation [In Russian]. - https://www.nalog.ru/opendata/.
[10] Clearspending: Open data. - https://clearspending.ru/opendata/.
[11] ***Dimou A, Heyvaert P, De Meester B, Verborgh R.*** What factors influence the design of a linked data generation algorithm? Proceedings of the 11th Workshop on Linked Data on the Web, LDOW (23 April, 2018, Lyon, France). CEUR-WS.org; 2018: 6.
[12] ***Dimou A, Vander Sande M, Colpaert P, Verborgh R, Mannens E, de Walle R.*** RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. Proceedings of the 7th Workshop on Linked Data on the Web, LDOW2014 (08 April, 2014, Seoul, Korea). CEUR-WS.org; 2014; 1184: 5.
[13] ***Lefrançois M, Zimmermann A, Bakerally N.*** A SPARQL Extension for Generating RDF from Heterogeneous Formats. 14th International Conference, ESWC (28 May–1 June, 2017, Portorož, Slovenia). Springer; 2017: 35–50.
[14] ***Morrison JP.*** Flow-Based Programming: A New Approach To Application Development, 2nd ed. Createspace Independent; 2010.
[15] ***Vassiliadis P.*** A survey of extract–transform–load technology. International Journal of Data Warehousing and Mining (IJDWM). IGI Global; 2009; 5(3): 1–27.
[16] ***Knap T, Kukhar M, Macháč B, Škoda P, Tomeš J, Vojt J.*** Unified Views: An ETL Framework for Sustainable RDF Data Processing. European Semantic Web Conference, ESWC (25-29 May, 2014, Anissaras, Crete, Greece). Springer, Cham; 2014; 475: 379–383.
[17] ***Klímek J, Škoda P.*** Linked Pipes ETL in use. Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services, iiWAS '17 (4-6 December, Salzburg, Austria). ACM Press; 2017: 441–445.
[18] ***Gupta S, Szekely P, Knoblock AC, Goel A, Taheriyan M, Muslea M.*** Karma: A System for Mapping Structured Sources into the Semantic Web. Extended Semantic Web Conference, ESWC (27-31 May, 2012, Heraklion, Crete, Greece). Springer; 2015: 430–434.
[19] ***Dimou A, Vander Sande M, Colpaert P, Mannens E, de Walle R.*** Extending R2RML to a Source-independent Mapping Language for RDF. International Semantic Web Conference, Posters & Demos (21-25 October 2013, Sydney, Australia). CEUR-WS.org; 2013; 1035: 237–240.
[20] ***Lefrançois M, Zimmermann A, Bakerally N.*** Flexible RDF Generation from RDF and Heterogeneous Data Sources with SPARQL-Generate. European Knowledge Acquisition Workshop, EKAW (19–23 November, 2016, Bologna, Italy). Springer; 2016: 131–135.
[21] ***Kolchin M.*** A practical review of non-RDF to RDF converters. - https://medium.com/datafabric/a-practical-review-of-non-rdf-to-rdf-converters-51686338927f.
[22] ***Dimou A, Vander Sande M.*** RDF Mapping Language. Unofficial Draft. - http://rml.io/spec.html.
[23] SPARQL-Generate - Language Overview. - https://ci.mines-stetienne.fr/sparql-generate/language.html.

Сведения об авторе

Лебедев Сергей Вячеславович, 1984 г. рождения. Окончил Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина) (СПбГЭТУ ЛЭТИ) в 2010 г. Ассистент кафедры вычислительной техники СПбГЭТУ ЛЭТИ, программист в ООО «Датафабрик». В списке научных трудов более 10 работ в области многоагентных систем и технологий семантического Web.
Sergey Vyacheslavovich Lebedev (b. 1984) graduated from the Saint Petersburg Electrotechnical University "LETI" (ETU) in 2010. He is an assistant at ETU (Department of Computer Science and Engineering) and a programmer in DataFabric LLC. He is co-author of more than 10 publications in the field of multi-agent systems and semantic web technologies.