УДК 004.89:004.4
DOI 10.25205/1818-7900-2019-17-2-138-152

Платформа комплексирования и тестирования средств трансформации естественно-языковых запросов в SPARQL-запросы

И. С. Постаногов, **И. А. Турова**

Пермский государственный национальный исследовательский университет
Пермь, Россия

Аннотация

Статья посвящена вопросу поддержки процесса создания средства трансформации естественно-языковых (ЕЯ) запросов в SPARQL-запросы (далее – средство трансформации).

Во введении описаны актуальность задачи понимания ЕЯ-запросов информационными системами, а также преимущества использования онтологий как средства представления знаний для решения данной задачи. Обозначена роль средств трансформации в системах с ЕЯ-запросами к реляционным базам данных. На основании анализа проблем, связанных с комплексированием и тестированием существующих средств трансформации, а также с целью поддержки создания и тестирования собственных модулей трансформации предложена концепция программной платформы, упрощающей эти задачи. Архитектура платформы удовлетворяет требованиям простоты подключения сторонних средств трансформации, переиспользования отдельных модулей, а также интеграции результирующих средств трансформации в другие системы, в том числе в системы тестирования.

Строительными блоками создаваемых систем трансформации служат отдельные модули трансформации, упакованные в Docker-контейнеры. Программный доступ к каждому модулю осуществляется при помощи gRPC. Загруженные в платформу модули могут быть выстроены в конвейер трансформации автоматически или вручную при помощи встроенного стороннего редактора диаграмм потоков данных SciVi. Совместимость отдельных модулей контролируется при помощи автоматического анализа программных интерфейсов. На основании спецификации конвейера в формате, поддерживаемом SciVi, а также gRPC-спецификаций отдельных модулей комплексируется единое многоконтейнерное приложение, которое может быть интегрировано в другие системы, а также протестировано на пополняемом наборе тестов. Ожидаемые и фактические результаты трансформации запроса доступны для просмотра в графическом виде в разработанном ранее средстве визуализации.

Ключевые слова

интеллектуальные информационные системы, онтология, естественно-языковой интерфейс, SPARQL, доступ к данным на базе онтологий, Docker, gRPC, Protocol Buffers, диаграмма потока данных

Для цитирования

Постаногов И. С., *Турова И. А.* Платформа комплексирования и тестирования средств трансформации естественно-языковых запросов в SPARQL-запросы // Вестник НГУ. Серия: Информационные технологии. 2019. Т. 17, № 2. С. 138–152. DOI 10.25205/1818-7900-2019-17-2-138-152

Platform for Integrating and Testing Tools which Transform Natural Language Queries into SPARQL Queries

I. S. Postanogov, **I. A. Turova**

Perm State University
Perm, Russian Federation

Abstract

In the paper we discuss how to support the process of creating tools which transform natural language (NL) queries into SPARQL queries (hereinafter referred to as a transformation tool).

© И. С. Постаногов, И. А. Турова, 2019

In the introduction, we describe the relevance of the task of understanding natural language queries by information systems, as well as the advantages of using ontologies as a means of representing knowledge for solving this problem. This ontology-based data access approach can be also used in systems which provide natural language interface to databases. Based on the analysis of problems related to the integration and testing of existing transformation tools, as well as to support the creation and testing own transformation modules, the concept of a software platform that simplifies these tasks is proposed. The platform architecture satisfies the requirements for ease of connecting third-party transformation tools, reusing individual modules, as well as integrating the resulting transformation tools into other systems, including testing systems.

The building blocks of the created transformation systems are the individual transformation modules packaged in Docker containers. Program access to each module is carried out using gRPC. Modules loaded into the platform can be built into the transformation pipeline automatically or manually using the built-in third party SciVi data flow diagram editor. Compatibility of individual modules is controlled by automatic analysis of application programming interfaces. The resulting pipeline is combined according to specified data flow into a single multi-container application that can be integrated into other systems, as well as tested on extendable test suites. The expected and actual results of the query transformation are available for viewing in graphical form in the visualization tool developed earlier.

Keywords

intelligent information systems, ontology, natural language interface, SPARQL, ontology-based data access, Docker, gRPC, Protocol Buffers, data flow diagram

For citation

Postanogov I. S., Turova I. A. Platform for Integrating and Testing Tools which Transform Natural Language Queries into SPARQL Queries. *Vestnik NSU. Series: Information Technologies*, 2019, vol. 17, no. 2, p. 138–152. (in Russ.) DOI 10.25205/1818-7900-2019-17-2-138-152

Введение

В современном мире все чаще используются естественно-языковые (ЕЯ) интерфейсы для взаимодействия с различными информационными системами. Примером этого может служить разговор на естественном языке с роботом-промоутером на выставке или в торговом центре, где любой человек вне зависимости от знаний в области информационных технологий может задать вопрос о товаре, услуге, получить ответ и, возможно, сразу же оформить новый заказ. Те же подходы к человеко-машинному взаимодействию активно применяются и в виртуальных ассистентах, работающих не только в мобильных устройствах (Яндекс Алиса, Google Ассистент, Apple Siri), но и в умных колонках, являющихся центральными узлами систем домашней автоматизации (Яндекс Станция, Google Home, Apple HomePod). Взаимодействие с такими системами может быть не только устное, но и письменное при помощи приложений обмена мгновенными сообщениями.

Для внутренних нужд компаний корпоративного сектора естественно-языковой интерфейс используется при общении с чат-ботами в корпоративных мессенджерах. Традиционным способом реализации таких чат-ботов является сопоставление текста запроса с образцом и выдача результата, полученного предопределенным для сопоставленного шаблона способом. Для предоставления возможности задания более сложных, нерегламентированных запросов следует использовать знания, которые бы определяли процесс вывода ответа на запрос.

В качестве средства представления знаний можно использовать онтологии. Онтология – это точная спецификация концептуализации [1], ключевыми элементами которой являются понятия и связи между ними. К настоящему моменту существуют стандарт OWL для записи онтологий в машиночитаемом виде, средства программной обработки и вывода на онтологиях. Активно развивается исследовательское направление организации доступа к данным, основанного на онтологиях (Ontology-based Data Access, OBDA) [2]. Существенная часть исследований в области OBDA посвящена предоставлению виртуального доступа к данным, хранящимся в реляционных базах данных, т. е. трансформации запросов на языке SPARQL в запросы на языке SQL.

Ранее в работе [3] была предложена концепция обогащения унаследованных информационных систем сервисом запросов на естественном языке *Reply*. Основными идеями концепции были двухэтапная трансформация ЕЯ-запроса в SQL-запрос с использованием промежуточного представления в виде SPARQL-запроса, а также автоматизация конфигурирования как используемых OBDA-фреймворков, так и всей системы в целом.

Reply разрабатывался как сервис для трансформации традиционных информационных систем в интеллектуальные системы с запросами на естественном языке безотносительно предметной области исходной информационной системы, а также к целевому языку запросов (русский, английский…). В то же время для получения более качественного и полного ответа на запросы пользователей в каждом конкретном случае следует предусмотреть возможность выбора оптимального средства трансформации ЕЯ-запроса в SPARQL-запрос либо сборки нового средства на основе существующих модулей. В данной статье описана разработанная платформа комплексирования и тестирования средств трансформации ЕЯ-запросов в SPARQL-запросы для поддержки этого процесса. Под комплексированием в работе понимается процесс объединения системных элементов для производства полной системы [1].

Платформы создания средств трансформации ЕЯ-запросов в SPARQL-запросы

В настоящем разделе описываются экосистема, современные подходы и существующие реализации средств трансформации.

Обзор архитектур средств трансформации

Идея, положенная в основу вопросно-ответной системы AskNow [4], заключается в использовании в качестве промежуточного представления особой синтаксической структуры, названной авторами нормализованной структурой запроса (Normalized Query Structure – NQS). На первом этапе исходный ЕЯ-запрос должен быть преобразован в соответствующую ему NQS. Для этого с помощью модуля Query Processor запрос разбивается на отдельные токены, которые впоследствии помечаются тегами частей речи (part-of-speech tag, POS-тег). Модуль Auxiliary Relation Handler выделяет вспомогательное отношение – отношение, которое содержит лексические формы выражений is, is kind of, much, might be, does. Модуль Token Merger объединяет токены в сегменты исходя из их POS-тегов и положения в запросе. Выделенные сегменты проходят через модуль NQS Instance Generator, который строит итоговую NQS по определенным правилам. После этого построение SPARQL-запроса по нормализованной структуре запроса происходит также в несколько этапов. Модуль NQS Parser определяет тип запроса по его NQS и сопоставляет с сегментами запроса наиболее подходящие элементы онтологии (DBpedia) с помощью сервиса DBpedia Spotlight. Для решения проблем с отображением сегментов на онтологию используется электронный тезаурус WordNet. В нем осуществляется поиск синонимов для неопределенных сегментов. Используя результаты предыдущих этапов, модуль SPARQL Generator преобразует NQS в итоговый SPARQL-запрос.

Работа вопросно-ответной системы Sina [5] разделяется на четыре этапа. На первом этапе производится предобработка запроса: выделяются токены в виде отдельных ключевых слов, удаляются стоп-слова, выполняется лемматизация. На втором этапе ключевые слова различным образом объединяются в сегменты (N-граммы), каждый сегмент проверяется на возможность соответствия некоторому элементу онтологии. Далее на третьем этапе для всех подходящих сегментов ведется поиск соответствующего ресурса в онтологии по свойству rdfs:label. На этапе устранения неоднозначностей (disambiguation) выбирается подграф

[1] ГОСТ Р ИСО/МЭК 12207-2010. Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств. М.: Стандартинформ, 2011. 106 с.

из базы знаний, наиболее соответствующий исходному запросу. Далее по полученному подграфу строится SPARQL-запрос.

Вопросно-ответная система Xser [6] представляет собой конвейер из двух этапов: выделение из запроса возможных семантических структур – экземпляров, классов, отношений, и отображение выделенных структур на элементы онтологии. На первом этапе происходит разметка исходного запроса с использованием меток «начало фразы-отношения», «продолжение фразы-экземпляра» и т. п. По полученной разметке строится ациклический ориентированный граф зависимостей фраз. На втором этапе полученный граф преобразуется в запрос к онтологии. Так как с каждой фразой, являющейся элементом графа зависимостей, может быть сопоставлено более одного элемента онтологии, все возможные отображения ранжируются, и выбирается наилучшее. В качестве метрики выступает произведение нормированных результатов функции похожести для каждого слова, полученных с помощью Freebase Search API (для запросов к базе знаний Freebase).

Хотя в описанных системах используются разные методы и средства для преобразования ЕЯ-запросов в SPARQL-запросы, в процессе трансформации можно выделить одинаковые этапы [7]. Этап анализа естественно-языкового запроса может включать определение типа запроса, выделение именованных сущностей, определение зависимостей между частями предложения, построение синтаксического дерева запроса и т. д. На этапе отображения сегментов запроса на элементы онтологии происходит поиск подходящих определенной части запроса экземпляров, классов и отношений онтологии. Так как из-за наличия неоднозначностей в естественном языке (многозначность, омонимия и т. п.) с одним сегментом запроса могли быть сопоставлены несколько ресурсов онтологии, в каждом случае необходимо определить наиболее подходящий к контексту ресурс. Эти проблемы решаются на этапе устранения неоднозначностей. На этапе построения SPARQL-запроса определяется необходимый тип запроса (SELECT, ASK), строится основной графовый шаблон запроса.

Обзор существующих платформ создания средств трансформации

Возможность выделения отдельных этапов при решении задач обработки естественного языка привела к появлению универсальных платформ по анализу и преобразованию текстов. Примерами таких платформ служат NLTK [8] и TAISim [9]. Кроме того, существуют платформы, позволяющие конструировать программные средства, решающие более частные задачи данной области знаний, в том числе задачу трансформации ЕЯ-запроса в SPARQL-запрос.

Qanary [10] – система для комплексирования средств трансформации ЕЯ-запросов из отдельных компонентов. Согласно методологии разработки вопросно-ответных систем, предложенной авторами Qanary, компоненты системы должны взаимодействовать посредством аннотаций, определенных в словаре Qanary, построенном на основе модели аннотаций WADM. Qanary Pipeline – центральный компонент системы – представляет собой сервер, к которому подключаются остальные компоненты посредством RESTful интерфейса с помощью фреймворка Spring. Qanary Pipeline имеет веб-интерфейс для выбора компонентов для текущего конвейера, определения их порядка и запуска конвейера с некоторым исходным запросом. Пользовательские компоненты также должны быть реализованы в виде сервисов. Разработчики Qanary предоставляют шаблон Maven-приложения на Java для создания компонента. Шаблон включает код для запуска Spring-сервиса и пустую реализацию Qanary-компонента. При запуске сервиса отправляется запрос на регистрацию компонента в системе. При вызове компонента из Qanary Pipeline он получает входные данные и сохраняет результат своей работы с помощью извлечения и добавления аннотаций в служебную онтологию посредством выполнения SPARQL-запросов. При необходимости использования других технологий разработчик может вызвать свою реализацию из Java-компонента. Собранная с помощью Qanary-фреймворка система может быть интегрирована в другие системы как Spring-сервис.

В работе [11] представлен фреймворк QALL-ME, предоставляющий переиспользуемую архитектуру для создания мультиязычных вопросно-ответных систем, учитывающих пространственно-временной контекст запроса. Авторы декомпозируют процесс трансформации на этапы и сопоставляют с каждым этапом отдельный тип компонентов, например аннотаторы именованных сущностей для различных языков. В QALL-ME компоненты разделяются на три типа: языкозависимые, пространственно-зависимые и общесистемные компоненты, реализация которых одинакова для всех языков и местоположений. Фреймворк поддерживает сервис-ориентированную архитектуру: компоненты системы реализуются в виде веб-сервисов, а их программный интерфейс описывается на языке WSDL (Web Service Description Language). QALL-ME не является средством комплексирования средств трансформации. Цель создания фреймворка – упрощение разработки мультиязычных вопросно-ответных систем. В частности, реализация содержит методы для вызова компонента определенного типа, подходящего для языковых и пространственных характеристик запроса. Архитектура средства трансформации должна быть определена заранее, и последовательный вызов компонент должен быть запрограммирован. QALL-ME позволяет расширять возможности системы посредством написания компонентов для других языков и местоположений и регистрации пользовательских компонентов в QAPlanner (ядре системы) с помощью отправки запросов. Так как компоненты являются веб-сервисами, они могут быть реализованы с использованием любых языков программирования и технологий.

openQA [12] – расширяемый вопросно-ответный фреймворк. Он нацелен на улучшение качества ответов на естественно-языковые запросы путем комбинирования результатов работы нескольких систем. Обработка пользовательского запроса происходит в четыре этапа: интерпретация запроса в запрос на формальном языке, получение результатов запроса, синтез и представление результатов. На первом этапе ЕЯ-запрос обрабатывается всеми активными модулями интерпретации, например, в качестве таких модулей в openQA присутствуют реализации вопросно-ответных систем Sina и TBSL. Далее из источника извлекаются результаты выполнения полученных запросов на формальных языках (SPARQL, SQL). На третьем этапе результаты ранжируются по релевантности, на четвертом – представляются пользователю. К openQA можно подключить пользовательские модули для каждого этапа, однако они должны быть реализованы в виде Java-плагинов. Фреймворк не поддерживает возможность комплексирования модуля интерпретации из нескольких компонент. Собранная система представляет собой сервис и доступна через RESTful интерфейс.

Сравнение существующих платформ создания средств трансформации

Основными критериями для сравнения существующих платформ были выбраны:

1) предъявляемые требования к степени связывания как непосредственно между модулями, так и между модулями и платформой;

2) наличие средств поддержки процесса развертывания создаваемых систем.

Основной принцип *слабого* связывания состоит в уменьшении числа допущений, которые делают друг о друге взаимодействующие стороны [13]. Возможность слабого связывания трактуется как преимущество платформы, поскольку упрощается процесс подмены реализации, тем самым ускоряется поиск наилучшей комбинации модулей средства трансформации.

Среди рассмотренных выше существующих платформ построения средств трансформации системой с сильным связыванием является система openQA, требующая оформления отдельных модулей системы в виде Java-плагинов, тем самым ограничивающая возможности для использования прочих языков программирования. Даже при наличии возможности написания программной обертки на языке Java, разработчику требуется самостоятельно организовывать коммуникацию между программами на языке Java и языке программирования ядра модуля трансформации.

Qanary позволяет создавать системы на принципах слабого связывания, однако коммуникация между отдельными модулями осуществляется при помощи модификации глобального состояния – множества аннотаций онтологии. Для модулей отсутствует формальное описание того, какие именно категории аннотаций будут добавлены, удалены, изменены в процессе работы. Корректность создаваемого конвейера в случае зависимости одного модуля от результатов работы другого модуля в данной платформе может быть проверена только на этапе исполнения.

QALL-ME также позволяет создавать системы на принципах слабого связывания, причем API каждого из сервисов описывается на языке WSDL. Наличие формального описания позволяет генерировать клиентские и / или серверные части взаимодействующих компонентов. Недостатком данного подхода является выбор технологий веб-сервисов для взаимодействия компонент. У данной технологии есть проблемы с производительностью, объясняемые использованием XML и HTTP 1.1 для передачи сообщений. Кроме того, в QALL-ME существенно ограничены типы модулей, для которых можно предоставить собственную реализацию.

Рассмотренные платформы не ориентированы на поддержку процесса развертывания итогового средства трансформации с учетом требований к среде исполнения отдельных модулей, а также с учетом потенциальной противоречивости этих требований. Например, несколько модулей могут зависеть от разных версий одной программной библиотеки, которая должна располагаться по одному и тому же предопределенному системному пути. В таком случае без дополнительного вмешательства на этапе сборки или запуска средства трансформации использование комбинации этих модулей будет невозможно.

Тестирование средств трансформации

Для сравнения средств трансформации необходимо производить их тестирование. На данный момент существуют тестовые наборы запросов и соответствующих эталонных SPARQL-запросов к различным онтологиям. Большинство тестовых наборов [14; 15] содержат запросы к обширным базам знаний общего назначения, таким как DBpedia, Freebase, YAGO. Например, с 2011 г. в рамках серии соревнований QALD [16] для оценки вопросно-ответных систем организаторами было разработано более десяти тестовых наборов запросов к DBpedia, среди которых помимо англоязычных наборов присутствует несколько мультиязычных. Тестовые наборы составляются вручную группой людей и включают в себя запросы различных типов и сложности, поэтому с их помощью можно провести качественную оценку вопросно-ответных систем.

Наличие в платформе комплексирования средств трансформации возможности проверки собранных систем на существующих тестовых наборах запросов, а также наглядного просмотра результатов таких тестов существенно упростит процесс разработки.

Архитектура предлагаемой платформы

На основании анализа недостатков описанных ранее программных средств было принято решение о разработке собственной платформы комплексирования и тестирования средств трансформации.

Основные требования к платформе:

1) минимальность предъявляемых ограничений на технологии, используемые модулями трансформации;

2) проверка совместимости интерфейсов модулей на этапе комплексирования;

3) эффективность технологий межмодульной коммуникации;

4) возможность проведения тестирования средств трансформации и визуализации его результатов;

5) простота встраивания системы в сторонние интеллектуальные информационные системы.

Для удовлетворения этих требований в качестве ключевых технологий были выбраны программная платформа контейнеризации Docker [2] и система удаленного вызова процедур gRPC [3].

Использование технологии контейнеризации позволяет разработчику модуля однократно настроить изолированное воспроизводимое окружение исполнения программного кода. Кроме того, существующие средства согласованного запуска нескольких контейнеров позволяют автоматизировать процесс сборки и запуска созданного при помощи платформы средства трансформации, в том числе как часть сторонних систем.

Система удаленного вызова процедур gRPC требует формального описания сервисов на языке описания интерфейсов Protocol Buffers [4]. Использование этого описания позволяет автоматически проверять совместимость интерфейсов сервисов, а также автоматически генерировать программный код по комплексированию отдельных сервисов. Требование реализации gRPC-интерфейса существенно не ограничивает разработчика модулей трансформации. Поддержка gRPC реализована для всех наиболее популярных современных языков программирования, а значительная часть программного кода для сетевого взаимодействия может быть сгенерирована автоматически. Еще одним преимуществом gRPC является эффективность при передаче сообщений за счет жесткости протокола, использования бинарной сериализации и HTTP 2.0 в качестве транспорта.

На этапе проектирования были выделены следующие сценарии использования платформы:

1) управление модулями трансформации;
2) комплексирование средств трансформации;
3) проведение тестирования средств трансформации;
4) анализ результатов тестирования средств трансформации;
5) выгрузка созданного средства трансформации.

В сценарии *Управление модулями трансформации* основной операцией является загрузка новых модулей трансформации в платформу. Для этого пользователь заблаговременно загружает в приватный Docker-репозиторий платформы Docker-образ программного модуля, предоставляющий после запуска gRPC-сервис. В пользовательском интерфейсе платформы пользователь добавляет новый модуль, указывая его имя, категорию, а также формальную спецификацию gRPC-сервиса при помощи proto-файла. Впоследствии пользователь может отредактировать эти параметры.

В сценарии *Комплексирование средств трансформации* пользователь выстраивает конвейер трансформации из загруженных в платформу модулей. Доступные модули отображаются в палитре модулей редактора диаграмм потоков данных системы научной визуализации SciVi [17]. Для пополнения этой палитры разработанная платформа на основании загруженного формального описания интерфейса каждого модуля на языке описания сервисов gRPC Protobuf автоматически формирует правила конфигурации в онтологическом формате, используемом редактором. В редакторе пользователь может перетащить необходимые модули из палитры в область редактирования диаграммы, после чего соединить соответствующие сокеты (входы и выходы) узлов вершин (рис. 1). Редактор позволяет отфильтровывать только совместимые с выбранным модули трансформации, а также проверять корректность создаваемых дуг. Построенную диаграмму трансформации можно сохранить в платформе с целью последующего редактирования.

[2] URL: https://www.docker.com/
[3] URL: https://grpc.io/
[4] URL: https://developers.google.com/protocol-buffers/

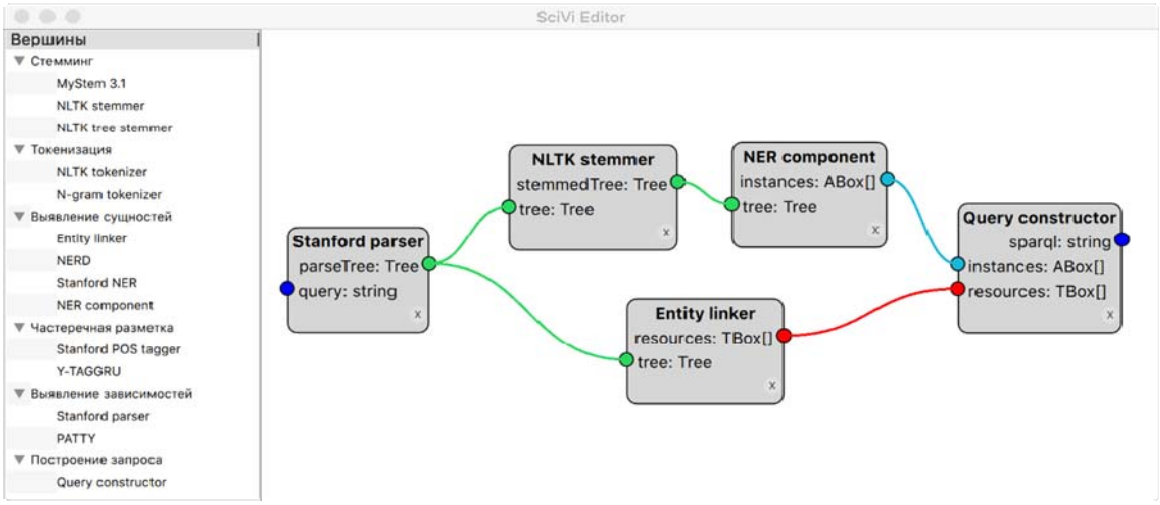


Рис. 1. Диаграмма средства трансформации, построенная в системе SciVi

Fig. 1. Diagram of Transforming Tool Built in SciVi System

После сохранения диаграммы в фоновом режиме запускается процесс комплексирования средства трансформации, соответствующего этой диаграмме. Для этого в копию заранее подготовленного шаблона проекта на языке Scala добавляются proto-файлы с описанием всех сервисов модулей, использованных в диаграмме. Затем автоматически генерируется программный код на языке Scala, который последовательно либо параллельно (в зависимости от дуг в диаграмме) осуществляет соответствующие gRPC-вызовы. Полученный проект компилируется и упаковывается в Docker-контейнер. После этого автоматически формируется соответствующий диаграмме конфигурационный файл для утилиты Docker Compose, позволяющей установить зависимости между Docker-контейнерами всех использованных модулей трансформации, а также настроить их совместный запуск. В платформе имеется возможность автоматической генерации нескольких средств трансформации по одной диаграмме. В таком случае генерируются альтернативные конвейеры трансформации, в которых отдельные модули заменены на другие совместимые по интерфейсу модули того же класса.

В сценарии *Проведение тестирования средств трансформации* пользователь выбирает одно или несколько комплексированных ранее средств трансформации и запускает их тестирование на одном или нескольких заранее подготовленных тестовых наборах. В настоящий момент тестовые наборы представляют собой множества из кортежей следующей структуры:

1) исходный естественно-языковой запрос;
2) ожидаемый результирующий SPARQL-запрос;
3) онтология, к которой задается запрос.

В сценарии *Анализ результатов тестирования средств трансформации* пользователь может просмотреть результаты тестирования в срезах по средствам трансформации и по тестовым наборам. Для каждого тестового случая пользователь может, не выходя из приложения, просмотреть ожидаемый и фактический результат трансформации ЕЯ-запроса в SPARQL-запрос в наглядном виде с использованием разработанного ранее средства визуального графического представления [18].

В сценарии *Выгрузка созданного средства трансформации* пользователь скачивает созданный на этапе комплексирования файл конфигурации для утилиты Docker Compose (docker-compose.yml). Благодаря принятым архитектурным решениям этого файла достаточно для автоматической загрузки и развертывания комплексированного средства трансформации на любом компьютере пользователя (при возможности подключения к приватному Docker-репозиторию платформы).

Особенности реализации

При загрузке модуля в платформу пользователь сопровождает его спецификацией сервиса на языке описания интерфейсов Protocol Buffers. Спецификация содержит имя сервиса, названия реализуемых сервисом серверных частей удаленных процедур, форматы входных и выходных сообщений. Встроенный сторонний редактор диаграмм потоков данных SciVi принимает на вход онтологическое описание модулей для добавления их в палитру редактора, отображения входных и выходных параметров модуля и анализа их совместимости. Для разбора файла спецификации и преобразования описания сервиса в требуемое SciVi онтологическое описание были использованы классы FileDescriptorSet, FileDescriptorProto и DescriptorProto из официальной библиотеки Protocol Buffers, позволяющие получить объектное представление спецификации сервисов.

В качестве языка программирования для автоматически генерируемого кода конвейера трансформации был выбран язык Scala. Статическая типизация и компилируемость этого языка позволяла на этапе разработки платформы быстрее находить некоторые ошибки в сгенерированном программном коде без необходимости запуска тестов. Кроме того, в системе сборки Scala build tool (sbt) были использованы релевантные для задачи плагины ScalaPB и sbt Native Packager. ScalaPB позволяет автоматически сгенерировать программный код gRPC-клиента на этапе компиляции без необходимости вызова сторонних программ генерации. sbt Native Packager позволяет автоматически упаковать Scala-приложение в Docker-контейнер и опубликовать его в задаваемом в параметрах проекта репозитории.

Исходя из того, что на данном этапе развития платформы предполагается, что пользователь в редакторе диаграмм потоков данных может строить только направленные ациклические графы, программный код конвейера трансформации генерируется с учетом возможных параллельных дуг графа. В таком случае более эффективным является параллельное выполнение запросов к отдельным модулям трансформации в асинхронном режиме. Для представления результатов асинхронных вызовов удаленных процедур gRPC в Scala используется класс Future, представляющий собой программную обертку над значением, которое в данный момент может быть еще не вычислено. Несмотря на возможное отсутствие значений, экземпляры классов Future можно комбинировать между собой в неблокирующем режиме. Благодаря этому весь генерируемый программный код представляет собой цепочку из неблокирующих вызовов с соответствующим параллельным исполнением запросов (см. пример в следующем разделе).

Пример комплексирования и тестирования средства трансформации с использованием разработанной платформы

Для демонстрации разработанной платформы была рассмотрена задача создания системы для трансформации ЕЯ-запросов в SPARQL-запросы к базе знаний Geobase [5]. База знаний предоставляется в виде фактов на языке Prolog. Для преобразования Prolog-фактов в OWL были использованы правила отображения, представленные в работе [19]. В качестве модулей системы были выбраны сторонние компоненты: стеммер NLTK Snowball stemmer (Python) [8], парсер зависимостей Stanford Parser (Java) [20]. Кроме того, были созданы демонстрационные компоненты NER Component (Kotlin), Entity Linker (Kotlin), Query Constructor (Kotlin), решающие задачи отображения токенов запроса на ресурсы онтологии и построения SPARQL-запроса по множеству найденных ресурсов.

Для каждого модуля была создана программная обертка, предоставляющая gRPC-сервис. Каждый модуль был загружен в Docker-репозиторий и зарегистрирован в разработанной платформе. Для каждого модуля автоматически сформирован файл с онтологическим описа-

[5] URL: http://www.cs.utexas.edu/users/ml/nldata/geoquery.html

нием в формате использованного редактора диаграмм потоков данных. На рис. 2 приведен скриншот графического представления фрагмента автоматически сформированной онтологии (графическое представление создано при помощи редактора онтологий ОНТОЛИС 2.0 [21]).

После этого при помощи редактора диаграмм потоков данных был построен граф трансформации ЕЯ-запроса в SPARQL-запрос (см. рис. 1). На основании этого графа был автоматически сгенерирован и скомпилирован Scala-проект, выполняющий трансформацию запроса посредством удаленного вызова процедур использованных модулей в соответствии со спецификацией пользователя.

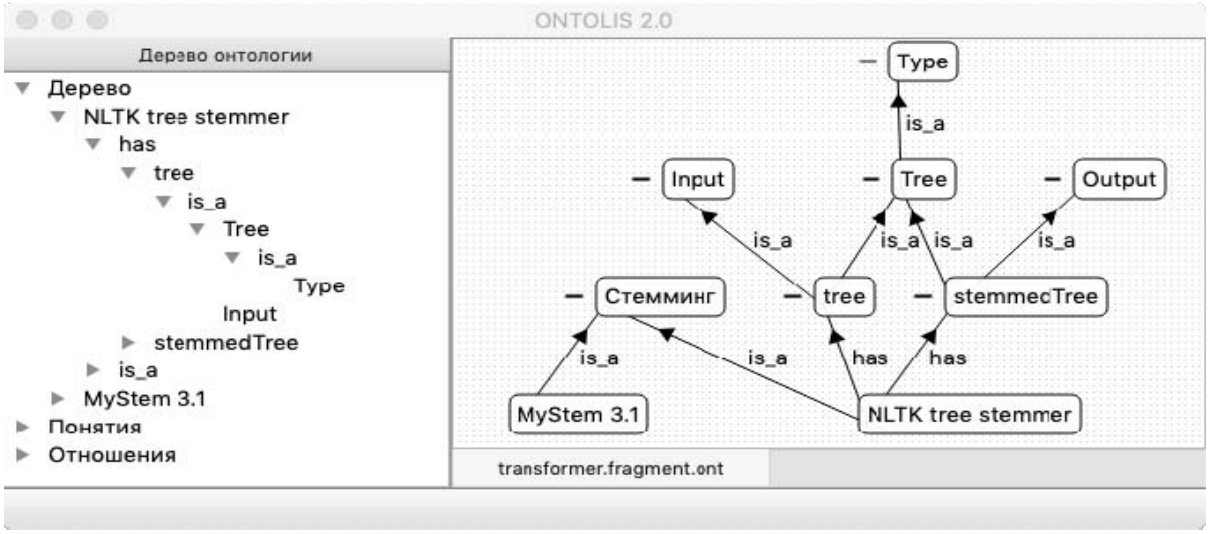


Рис. 2. Фрагмент автоматически сформированной онтологии

Fig. 2. Fragment of automatically built ontology

В листингах 1–5 приведены результаты работы всех модулей трансформации в виде текстового представления сообщений Protocol Buffers на примере запроса «What states does the Mississippi run through?».

Листинг 1. Результат работы модуля Stanford parser
Listing 1. Stanford Parser Module's Result

```
Tree: {
  tokens: [
    token: { index: 1, word: "What", pos: "WDT" },
    token: { index: 2, word: "states", pos: "NNS" },
    token: { index: 3, word: "does", pos: "VBZ" },
    token: { index: 4, word: "the", pos: "DT" },
    token: { index: 5, word: "Mississippi", pos: "NNP" },
    token: { index: 6, word: "run", pos: "NN" },
    token: { index: 7, word: "through", pos: "IN" },
    token: { index: 8, word: "?", pos: "." }
  ],
  dependencies: [
    dependency: { dep: "ROOT", governor: 0, dependent: 3 },
    dependency: { dep: "det", governor: 2, dependent: 1 },
    dependency: { dep: "nmod", governor: 3, dependent: 2 },
```

```
      dependency: { dep: "det", governor: 6, dependent: 4 },
      dependency: { dep: "compound", governor: 6, dependent: 5 },
      dependency: { dep: "dobj", governor: 3, dependent: 6 },
      dependency: { dep: "case", governor: 2, dependent: 7 }
    ]
  }
```

Листинг 2. Результат работы модуля Entity linker
Listing 2. Entity Linker Module's Result

```
TBox: [
  iri: "http://www.fluz.sp.owl#State",
  iri: "http://www.fluz.sp.owl#run_through"
]
```

Листинг 3. Результат работы модуля NLTK Snowball stemmer
Listing 3. NLTK Snowball Stemmer Module's Result

```
Tree: {
  tokens: [
    token: { index: 1, word: "what", pos: "WDT" },
    token: { index: 2, word: "state", pos: "NNS" },
    token: { index: 3, word: "doe", pos: "VBZ" },
    token: { index: 4, word: "the", pos: "DT" },
    token: { index: 5, word: "Mississippi", pos: "NNP" },
    token: { index: 6, word: "run", pos: "NN" },
    token: { index: 7, word: "through", pos: "IN" },
    token: { index: 8, word: "?", pos: "." }
  ],
  dependencies: [
    dependency: { dep: "ROOT", governor: 0, dependent: 3 },
    dependency: { dep: "det", governor: 2, dependent: 1 },
    dependency: { dep: "nmod", governor: 3, dependent: 2 },
    dependency: { dep: "det", governor: 6, dependent: 4 },
    dependency: { dep: "compound", governor: 6, dependent: 5 },
    dependency: { dep: "dobj", governor: 3, dependent: 6 },
    dependency: { dep: "case", governor: 2, dependent: 7 }
  ]
}
```

Листинг 4. Результат работы модуля NER component
Listing 4. NER Component Module's Result

```
ABox: [
  iri: "http://www.fluz.sp.owl#mississippi_river"
]
```

Листинг 5. Результат работы модуля Query constructor
Listing 5. Query Constructor Module's Result

```
sparql: "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
  PREFIX geo: <http://www.fluz.sp.owl#>
  SELECT ?A {
```

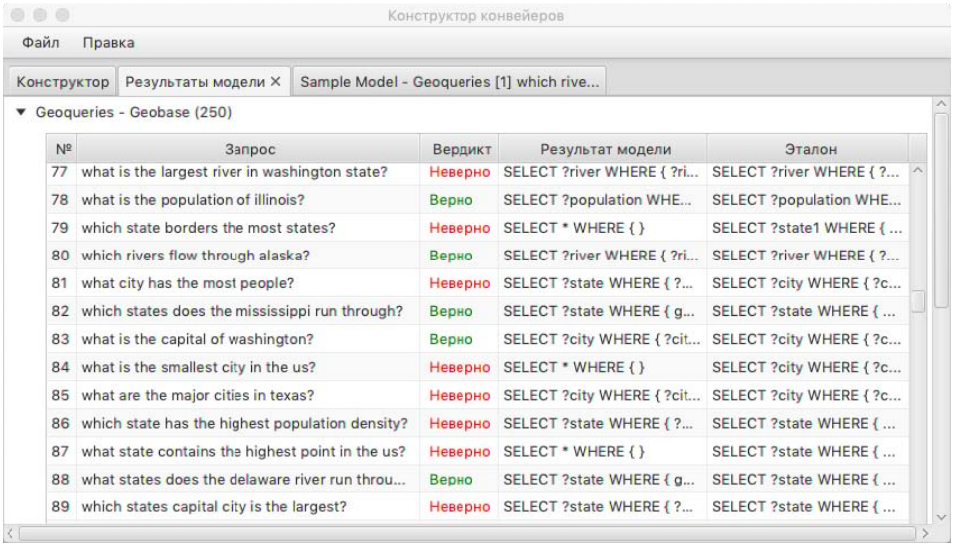
```
    geo:mississippi_river geo:run_through ?A .
    ?A rdf:type geo:State
  }"
```

Собранный проект был автоматически упакован в Docker-контейнер и загружен в приватный Docker-репозиторий платформы.

Листинг 6. Фрагмент автоматически сгенерированного кода вызова модулей трансформации
Listing 6. Fragment of Automatically Generated Code which Calls Transformation Modules

```
override def transform(request: TransformRequest): Future[TransformResponse] = {
  val parseResponseF = stubDependencyParser.parse(
                    ParseRequest(request.ontologySource, request.query))
  val stemmerResponseF = parseResponseF.flatMap(pr =>
    stubStemmer.stem(StemRequest(request.ontologySource, pr.parseTree)))
  val linkerResponseF = parseResponseF.flatMap(pr =>
    stubLinker.link(LinkRequest(request.ontologySource, pr.parseTree)))
  val nerResponseF = stemmerResponseF.flatMap(sr =>
    stubNer.ner(NerRequest(request.ontologySource, sr.stemmedTree)))
  val constructorResponseF = (for {
    nr <- nerResponseF
    lr <- linkerResponseF
  } yield stubConstructor.construct(ConstructRequest(request.ontologySource,
        nr.instances, lr.resources))).flatten
  constructorResponseF.map(cr => TransformResponse(cr.sparql))
}
```

С целью демонстрации пользовательского интерфейса просмотров результатов тестирования комплексированное средство трансформации было протестировано на наборе из 250 запросов Geoquery. На рис. 3 приведен пример просмотра результата модели на наборе тестов, а на рис. 4 – пример просмотра ожидаемых и фактических результатов тестового случая.



№	Запрос	Вердикт	Результат модели	Эталон
77	what is the largest river in washington state?	Неверно	SELECT ?river WHERE { ?ri...	SELECT ?river WHERE { ?...
78	what is the population of illinois?	Верно	SELECT ?population WHE...	SELECT ?population WHE...
79	which state borders the most states?	Неверно	SELECT * WHERE { }	SELECT ?state1 WHERE { ...
80	which rivers flow through alaska?	Верно	SELECT ?river WHERE { ?ri...	SELECT ?river WHERE { ?...
81	what city has the most people?	Неверно	SELECT ?state WHERE { ?...	SELECT ?city WHERE { ?c...
82	which states does the mississippi run through?	Верно	SELECT ?state WHERE { g...	SELECT ?state WHERE { ...
83	what is the capital of washington?	Верно	SELECT ?city WHERE { ?cit...	SELECT ?city WHERE { ?c...
84	what is the smallest city in the us?	Неверно	SELECT * WHERE { }	SELECT ?city WHERE { ?c...
85	what are the major cities in texas?	Неверно	SELECT ?city WHERE { ?cit...	SELECT ?city WHERE { ?c...
86	which state has the highest population density?	Неверно	SELECT ?state WHERE { ?...	SELECT ?state WHERE { ...
87	what state contains the highest point in the us?	Неверно	SELECT * WHERE { }	SELECT ?state WHERE { ...
88	what states does the delaware river run throu...	Верно	SELECT ?state WHERE { g...	SELECT ?state WHERE { ...
89	which states capital city is the largest?	Неверно	SELECT ?state WHERE { ?...	SELECT ?state WHERE { ...

Рис. 3. Пример просмотра результатов модели
Fig. 3. User Interface for Viewing Model's Testing Results

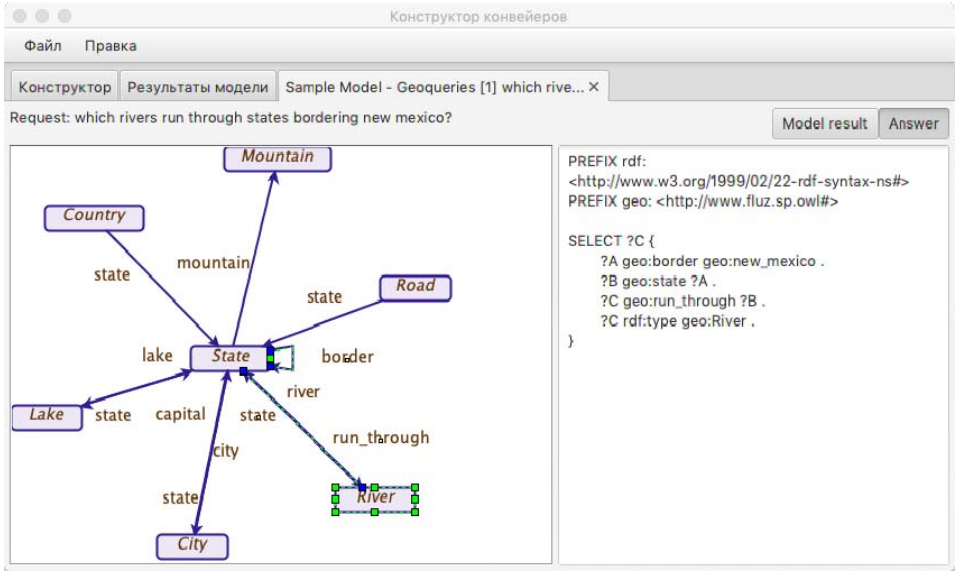


Рис. 4. Пример просмотра информации о результатах трансформации

Fig. 4. User Interface for Viewing Test's Detailed Information

Заключение

В работе были рассмотрены некоторые из современных средств трансформации ЕЯ-запросов в SPARQL-запросы. На основании схожести этапов их работы, а также по причине выявленных ограничений существующих платформ создания средств трансформации была сформулирована задача разработки собственной платформы комплексирования и тестирования средств трансформации.

Удовлетворение сформулированных к платформе требований было достигнуто при помощи использования двух ключевых технологий: программной платформы контейнеризации *Docker* и системы удаленного вызова процедур *gRPC*. Их использование позволило реализовать поддержку комплексирования переносимой системы из модулей, использующих наиболее подходящие для того или иного этапа программные технологии.

Описанная платформа автоматизирует многие процессы разработки средств трансформации: созданы средства поддержки связывания отдельных модулей с поддержкой автоматической проверки корректности, создана система автоматического комплексирования средств трансформации на основе графа трансформации, создана система тестирования, интегрированная с разработанной ранее системой визуализации результатов трансформации.

В будущем планируется автоматически генерировать средства трансформации произвольной структуры в дополнение к реализованной возможности генерации альтернативных средств заданной структуры, а также отображать различия между ожидаемым и фактическим результатами трансформации в наглядном виде при помощи визуального графического представления онтологии.

Список литературы / References

1. **Gruber T. R.** A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 1993, vol. 5, no. 2, p. 199–220. DOI 10.1006/knac.1993.1008.

2. **Xiao G., Calvanese D., Kontchakov R., Lembo D., Poggi A., Rosati R., Zakharyaschev M.** Ontology-based data access: A survey. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, 2018, p. 5511–5519. DOI 10.24963/ijcai.2018/777.

3. **Чуприна С. И., Постаногов И. С.** Концепция обогащения унаследованных информационных систем сервисом запросов на естественном языке // Вестник Перм. ун-та. Математика. Механика. Информатика. 2015. Т. 2. С. 78–86.

   Chuprina S. I., Postanogov I. S. Kontseptsiya obogashcheniya unasledovannykh informatsionnykh sistem servisom zaprosov na estestvennom yazyke [Enhancing Legacy Information Systems with a Natural Language Query Interface Service]. *Vestnik Permskogo universiteta. Matematika. Mekhanika. Informatika*, 2015, vol. 2, p. 78–86. (in Russ.)

4. **Dubey M., Dasgupta S., Sharma A., Höffner K., Lehmann J.** Asknow: A framework for natural language query formalization in SPARQL. In: International Semantic Web Conference. Cham., 2016, p. 300–316. DOI 10.1007/978-3-319-34129-3_19.

5. **Shekarpour S., Marx E., Ngomo A. C. N., Auer S.** SINA: Semantic Interpretation of User Queries for Question Answering on Interlinked Data. In: Web Semantics: Science, Services and Agents on the World Wide Web, 2015. DOI 10.1016/j.websem.2014.06.002.

6. **Xu K., Zhang S., Feng Y., Zhao D.** Answering natural language questions via phrasal semantic parsing. In: Natural Language Processing and Chinese Computing, 2014, p. 333–344. DOI 10.1007/978-3-662-45924-9_30.

7. **Diefenbach D., Lopez V., Singh K., Maret P.** Core techniques of question answering systems over knowledge bases: a survey. *Knowledge and Information systems*, 2018, vol. 3, no. 55, p. 529–569. DOI 10.1007/s10115-017-1100-y

8. **Bird S., Klein E., Loper E.** Natural language processing with Python: analyzing text with the natural language toolkit. O'Reilly Media, Inc., 2009, 504 p.

9. **Kostareva T., Chuprina S., Nam A.** Using Ontology-Driven Methods to Develop Frameworks for Tackling NLP Problems. In: AIST (Supplement), 2016, p. 102–113.

10. **Both A., Diefenbach D., Singh K., Shekarpour S., Cherix D., Lange C.** Qanary – a methodology for vocabulary-driven open question answering systems. In: The Semantic Web. Latest Advances and New Domains, 2016, p. 625–641. DOI 10.1007/978-3-319-34129-3_38.

11. **Ferrandez O., Spurk C., Kouylekov M., Dornescu I., Ferrandez S., Negri M., Magnini B.** The QALL-ME Framework: A specifiable-domain multilingual Question. *Answering architecture. Web semantics: Science, services and agents on the world wide web*, 2011, vol. 2, no. 9, p. 137–145. DOI 10.1016/j.websem.2011.01.002

12. **Marx E., Usbeck R., Ngomo A.C.N., Höffner K., Lehmann J., Auer S.** Towards an open question answering architecture. In: Proceedings of the 10th International Conference on Semantic Systems, 2014, p. 57–60. DOI 10.1145/2660517.2660519.

13. **Hohpe G.** Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Prentice Hall, 2004, 683 p.

14. **Trivedi P., Maheshwari G., Dubey M., Lehmann J.** Lc-quad: A corpus for complex question answering over knowledge graphs. In: Proceedings of the 16th International Semantic Web Conference (ISWC), 2017, p. 210–218. DOI 10.1007/978-3-319-68204-4_22.

15. **Bordes A., Usunier N., Chopra S., Weston J.** Large-scale simple question answering with memory networks. In: arXiv preprint arXiv:1506.02075, 2015.

16. **Usbeck R., Gusmita R. H., Saleem M., Ngomo A. C. N.** 9th challenge on question answering over linked data (QALD-9). In: CEUR Workshop Proceedings, 2018, vol. 2241, p. 58–64.

17. **Рябинин К. В., Чуприна С. И., Бортников А. Ю.** Автоматизация настройки систем научной визуализации на специфику разнообразных источников данных // Научная визуализация. 2016. Т. 8, № 4. С. 1–14.

Ryabinin K. V., Chuprina S. I., Bortnikov A. Yu. Avtomatizatsiya nastroiki sistem nauchnoi vizualizatsii na spetsifiku raznoobraznykh istochnikov dannykh [Automated Tuning of Scientific Visualization Systems to Varying Data Sources]. *Nauchnaya vizualizatsiya*, 2016, vol. 8, no. 4, p. 1–14. (in Russ.)

18. **Турова И. А., Постаногов И. С.** Использование визуального графического представления в системах трансформации естественно-языковых запросов в SPARQL-запросы // Математика и междисциплинарные исследования – 2018. Пермь, 2018. С. 111–114.

    Turova I. A., Postanogov I. S. Ispol'zovanie vizual'nogo graficheskogo predstavleniya v sistemakh transformatsii estestvenno-yazykovykh zaprosov v SPARQL-zaprosy [Using the Visual Graphical Representation in Systems that Transform Natural Language Queries into SPARQL Queries]. In: Matematika i mezhdistsiplinarnye issledovaniya – 2018. Perm, 2018, p. 111–114. (in Russ.)

19. **Luz F. F., Finger M.** Semantic Parsing Natural Language into SPARQL: Improving Target Language Representation with Neural Attention. In: arXiv preprint arXiv:1803.04329. – 2018.

20. **Chen D., Manning C.** A fast and accurate dependency parser using neural networks. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), 2014, p. 740–750. DOI 10.3115/v1/D14-1082.

21. ОНТОЛИС 2.0: Свидетельство о государственной регистрации программ для ЭВМ / И. С. Постаногов, С. И. Чуприна. № 2017610729; дата регистрации 16.01.2017.

    ONTOLIS 2.0: svidetel'stvo o gosudarstvennoi registratsii programm dlya EVM. I. S. Postanogov, S. I. Chuprina. № 2017610729; data registratsii 16.01.2017. (in Russ.)

Материал поступил в редколлегию
Received
04.02.2019

Сведения об авторах / Information about the Authors

Постаногов Игорь Сергеевич, аспирант, старший преподаватель кафедры математического обеспечения вычислительных систем Пермского государственного национального исследовательского университета (ул. Букирева, 15, Пермь, 614068, Россия)

Igor S. Postanogov, PhD Student, Senior Lecturer, Department of Software Computing Systems, Perm State University (15 Bukirev Str, Perm, 614068, Russian Federation)

ipostanogov@outlook.com

Турова Ирина Алексеевна, студентка кафедры математического обеспечения вычислительных систем Пермского государственного национального исследовательского университета (ул. Букирева, 15, Пермь, 614068, Россия)

Irina A. Turova, Student, Department of Software Computing Systems, Perm State University (15 Bukirev Str, Perm, 614068, Russian Federation)

turovaia@yandex.ru