Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Южно-Уральский государственный университет»
(национальный исследовательский университет)
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

РАБОТА ПРОВЕРЕНА
Рецензент
_____
«___»_____ 2017 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой
_____К.А. Домбровский
«___»_____2017 г.

Формирование семантической сети на основе анализа текстов

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-09.04.01.2017.040 ПЗ ВКР

Руководитель работы,
доцент каф. «Электронные
Вычислительные машины»
_____И.Л. Кафтанников
«___»_____2017 г.

Автор работы
студент группы КЭ-271
_____В.В. Яшин
«___»_____2017 г.

Нормоконтролёр, ст. преп. каф.
«Электронные вычислительные
машины»
_____В.В. Лурье
«___»_____2017 г.

Челябинск 2017

Аннотация

Яшин В.В. Формирование семантической сети на основе анализа текстов. – Челябинск: ФГАОУ ВО «ЮУрГУ» (НИУ), КЭ; 2017, 94 с., 11 ил. Библиография литературы – 40 наименований, 2 приложения

В работе рассматриваются основные проблемы описания знаний в виде графов, проводится обзор существующих тезаурусов и способов их пополнения, а также анализ подходов к формализации текстов на естественном языке. Особое внимание уделено структурному, лексическому, анализу текстов.

Целью работы является обобщение подходов, методов и алгоритмов анализа текстов на естественных языках для проектирования и разработки семантической сети, расширяющейся на основе текстов на русском языке.

Для достижения цели были изучены литература и публикации по темам, связанным с семантическими сетями, тезаурусами, дистрибутивным анализом, как методом исследования языка, методами анализа слабоструктурированных источников, методами лексического анализа текстов; были описаны существующие тезаурусы. В результате анализа публикаций и существующих реализаций была получена модель семантической сети и был представлен алгоритм ее пополнения на основе анализа текстов.

При технической реализации программных модулей использовались и были поверхностно описаны в работе современные подходы программной инженерии. При реализации проекта был использован C#, TypeScript и широкий список фреймворков, библиотек, а также паттернов и подходов к проектированию и написанию программного кода.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В условиях повышения требований к пользовательскому интерфейсу, к функциональным возможностям, количеству пользователей программных систем в различных сферах человеческой жизни растет потребность в более интеллектуальных компонентах этих систем. Пользователи ожидают от приложений персонализации. Сервисам электронной торговли, размещения контекстной рекламы требуется возможность целевых рекомендаций. А современные операционные системы и некоторые прикладные приложения содержат помощников с функциями распознавания естественного языка. Поэтому в последнее время возрастает актуальность исследований в области искусственного интеллекта, в целом, и формализации текстов на естественных языках, в частности.

Интерес проявляют как крупные компании, IT-гиганты, так и небольшие стартапы. Основные тренды IT: виртуальные ассистенты, чат-боты, автоматические каталогизаторы, сервисы распознавания речи, сервисы персональных рекомендаций, сервисы выделения фактов из текстов на естественном языке и анализа неструктурированных данных или слабоструктурированных данных, Data Mining, Web Mining, семантический веб – используют алгоритмы формализации информации, представленной на естественных языках.

Активно развивается один из популярных способов представления знаний в виде семантических графов, в том числе тезаурусов и онтологий. Расширяются существующие мультиязычные и одноязычные тезаурусы, а также появляются сравнительно новые.

Несмотря на то, что область исследования развивается уже многие годы, приложения, в том числе созданные крупнейшими компаниями, по-прежнему ошибаются. Степень понимания естественных языков машинами остается далекой от степени понимания человеком. Тезаурусы в основном расширяются вручную, что не позволяет им развиваться также динамично, как языкам, которые они

отражают. Более того наполнение связями между концептами языка в тезаурусе даже для часто употребляемых слов, не являющихся неологизмами, или словами, приобретающими новый смысл, оставляет желать лучшего.

На сегодняшний день достигнут высокий уровень развития аппаратных, программных, лингвистических средств, высокий уровень развития смежных дисциплин, в частности программной и системной инженерии. И результаты этих достижений можно использовать для развития технологических средств формирования и представления семантических сетей, анализа текстов на естественных языках.

1 АНАЛИЗ ЗАДАНИЯ

1.1 Актуальность задачи

Развитие алгоритмов формализации текстов на естественном языке обусловлено не только ростом количества исследований и публикаций, но и интересом различных коммерческих компаний, как небольших, так и крупнейших IT-гигантов. Google, Apple, Facebook, Microsoft, Amazon, Yandex разрабатывают средства формализации естественной речи или текстов и применяют их в своих продуктах. В 2017 году большинство людей отлично знают Siri, Cortana, Google Assistant или Alexa.

Появляется большое количество стартапов в области обработки неструктурированных данных, в том числе и информации, содержащейся только на естественном языке. Например, чат-боты, используемые для консультирования и поддержки, или системы анализа пользовательского поведения, или системы, анализирующие потребности потенциальных клиентов, либо их мнение о компании, продукте. Вот некоторые из них: Digital Genius, Bot Scanner, Api.ai.

Глобальный веб аналогично стремится к новым стандартам. Известное как Semantic Web (или Web of Data) направление развития глобальной паутины позволяет осуществлять автоматическую обработку семантики, смысла, информации в интернете. Это может использоваться для повышения релевантности результатов поисковых запросов и стать толчком к созданию или преображению к более совершенным поисковым системам, что происходит уже сейчас.

Другим важным фактором, обусловливающим актуальность исследований в области искусственного интеллекта, является развитие аппаратных, программных и лингвистических средств. Возрастают вычислительные возможности аппаратного обеспечения. Расширяются API операционных систем, создаются новые языки программирования, а также еще активнее создаются фреймворки и облачные платформы, реализующие алгоритмы статистического анализа и

структурного анализа естественных языков, алгоритмы машинного обучения. В связи с этим большая часть алгоритмов и концепций искусственного интеллекта, от семантических сетей до машинного обучения, зарождавшиеся в шестидесятые – семидесятые годы двадцатого века, на этапе формирования классических направлений искусственного интеллекта, вступили в новую эпоху своего развития.

Несмотря на высокий уровень интереса к проблемам обработки неструктурированной информации, представленной на естественном языке, область остается недостаточно изученной. Это связано с большим количеством естественных языков, которые могут сильно отличаться друг от друга, и довольно высокой сложностью как естественных языков, так и сложностью формального описания семантики.

Довольно распространенным способом представления семантики является граф, отображающий связь между различными понятиями и названный семантической сетью.

Семантические сети позволяют графически представить любое предложение, а затем вернуть его в текстовое представление, условно, семантически эквивалентным. Условность связана со сложностью определения критериев семантической эквивалентности. Из семантических сетей за счет механизмов наследования некоторых типов связей можно получить данные – знания, которые являются логичным выводом из помещенной в нее информации, но не помещенные в нее напрямую.

Сегодня существует множество семантических сетей, имеющих разную структуру, разную степень полноты, а также специфику хранящихся в ней знаний. Перечислим Word Net, Cyc, Concept Net, DOLCE, Euro Word Net, Germa Net и прочие. А в России – РуТез, RussNet, Yet Another RussNet (YARN).

1.2 Цели и задачи работы.

Актуальность проблемы определяет важность исследований в области выделения знаний из информации, представленной на естественных языках. В качестве модели хранения знаний была выбрана семантическая сеть как активно используемая на различных языках и имеющая множество аналогов, как в России, там и в мире.

Исходя из вышеописанного была сформулирована цель проекта – обобщение подходов, методов и алгоритмов семантического анализа текстов на естественных языках для проектирования и разработки семантической сети, расширяемой на основе анализа текстов на русском языке. Обеспечение возможностей интеграции разрабатываемой системы с системами, использующими другие подходы к пополнению сети, а также системами представления и использования знаний, хранящихся в сети.

Исходя из цели работы сформулируем задачи.

– Изучение теоретических основ, литературы и публикаций на тему семантических сетей и семантического анализа. Рассмотрение существующих тезаурусов и систем семантического разбора текстов.

– Обобщение описанных методов, подходов и алгоритмов для выполнения разработки системы.

– Проектирование системы формирования концептов и связей между ними на основе текстов на русском языке.

– Проектирование API для обеспечения возможностей внешнего расширения сети.

– Техническая реализация минимально жизнеспособной спроектированной системы.

Выводы по разделу один

В данном разделе рассматривалась актуальность задачи. Была сформулирована цель выпускной квалификационной работы. Были перечислены основные задачи, необходимые для выполнения цели работы.

2 АНАЛИТИЧЕСКАЯ ЧАСТЬ

2.1 Определение основных понятий, используемых в работе

«Семантическая сеть – ориентированный граф для представления знаний в моделях взаимосвязанных узлов и дуг» [1]. В работе также используется понятие семантический граф, эквивалентное понятию семантическая сеть. На рисунке 2.1 представлен пример семантической сети.

Исходя из определения выше, предложенного Джоном Совой (John Sowa), сеть может иметь различную структуру, т.е. иметь различное количество типов дуг, бинарные или n-арные отношения. Сеть может иметь мультиграфовую структуру, или представлять граф без кратных дуг.

Распространенной разновидностью семантических сетей, имеющей большое количество приложений, являются тезаурусы.

Тезаурус, применительно к компьютерным наукам, рассматривается как «словарь, в котором слова и словосочетания с близкими значениями сгруппированы в единицы, называемые понятиями, концептами или дескрипторами, и в которые явно (в виде отношений, иерархий) указываются семантические отношения между этими понятиями (концептами, дескрипторами).» [2, с.20]

Основными типами отношений, используемыми в большинстве существующих на сегодняшний день тезаурусов, являются гиперонимы – гипонимы, меронимы – холонимы, антонимы – синонимы, омонимы [2, с.30-37]. Выше и далее сначала приводятся лингвистические определения.

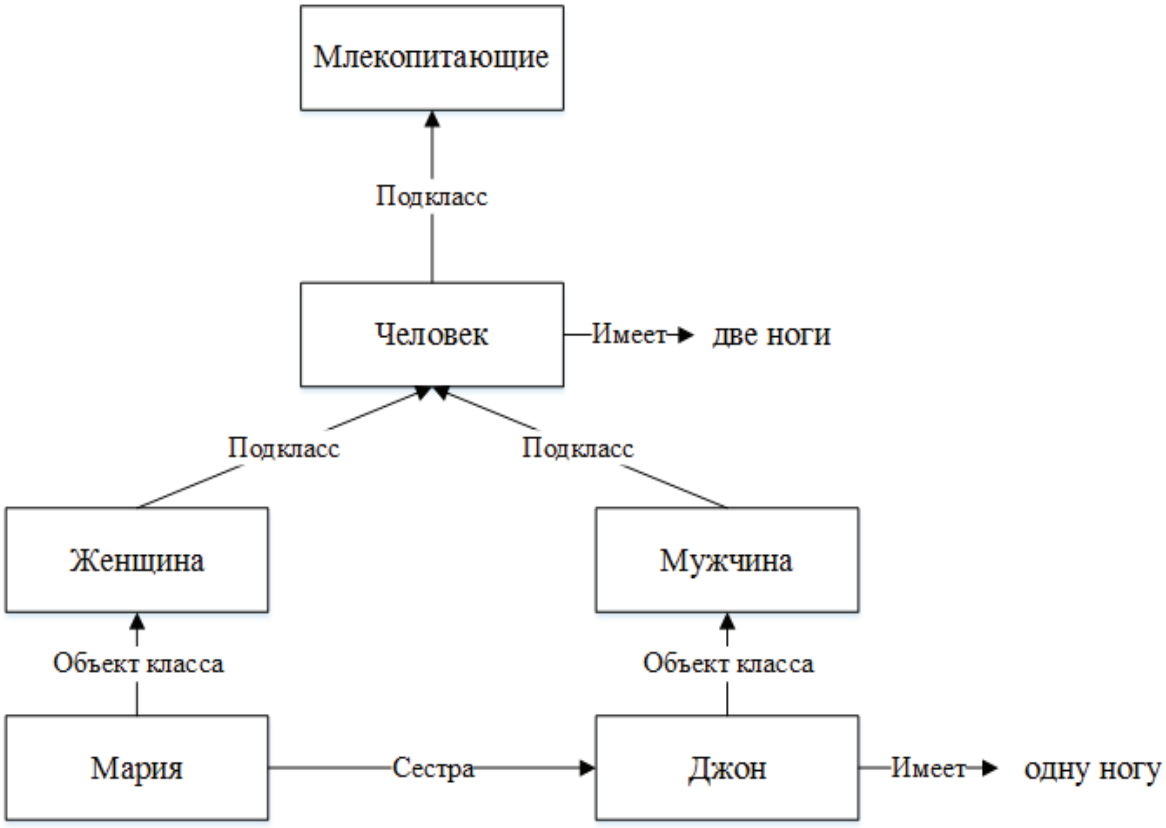

Рисунок 2.1 – Пример семантической сети

Гипероним представляет более общую сущность, относительно другой сущности. Например, «зверь» – гипероним сущности «кошка». Гипоним представляет более частную сущность. «Кошка» – гипоним сущности «зверь». В компьютерных науках более применимы понятия: отношение IS-A, отношение класс-подкласс, отношение KIND-OF. В других науках отношение также может называться таксономическим. В дальнейшем будем обозначать данный тип связи, как класс-подкласс, либо для обозначения конкретного направления отношения гипероним-гипоним.

Мероним – сущность, которая является составной частью другой сущности. Например, «двигатель» – мероним сущности «автомобиль». Холоним – сущность, которая включает, как часть, другое понятия бинарного отношения. «Автомобиль»

– холоним сущности «двигатель». В компьютерных науках отношение называют часть-целое или «A-PART-OF» или «HAS-A» (с учетом направления отношения) или MEMBER-OF, которое часто рассматривается как подмножество множества отношений часть-целое. В дальнейшем будем обозначать данный тип связи часть-целое, или мероним-холоним для отображения направления отношения.

Все описанные выше отношения обладают свойством транзитивности.

Также существует отношение «класс-объект», которое указывает на принадлежность конкретного предмета к тому или иному классу. Этот тип отношений крайне редко используется в существующих тезаурусах. Этот тезис следует из обзора существующих тезаурусов в пункте 2.4. Дополнительно может использоваться крайне широкий список различных типов вспомогательных отношений.

В пунктах 2.2 и 2.3 также используются такое понятие, как корпус языка.

«Корпус – это собрание текстов в электронной форме, в котором можно осуществлять поиск слов, словосочетаний, грамматических форм, значений слов с помощью определенной поисковой системы.» [3]

Корпусы могут содержать различные выборки текстов. Например, отдельную книгу, тексты определенной тематики или автора. Однако наибольшую практическую пользу имеют национальные корпусы. Хороший национальный корпус должен быть представительным и сбалансированным, т.е. отображать тексты всех возможных жанров, диалектов, различной лексики, применяемой как устно, так и письменно. Кроме того, тексты в национальном корпусе должны быть представлены пропорционально их доле в применении всеми носителями языка.

Подробнее корпусы рассмотрим в пунктах 2.3 и 3.6.

Применительно к тезаурусу WordNet, рассматриваемому в пункте 2.4, определим термин синсет.

«Синсет – синонимический ряд – множество слов, связанных отношением синонимии, являющимся разбиением множества всех лексических единиц на классы эквивалентности, выражающие сущность каких-либо понятий.» [4]

2.2 Обзор литературы и публикаций

В работах T.K. Landauer, P.W. Folts, D. Laham «An Introduction to Latent Semantic Analysis» [5] и T.K. Landauer, S.T Dumais «A Solution to Plato's Problem: The Latent Semantic Analysis Theory of Acquisition, Induction, and Representation of Knowledge» [6] описывается метод, названный латентно-семантический анализ (ЛСА или LSA).

Латентно-семантический анализ – это полностью автоматическая математическая/статистическая техника для извлечения и вывода отношений между контекстуальными использованиями слов при проходе высказывания. [5]

Данный метод не использует описанные людьми словари, семантические сети, базы знаний, грамматики, синтаксические или морфологические правила.

Для выполнения анализа авторы предлагают провести токенизацию текста, выделить все слова и отфильтровать их таким образом, чтобы получить множество всех уникальных слов из текста. Некоторые авторы, например, И.В. Барановский в работе «Методы и средства семантического анализа документов в системе делопроизводства», [7] предлагают получать не только отдельные слова, но и некоторые словосочетания, которые имеют особый смысл именно в словосочетании. Для выделения словосочетаний предлагается использовать словарь терминов в виде тезауруса или иной семантической сети.

Авторами «An Introduction to Latent Semantic Analysis» предлагается строить матрицу, строками которой являются уникальные слова, а столбцами – высказывания (документы), в ячейках указывается, какое количество раз слово встречается в соответствующем высказывании (документе). Над построенной матрицей проводится сингулярное разложение по формуле 2.1

$$A = USV^T, \tag{2.1}$$

где U и V – ортогональные матрицы, а S – диагональная матрица.

Элементы матрицы S необходимо упорядочить в порядке убывания.

Причем у сингулярного разложения есть свойство, при котором можно ограничить количество строк в матрицах S и V^T и, соответственно, количество столбцов матрицы U. Результирующая матрица будет близкой к A.

Например, в результате разложения была получена матрица U, представленная в таблице 2.1. [8]

Таблица 2.1 – Матрица U результата сингулярного разложения.

	Выражение 1	Выражение 2	Выражение 3
Человек	-0,254	-0,102	0,289
Интерфейс	-0,148	-0,071	0,135
Компьютер	-0,22	0,046	-0,0164
Пользователь	-0,411	0,059	-0,0338
Система	-0,612	-0,167	0,361

Так как матрица S упорядочена в порядке убывания элементов, то по правилам умножения матриц каждый последующий столбец матрицы U и строки матриц S и V^T будет меньше влиять на результат умножения. Исходя из этого можно сократить количество столбцов и строк соответствующих матриц. A оставшиеся интерпретировать, как координаты точек системы из k координат, где k – количество столбцов / строк соответствующих матриц.

Для таблицы 2.1 при k = 2, «Человек» имеет координаты [-0,0254; -0,102], а компьютер – [-0,22; 0,046]. Исходя из этих результатов можно получить меру близости этих слов как расстояние между точками в системе из k координат.

Существует также другой алгоритм разложения, описанный в статьях T. Hofmann «Probabilistic Latent Semantic Analysis» [9], использующий для разложения не линейную алгебру, а вероятностные модели. В работе отмечается, что такой подход дает более принципиальный результат, поэтому данный алгоритм подходит для практического применения.

Латентный семантический анализ является частным случаем анализа на основе дистрибутивной модели представления знаний.

Дистрибутивные модели используются в работах M. Sahlgren и J. Karlgren «Automatic Bilingual Lexicon Acquisition Using Random Indexing of Parallel Corpora» [10] и Ю. И. Морозова, Е. Б. Козеренко, М. М. Шарнин «Методика извлечения пословных переводных соответствий из параллельных текстов с применением моделей дистрибутивной семантики». [11]

«Дистрибутивный анализ – это метод исследования языка, основанный на изучении окружения (дистрибуции, распределения) отдельных единиц в тексте и не использующий сведений о полном лексическом или грамматическом значении этих единиц.» [11]

Дистрибутивный анализ основывается на описании слова в виде вектора, и позиционировании его на координатной плоскости. Таким образом, для любой предметной области можно построить «пространство слов» (word space) на заданной координатной плоскости. На которой расстояние между точками-словами отображает их семантическую близость.

В работах, представленных выше, приводятся методы извлечения переводных эквивалентов из двух параллельных (переведенных) текстов на основе дистрибутивного анализа.

А.Е. Письмак, А.Е. Харитонова, Е.А. Цопа, С.В. Клименков, в статье «Метод автоматического формирования семантической сети из слабоструктурированных источников» [12][13] описывают метод построения семантической сети на основе внешнего открытого словаря Wiktionary.

Посевкин Р.В. и Бессмертный И.А. в работе «Естественно-языковой пользовательский интерфейс диалоговой системы» [14] описан алгоритм формализованной обработки текстового ввода на ограниченном подмножестве естественного языка. Подмножество языка не содержит неоднозначностей на лингвистическом уровне. Что упрощает алгоритм формализации текста. Для выполнения обработки текста последовательно выполняются морфологический и морфемный, синтаксический и семантический анализ.

Результатом первого этапа являются морфологические характеристики: падеж, склонение, часть речи и прочее. Список характеристик может изменяться в

зависимости от используемого естественного языка. Для этого предлагается использовать словарь всех словоформ языка, или, в данном случае, его подмножества.

При морфемном анализе выделяются морфемы слова: приставка, корень, суффикс, окончание. Также происходит лемматизация, или выделение леммы слова.

На этапе синтаксического анализа строится синтаксическое дерево или дерево зависимостей, на основе которой выполняется построение семантической модели.

Так как применяется подмножество языка, то в предложенном алгоритме не требуется снятие морфологической неоднозначности.

В работе [15] авторы предлагают использовать следующие методы лингвистического и семантического анализа текстов:

Кластеризация массивов текстов на основе дистрибутивного анализа (конкретнее при помощи алгоритмов LDA или K-means), определение тематики высказывания из заранее определенного списка, выделение ключевых слов и семантических связей и представление их в виде облака «тегов». Разработаны алгоритмы построения контекстов по синтаксическому дереву и детектирования нетипичных для данной выборки текстов на основе One-Class SVM.

Hunter S. в статье [16] предложил новый метод построения сети на основе анализа этимологии слов английского языка.

R. Navigli и S.P. Ponzetto в работах «The automatic construction, evaluation and application of a wide-coverage multilingual semantic network» [17] и «BabelNet: Building a Very Large Multilingual Semantic Network» [18] рассматривают формирование мультиязыковой семантической сети путем интеграции лексикографических и энциклопедических знаний из WordNet и Википедии. Также рассматриваются методы машинного перевода для получения ресурсов с лексической информацией для различных языков.

Знания BabelNet формируются автоматически за счет представления всех концептов и всех лексических и семантических отношений между синсетами WordNet, а также представления страница Википедии, как концептов и

гиперссылок между страницами, как семантических отношений. В работе описан маппинг концептов WordNet на название страницы Wikipedia.

J. Nivre, I.M. Boguslavsky и L.L. Iomdin в работе «Parsing the SynTagRus Treebank of Russian» [19] описан парсер естественно-языковых текстов на основе лексических и морфологических особенностей языка. Описана система MaltParser, основанная на машинном обучении, являющаяся частным случаем такого парсера.

H. Palangri в статье «Deep Learning of Grammatically-Interpretable Representations Through Question-Answering» [20] представил архитектуру системы, в которой внутреннее представление обученной на задачах вопросно-ответных систем нейронной сети может быть интерпретировано с использование базовых концепций лингвистической теории, что позволяет выделять лексических похожие слова по группам, тем самым определяя скрытые семантические связи. Группа различных символов, замененная некоторым токеном, может отражать соответствующие абстрактное понятие.

В результате анализа литературы и научных публикаций, можем выделить следующие подходы к формированию семантических графов на основе анализа текстов:

1. Использование вероятностных подходов, основанных на частоте встречаемости множества слов в определенных контекстах. Для этой цели существуют алгоритмы дистрибутивного анализа, в частности, латентный семантический анализ, вероятностный латентный семантический анализ, другие алгоритмы, позволяющие определить пространство слов на некоторой системе координат, и представить каждое слово в виде вектора. А семантическую близость слов получать исходя расстояния между точками. Такой подход позволяет выделять скрытые ассоциации между словами, однако не позволяет определить, тип отношения между этими словами.

2. Алгоритмы обработки слабоструктурированных текстов на естественном языке. Особенно часто используется Wictionary (или Викисловарь) и Википедия, в качестве источника знания. Проблемами, которые решают данные алгоритмы, являются восстановление структуры текстов с нарушенной структурой

(написанных с нарушениями принятых правил структуризации) и выделение фактов на основе описанной структуры.

3. Структурный или лексический подход. Использование грамматических, синтаксических правил языка для выделения согласованных слов в предложении и построение семантической модели на их основе, и на основе уже существующей семантической модели.

Во всех вышеперечисленных подходах к выявлению отношений между понятиями на основе неструктурированных или слабоструктурированных естественно-языковых текстов могут быть использованы методы машинного обучения, что также представлено в некоторых из рассмотренных работ.

В данной работе не оговариваются ограничения на производительность разрабатываемого алгоритма. Целью является получение наиболее точной модели знаний. Исходя из этого наибольший интерес применительно к работе представляют алгоритмы со структурным подходом к анализу текстов на естественном языке.

2.3 Текущее состояние исследований проблематики представления знаний в виде графа и анализа текстов на естественных языках

Одним из важных свойств семантических сетей выступает наследование [25, С.479–480]. Наследование позволяет передавать некоторые типы связей от классов к их подклассам. Например, на рисунке 2.1 Мария является человеком и потому наследует свойство иметь две ноги. Однако, как правило, в семантических сетях один класс является подклассом сразу нескольких классов, то есть наблюдается множественное наследование. Множественное наследование может приводить к конфликту, при котором одним классов наследуется связи, противоречащие друг другу.

В качестве примера такого конфликта рассмотрим «парадокс Никсона» [26, С. 489–490]. Этот пример основан на том наблюдении, что Ричард Никсон был одновременно и квакером (поэтому по умолчанию пацифистом), и республиканцем (поэтому по умолчанию не пацифистом).

Таким образом, требуется либо игнорировать такие свойства, либо дополнять систему алгоритмами разрешения подобных ситуаций. Для решения этой проблемы, например, можно воспользоваться косвенным описанием с приоритетом, при котором будет задекларировано, что религиозные верования имеют приоритет над политическими.

На рисунке 2.1 классы «Женщина» и «Мужчина» могут наследовать от класса «Человек» связь «Имеет» с вершиной «две ноги». Также можно сделать вывод, что «Мария» имеет данную связь. Однако для Джона определена дополнительная связь «Имеет» «одну ногу», которая противоречит наследуемому слову. В данном случае данная связь однозначно переопределяет наследуемую и конфликта не происходит.

Парадокс Никсона может быть решен путем прямого определения конфликтующего свойства понятия в конкретном узле иерархии.

Как следствие парадокса Никсона можно выделить проблему межпредметных понятий, или в более широком смысле проблему омонимии.

Для примера, мы можем привести понятие «координаты». Координаты могут быть математическими, историческими, географическими и пр. Однако все эти понятия имеют некоторую общую семантику, выделенную в одно межпредметное понятие. Каждое подчиненное понятие, описанное выше, можно представить, как более конкретный элемент иерархии, то есть является гипонимом понятия «Координата».

Общие связи для межпредметного понятия можно вынести в обобщенные узлы. Для примера выше такой узел имел бы языковой термин «Координата». А связи, являющиеся специфичными, связать с его гипонимом, например, «Географическая координата». Таким образом, необходимо создавать связи не только для отдельных узлов, но и подграфов, который также может иметь множество эквивалентных

структур. Например, понятие «Географическая координата» можно представить также представить, как «Координата в географии». [26]

Это необходимо учитывать в структуре разрабатываемой сети.

Более общей проблемой является проблема омонимии. Для многих термов естественных языком характерен разное значение и смысл в зависимости от использованного контекста. Причем омонимия может быть не только семантической, как, например, для терма «лук», но и морфологической, как для слова «гладь», который в зависимости от контекста может иметь различную морфологическую структуру.

Слова-омонимы часто не имеют между собой никакой семантической общности, и потому рассматривать их в качестве одного и того же узла семантического графа не имеет смысла. Для того чтобы отличать омонимы в сети, необходимо описывать как часть некоторой предметной области, либо любой другой значащей связью. Например, «лук IS-A оружие» или «гладь IS-A действие».

Также рассмотрим основные проблемы анализа текстов на естественных языках. Большая часть проблем анализа текстов была озвучена в пункте 2.2 при выполнении обзора литературы и научных публикаций

В данном пункте обобщим полученные знания.

При рассмотрении алгоритмов дистрибутивного анализа были выделены проблемы наличия скрытых семантических связей. Данные связи являются ассоциативными. Согласно исследованиям, выявление ассоциативных связей широко распространено у детей до семи лет, что оказывает влияние на их когнитивное развитие. [27]

Однако для выявления значимых фактов из текста недостаточно выделения ассоциативных отношений между словами. Требуется определение функциональных отношений – отношений, а также лингвистических отношений – гиперонимии, меронимии, антонимии и некоторых других встречающихся реже.

Поэтому основной проблемой является формализация представления текстов. Для этого могут использоваться синтаксические деревья, семантические модели представления, в частности, семантические графы. Помимо семантических сетей

для формализации могут использоваться продукционные модели, модели фреймов, концептуальных графов [28]. Также текст на естественном языке может быть приведен, к некоторому искусственному языку, упрощенному для формализованной обработки. Например, к языку ситуационного управления [29, С. 55–63].

Однако в любом случае для построения описанных моделей, требуется провести структурный анализ естественного языка. Что приводит к основным проблемам анализа текстов – омонимии, большой вариативности семантически эквивалентных текстовых представлений и их приведению к эквивалентным семантическим моделям.

Проблема омонимии с точки зрения семантического анализа шире описанной выше проблемы представления омонимичных концептов в семантических сетях. Существуют омонимичные формы морфологически различных слов. Например, слово «стать», одновременно является глаголом в повелительном наклонении и существительным в именительном падеже. Однако данные слова необходимо отличать на этапе синтаксического и семантического анализа текста, так как неправильное распознавание морфологических свойств текста, сильно сказывается на результате анализа.

Проблема вариативности описаний на естественных языках может быть решена алгоритмами машинного обучения. Что порождает необходимость подготовки большого количества данных для выполнения обучения модели алгоритма.

В качестве данных для обучения, как правило, используются различные корпусы текстов. Корпус выбирается в зависимости от поставленных задач семантического разбора. Однако для большинства задач подходят корпусы языка, в которых представлены самые различные виды текстов.

Основной проблемой корпусов являются проблемы представительности – то есть корпус должен быть основан на текстах различных жанров, различной тематики и лексики, причем представительство того или иного жанра, тематики и лексики должно быть пропорционально ее применению носителями языка. [30]

Другой проблемой корпусов является соблюдение авторского права, которая также связана с проблемой представительности. Так как распределение текстов, защищенных авторским правом, является неравномерным относительно жанров, тематик или используемой лексики. Так художественная литература, в отличии от научно-публицистической, не находится в свободном доступе в необходимом количестве. [31]

Важной задачей при построении корпуса является выполнение корпусной разметки, при выполнении задачи должна также решаться проблема морфологической неоднозначности. Морфологическая неоднозначность сначала снимается автоматически при помощи структурных алгоритмов, основанных на словарях и грамматических правилах языка. Однако этого недостаточно для полного снятия морфологической неопределенности, поэтому дополнительно применяется автоматизированные подходы с большим количество ручной работы.

Для возможности использования корпусов для машинного обучения при этом требуется очень большое представительство в корпусе самых редко используемых слов и словоформ для корректной работы алгоритма с ними.

2.4 Обзор существующих сетей и систем автоматического анализа текстов на естественных языках

WordNet как тезаурус считается стандартом в Европе и англоязычном мире. Большинство рассмотренных в работе научных публикаций ссылается на данный тезаурус.

Базовой единицей WordNet является синсет. Синсет представляет собой множество слов, связанных отношением синонимии, с описанием применимости данного синонима в различных формах. WordNet – это сеть, узлами которой являются синсеты. Другими словами, узлы WordNet представляют множество единиц языка, разделенного на классы эквивалентности, отражающие определенную сущность.

В Wordnet описаны следующие типы отношений между узлами:

- Гипероним (гипоним);
- Has-member (member-of);
- Мероним (холоним);
- Антоним;
- И другие.

WordNet является многоязыковым тезаурусом и содержит словари множества языков: английский, испанский, итальянский, немецкий, французский и прочие.

Синсет в WordNet содержит также описание части речи отдельного концепта. Одно и то же слово может появляться в различных синсетах.

Представим синсет, описывающий концепт «play» в качестве театральной игры. [17].

$\{ play_n^1, drama_n^1, dramatic\ play_n^1 \}$

В синсетах нижний индекс (например, n для синсета, представленного выше) означает часть речи (n – noun, существительное), а верхний индекс – «number sense» – частоту использования соответствующего токена. Причем 1 – наиболее часто используемые. Для каждого синсета описывается текстуальное представление. Например, play – «a dramatic work for performance by actors on a stage».

Одним из компонентов WordNet является Interlingual Index (ILI), реализация которого для тезауруса на конкретном языке, позволяет связать его с остальными языками, представленными в WordNet. Для этого каждый синсет WordNet, определенный для национального языка должен иметь, отношение как минимум с одной записью в ILI. Синсеты из разных языков, связанные с одной и той же записью ILI являются эквивалентными на разных языках.

На рисунке 2.2. представлен скриншот веб-интерфейса WordNet

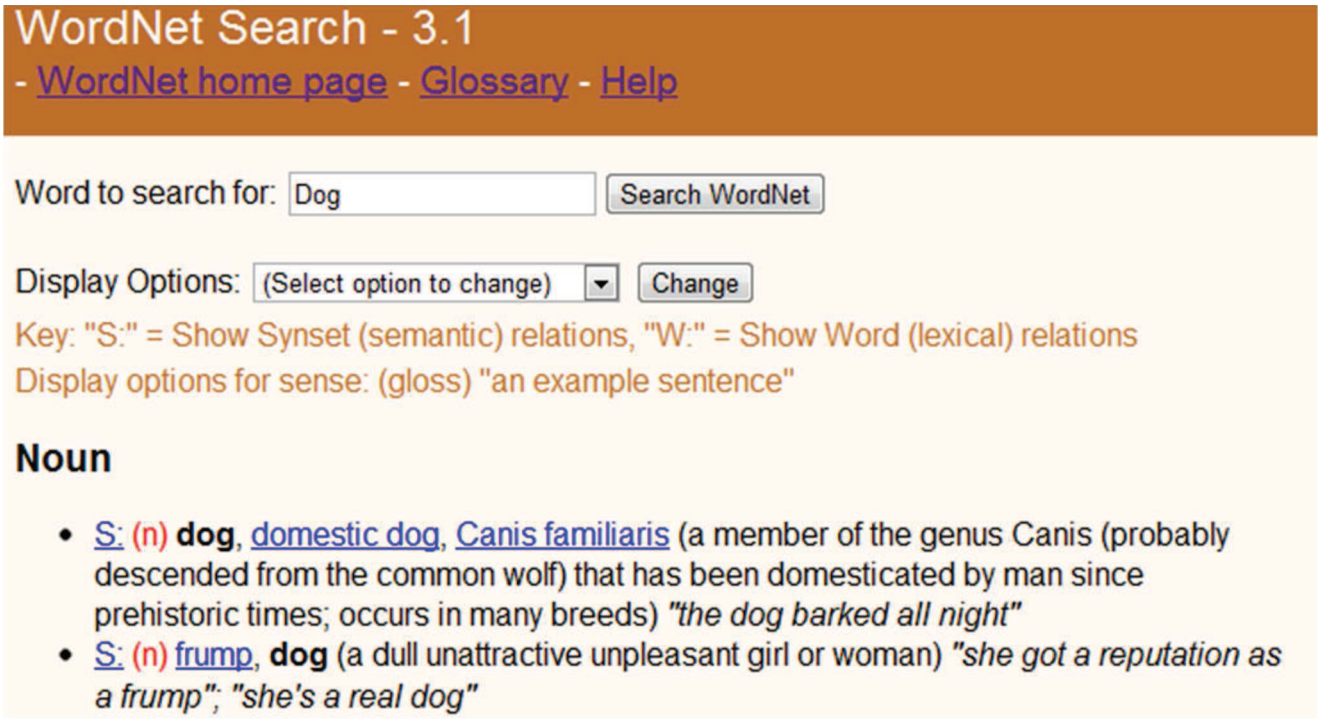

Рисунок 2.2 – Веб-интерфейс WordNet

Одним из самых распространенных тезаурусов в России является РуТез. РуТез разработан только для русского языка. Описывает самое большое количество слов и выражений по отношению к другим аналогичным русскоязычным проектам.

Словарь РуТеза состоит из множества понятий, описывающие классы сущностей современного мира. Каждому понятию соотносится набор слов и словосочетаний. Набор слов или словосочетаний называется текстовым входом. Слова или словосочетания, относящиеся в одному понятию, называются онтологическими синонимами. В отличии от синсетов понятие должно иметь однозначное имя, построенное на основе слов и словосочетаний, входящих в данное понятие.

В РуТез определены следующие типы отношения:

- Гипоним-гипероним, или в терминологии РуТез родовидое отношение «ниже – выше»
- Мероним-холоним. Причем данная связь не может отражать отношение агрегации.

- Отношение несимметричной ассоциации, когда два понятия между собой связаны, так как одно понятие не может существовать без другого. Например, от кипения к жидкости является несимметричной ассоциацией.
- Отношение симметричной ассоциации, или предсинонимия.

Существует веб-версия тезауруса, а также xml-файлы с данными тезауруса, которые могут быть использованы для бесплатного некоммерческого использования. Лицензия позволяет копировать, изменять и некоммерчески использовать РуТез-lite версию тезауруса. [2]

РуТез имеет отличную от WordNet структуру и не связан с ним, что некоторые отмечают как минус данного тезауруса. [21]

Веб-интерфейс тезауруса РуТез представлен на рисунке 2.3.

Данного минуса нет у тезауруса RussNet, который является русской реализацией WordNet. Однако последние публикации, связанные с ним, были в 2002 году, а новости на сайте в 2003. Из чего можно сделать вывод, что проект не развивается. [22]

YARN (Yet Another RussNet) – справочник, построенный на основе WordNet 3.0. Таким образом YARN содержит ILI идентификатор для связи с сетями на других языках, однако реализации данного идентификатора нет. Тезаурус YARN формируется на основе краудсорсингового подхода, то есть любой желающий может редактировать справочник слов, синсетов и отношений через веб-интерфейс.

Соответственно, YARN имеет почти эквивалентную WordNet структуру тезауруса, т.е. тезаурус в основном состоит из синсетов и связей между ними. Однако YARN также содержит орфографический словарь, который содержит список слов, с минимальным количеством дополнительной информации. В листинге 2.1 представим пример описания слова в формате YARN.

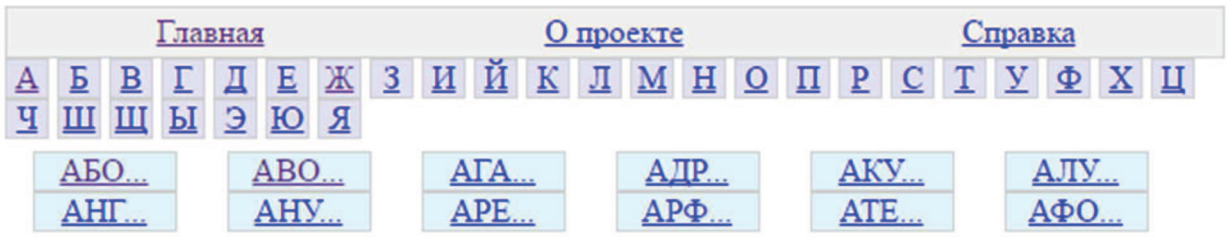

Рисунок 2.3 – Веб-интерфейс РуТез

Листинг 2.1 – Описание слова в формате YARN [23]

```
<wordEntry id="n123" approvedBy="mfly"
        approvedWhen="2012-12-26T17:14:00Z" author="pb"
        version="2" timestamp="2012-12-26T17:12:00Z">
    <word>престидижитатор</word>
    <grammar>1a</grammar>
    <accent>12</accent>
    <url>http://ru.wiktionary.org/wiki/престидижитатор</url>
</wordEntry>
```

Как можно убедиться по листингам 2.1 и 2.2 в основе формата описания слов и синсетов лежит формат XML.

А в листинге 2.2 представим пример описания синсета в формате YARN.

Листинг 2.2 – Описание синсета в формате YARN [23]

```
<synsetEntry id="sn1" approvedBy="olesar"
  approvedWhen="2012-12-26T17:14:00Z" author="bonch"
  version="2" timestamp="2012-12-25T17:12:00Z">
  <word ref="n123">
    <mark>устар</mark>
    <sample url="http://artbrus.livejournal.com/204315.html">
Лучший фокус, когда престидижитатор (так называют фокусника,
манипулятора с вещами) начинает вынимать шар, то изо рта, то из уха,
то из-за шиворота обалдевшего зрителя.
    </sample>
    <sample source="В. В. Крестовский, 'Петербургские трущобы',
    1867 г., НКРЯ">
Мечут же карты, передёргивают и всякие иные фокусы употребляют только
главные и самые искусные престидижитаторы, которые поэтому специально
называются «дергачами".
    </sample>
  </word>
<word ref="n5678"/> <!--манипулятор-->
  <definition
    url=http://ru.wiktionary.org/wiki/престидижитатор
    source="wiktionary">
фокусник, отличающийся ловкостью рук;манипулятор
  </definition>
  <definition source="Толковый словарь Ушакова">
Фокусник с большой быстротой и ловкостью рук
  </definition>
</synsetEntry>
```

Веб-интерфейс YARN представлен на рисунке 2.4

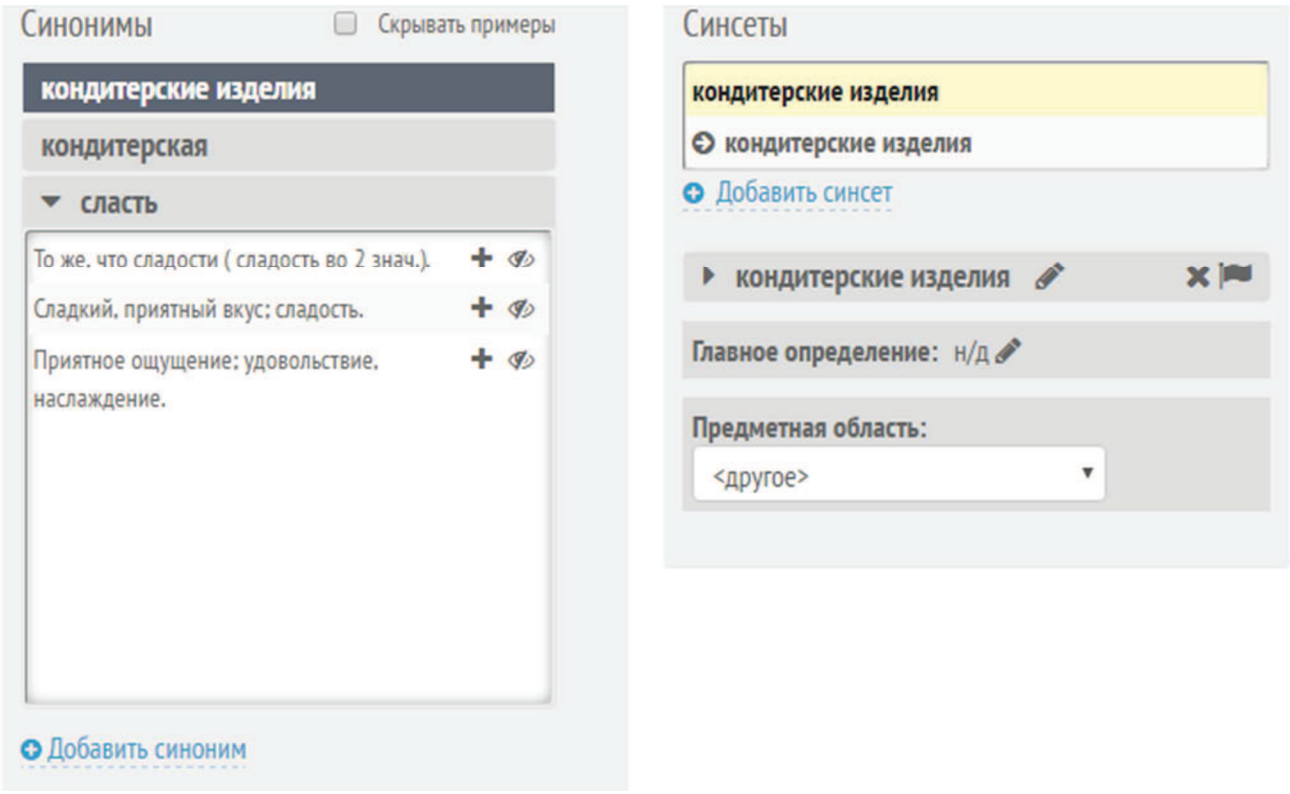

Рисунок 2.4 – Веб-интерфейс YARN

ConceptNet – многоязычная семантическая сеть. Источником пополнения сети является краудсорсинг, т.е. любой человек может внести вклад в развитие сети, при помощи внесения фактов через веб-сайт, также сеть пополняется при помощи средства извлечения связанных данных из статей Википедии, разработанной в рамках проекта DBPedia. Часть данных извлечено из Викисловаря, WordNet'a. [24]

Центральным понятием для ConceptNet является URI. На основе URI определяются типы концептов и отношений, идентифицируется язык и многое другое.

Например, /c/ – концепты, или /r/ – отношение, независимое от языка (например, «класс-подкласс» или в терминологии ConceptNet «IS-A»), или /en/ – на английском языке. Следовательно, /c/en/play – это концепт «играть» на английском языке.

ConceptNet имеет гиперграфовую структуру. Гиперграф формируется благодаря понятию «edge». Edge может содержать любой URI и может быть связан либо с другим подграфом, либо с отдельным концептом. Т.е. в виде узла графа может быть представлено, что угодно: концепт, связь либо выражение.

На рисунке 2.5 представлен веб-интерфейс ConceptNet.

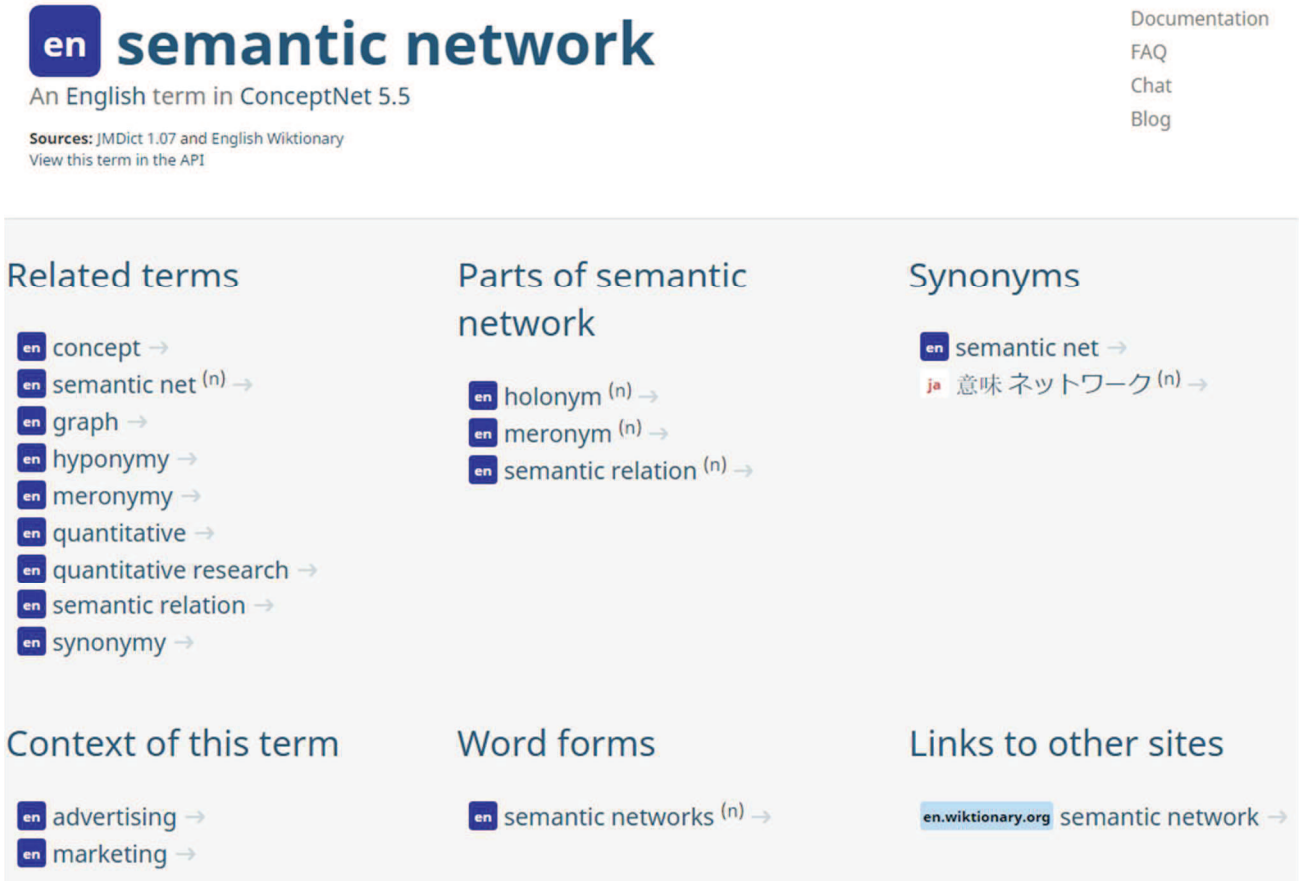

Рисунок 2.5. – Веб-интерфейс ConceptNet

В листинге 2.3 представлен формат описания отношения «Example Used For Explain».

Листинг 2.3 – Формат описания отношения.

```
{
  "@id": "/a/[/r/UsedFor/,/c/en/example/,/c/en/explain/]",
  "dataset": "/d/conceptnet/4/en",
  "end": {
    "@id": "/c/en/explain",
    "label": "explain something",
    "language": "en",
    "term": "/c/en/explain"
  },
  "license": "cc:by/4.0",
  "rel": {
    "@id": "/r/UsedFor",
```

Продолжение листинга 2.3.

```
      "label": "UsedFor"
    },
    "sources": [
      {
        "activity": "/s/activity/omcs/omcs1_possibly_free_text",
        "contributor": "/s/contributor/omcs/pavlos"
      }
    ],
    "start": {
      "@id": "/c/en/example",
      "label": "an example",
      "language": "en",
      "term": "/c/en/example"
    },
    "surfaceText": "You can use [[an example]] to [[explain
something]]",
    "weight": 1.0,
    "@context": [
      "http://api.conceptnet.io/ld/conceptnet5.5/context.ld.json",
      "http://api.conceptnet.io/ld/conceptnet5.5/pagination.ld.json"
    ]
  }
```

Выводы по разделу два

В пункте 2.1. были даны определения основных терминов, используемых в работе: семантическая сеть, тезаурус, корпус, синсет, а также лингвистическим типам отношений: гипоним, гипероним, холоним, мероним.

В пункте 2.2 был проведен обзор литературы и публикаций, представленные алгоритмы были обобщены в группы: вероятностные алгоритмы, алгоритмы анализа слабоструктурированных текстов, алгоритмы полного лексического

анализа. Также было выделено машинное обучение, как метод, используемый в качестве вспомогательного для любой группы алгоритмов.

В пункте 2.3 были описаны существующие проблемы в области представления знаний в виде семантического графа, а также в области анализа текстов на естественных языках, в частности проблемы омономии. Были описаны случаи полной омонимии и морфологической омонимии. Также были рассмотрены модели представления межпредментых понятий и проблемы транзитивности и множественного наследования в семантических графах.

В пункте 2.4. был проведен обзор существующих семантических систем, как российских, так и зарубежных разработок. Были описаны основные особенности семантических графов WordNet, РуТез, YARN, ConceptNet. Концептуально были расмотрены методы их формирования.

3 ПРОЕКТНАЯ ЧАСТЬ

3.1 Описание требований

Необходимо разработать описательную обучаемую семантическую сеть, отображающую основные лингвистические отношения. Алгоритм обучения основывается на анализе текстов на естественном языке.

Опишем функциональные требования системы.

Структуру сети будем создавать по примеру передовых российских и зарубежных разработок. Все рассмотренные тезаурусы в разделе два описывают основные иерархические отношения, т.е. «гипероним-гипоним», либо применительно к компьютерным наукам «класс-подкласс» и «холоним-мероним», либо «часть-целое». Рассмотренные тезаурусы отображают в основном обобщенные категории и реже отображают конкретные экземпляры, поэтому на

данном этапе разработки системы отношение «класс-экземпляр» рассматривать не будем.

В сети должны храниться концепты, представленные не только отдельными словами, но и концепты, представленные различными словоформами. Сеть должна отображать, помимо остальных, межпредметные понятия.

Требуется разработка внешнего API для возможности получения отдельных концептов сети, а также возможности расширения графа за счет сторонних и внутренних разработок. Необходимо разработать базовый веб-интерфейс для отображения концептов и основных связанных с ним понятий. Разработку требуется вести с учетом необходимости расширения возможностей отображения семантического графа в следующих версиях системы.

Сеть может отражать ассоциативные взаимосвязи между отношениями, также данное требование должно быть учтено при проектировании API.

Отношения графа должны иметь весовые коэффициенты для отображения коэффициента достоверности знаний, содержащихся в графе.

Алгоритм автоматического расширения семантической сети на основе анализа текстов на естественном языке должен создавать концепты и отношения между ними различных типов. Требований по скорости работы алгоритма нет.

Требований к безопасности и правам доступа на данном этапе разработки нет.

Особых требований к квалификации пользователя системы нет.

Особых требований к аппаратному и программному обеспечению нет.

В качестве языка разработки будет использоваться C#. Используемые внешние библиотеки и фреймворки должны быть максимально свободными, как для некоммерческих, так и для коммерческих целей.

Семантическая сеть может хранить иные типы отношений, или особые служебные концепты и отношения между ними, необходимые для выполнения анализа текстов на естественном языке.

3.2 Структура сети

В качестве функциональных требований описан список различных типов отношений, что является прямым указанием на мультиграфовую структуру сети. Так как основные лингвистические отношения имеют направление, то граф ориентированный.

Согласно функциональным требованиям концептами семантической сети могут быть различные словоформы. В пункте 2.3 описано решение для отображения таких концептов в виде графа. Следовательно, концептом сети может выступать граф. Значит, семантическая сеть должна иметь структуру гиперграфа.

Таким образом, семантический граф можно охарактеризовать как:

– мультиорграф, т.е. ориентированный граф с кратным числом ребер, связывающих два узла.

– гиперграф, т.е. граф, в котором каждым ребром могут соединяться не только две вершины, но и любые подмножества вершин.

Необходимые отношения исходя из функциональных требований.

1. Гипоним-гипероним (или класс-подкласс).
2. Мероним-холоним (или часть-целое).
3. Ассоциативное отношение.

Для описания гиперграфовой структуры в качестве узла-концепта может выступать подсеть, описанная текстом, например, «Координата PART-OF География» или «Географическая координата», как одна из возможных форм описания концепта.

Все отношения графа имеют весовые коэффициенты для отображения семантической близости понятий.

Терминология. Концепт – слово или словоформа, представленная в виде узла графа. Словоформа может определяться через подграф. Отношение – связь между концептами, имеющая тип и весовой коэффициент. Между двумя одинаковыми концептами может быть несколько отношений.

3.3 Общая архитектура решения

Для уменьшения сложности разработки и внесения изменений в различные компоненты системы, для выполнения требования обеспечения расширяемости системы другими компонентами формирования семантической сети разделим разрабатываемую систему на структурные блоки – модули.

Основные особенности модулей:

1. Один модуль не может иметь доступ к программному коду другого модуля.
2. Модули могут использовать общие внешние зависимости.
3. Общая логика между модулями выделяется в общие внешние сборки
4. Каждая внешняя сборка имеет спецификацию и разрабатывается по принципу открыт – закрыт [32]
5. Взаимодействие между модулями выполняется через их публичный REST API.
6. Каждый модуль имеет собственную доменную модель и хранилище данных.
7. Никакой модуль не имеет доступа к хранилищу данных другого модуля.

Архитектура системы представлена на рисунке 3.1.

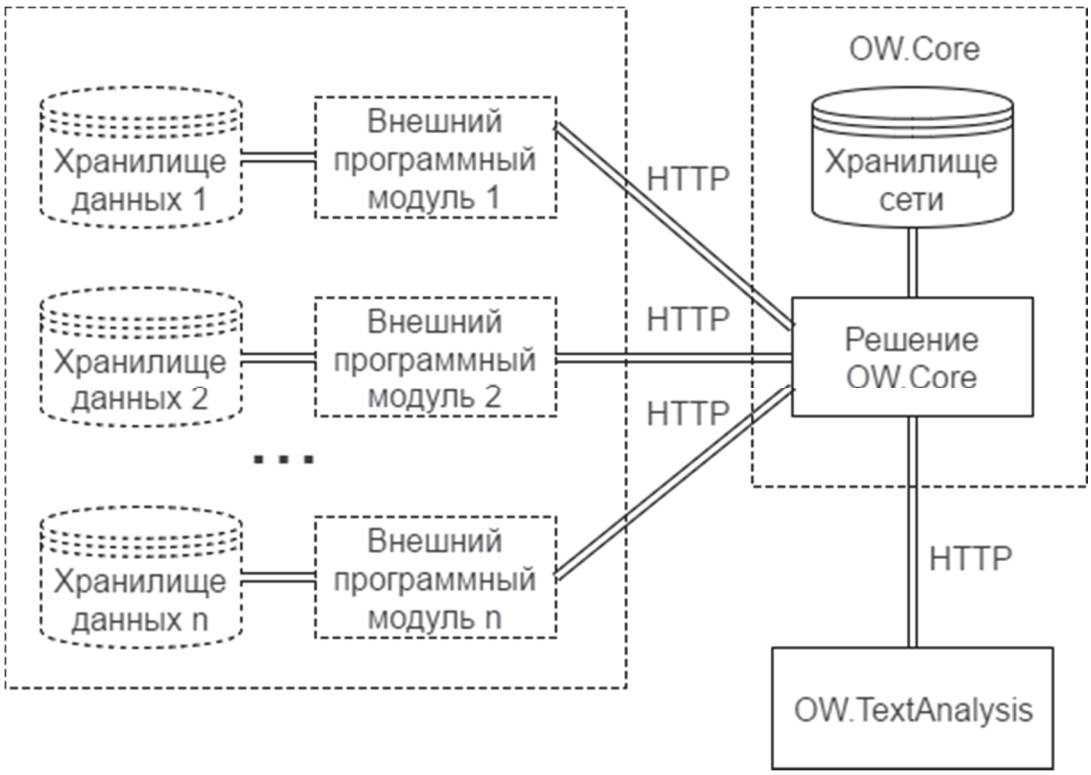

Рисунок 3.1 – Архитектура системы

На данном этапе требуется разработать два модуля: модуль хранения и представления семантической сети – OW.Core и модуль автоматического анализа текста на естественном языке для формирования семантической сети – OW.TextAnalysis.

3.4 Описание применяемых технологий

Используемый язык разработки C#, объектно-ориентированный язык программирования. Основой языка является объектно-ссылочная модель (все типы делятся на значащие и ссылочные). Язык программирования также поддерживает функциональную парадигму программирования и склоняет к одновременному использованию объектно-ориентированной и функциональной парадигмы программирования.

Язык C# имеет сборщик мусора и работает на виртуальной машине, т.е. требует установленного приложения поверх операционной системы. На сегодняшний день существует для Windows .Net Framework, для Unix-подобных систем Mono. Также разрабатываются общие стандарты для различных программных платформ – .Net Standards и .Net Core, позволяющие без проблем собирать и запускать одно приложение на различных программных платформах.

Приложение OW.Core для хранения и представления семантической сети является веб-приложением. Серверная часть разрабатывается на основе ASP.Net Core и представляет собой Web API. Следовательно, данное приложение является полностью кроссплатформенным. То есть модуль OW.Core может быть развернут на операционных системах Windows и Linux, а также на иных системах, имеющих поддержку ядра .Net Core. Однако большинство современных хостингов предлагают в качестве программной платформы ОС семейства Linux, реже Windows.

Клиентское приложение построено на принципах одностраничных веб-приложений (SPA – Single Page Application). Клиентское приложение

разрабатывается на языке программирования TypeScript с использованием легковесного фреймворка Vue.JS.

Vue.JS используется для разбиения клиентской веб-страницы на отдельные компоненты, что упрощает навигацию по клиентскому приложению и позволяет использовать одни и те же компоненты и логику их работы в различных частях клиентского приложения.

В качестве фреймворка для верстки и стилизации приложения используется фреймворк Vuetify. При помощи Vuetify разрабатывается внешний вид главного окна приложения, меню, диалоговых окон и других компонентов приложения. В качестве языка разметки используется HTML, для описания стилистики страницы – Stylus, препроцессорный язык над CSS.

Клиент-серверное взаимодействие выполняется на основе REST API и протокола HTTP.

На текущем этапе модуль OW.TextAnalysis представляет собой простое ASP.Net MVC приложение с формой ввода текста и кнопкой отправки текста для сохранения в сеть.

OW.TextAnalysis зависит от внешней библиотеки SDK Pullenti, которая написана с использованием .Net Framework. Следовательно, OW.TextAnalysis также может быть реализован только на основе этой платформы.

При разработки всех модулей приложения используются подходы «Разработка, управляемая поведением» (BDD), «Проблемно-ориентированное проектирование» (DDD). [34]

Автоматизированные тесты написаны на основе фреймворка xUnit. В качестве изоляционного фреймворка для написания модульных тестов используется NHibernate. Также для повышения читаемости тестов, и как следствие, для поддержки подхода BDD, а также для ускорения написания тестов используется Fluent Assertions. [35]

При написании программной реализации активно используется принципы внедрения зависимостей, описанные Марком Симаном в книге «Внедрение зависимостей в .NET». [36]

3.5 Разработка модуля хранения и отображения сети

Для хранения семантической сети будем использовать графовую СУБД Neo4j, одну из самых распространенных графовых СУБД – Neo4j, позволяет визуализировать граф, имеет собственный язык запросов и манипулирования данными Cypher, поддерживает ACID, имеет библиотеки, реализованные для различных языков программирования, включая C#.

Приложение имеет трехзвенную архитектуру. Клиентское приложение выполняет презентационную логику, серверное приложение содержит логику обновления хранилища, а также логику извлечения данных из нее.

Структура проектов решения программного компонента модуля представлена на рисунке 3.2

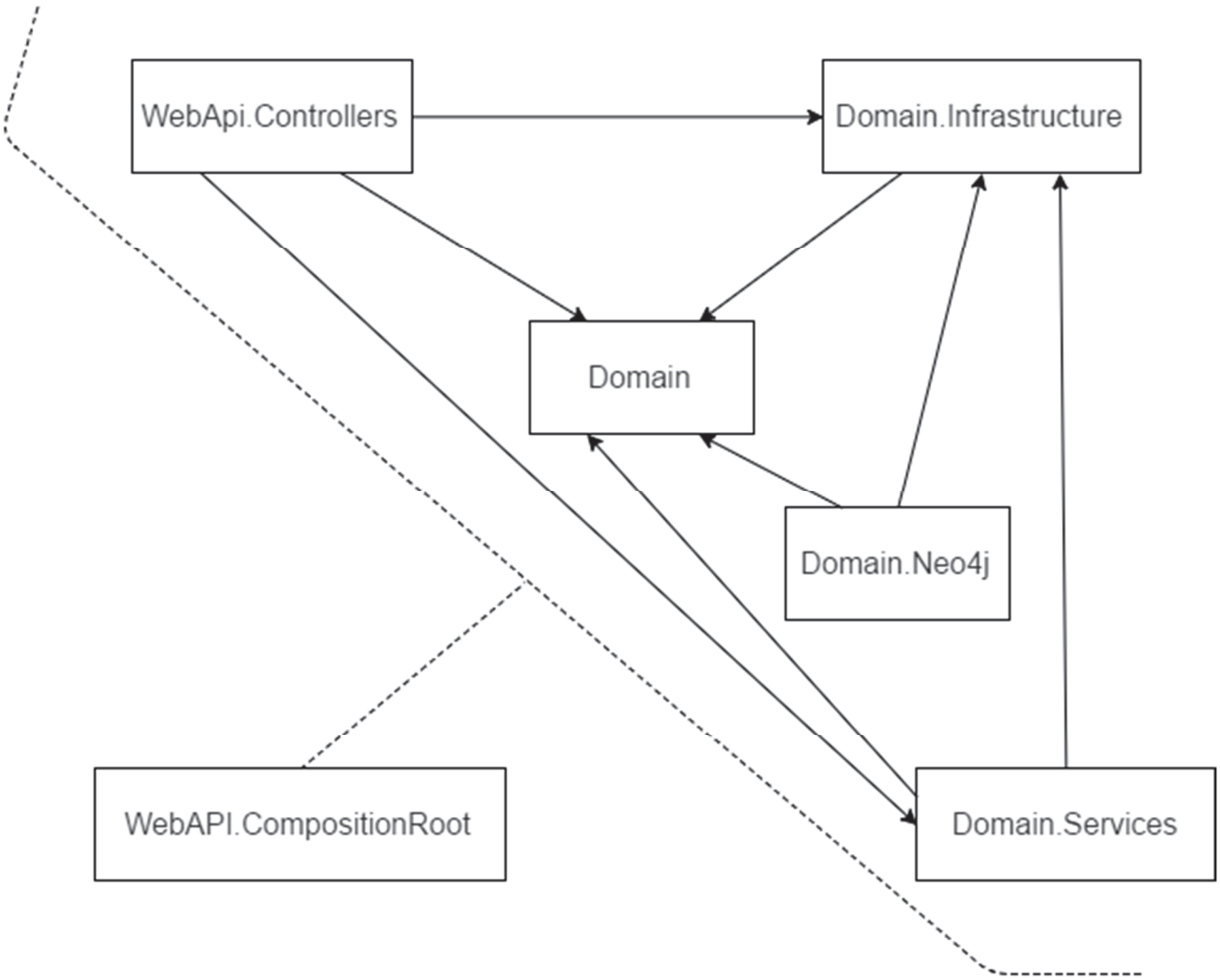

Рисунок 3.2 – Структура проектов решения модуля

Все внешние модули системы должны быть предварительно зарегистрированы в модуле OW.Core, каждому модулю присваивается коэффициент достоверности от 0 до 100. Никакие действия внешних модулей с коэффициентом достоверности равным нулю не оказывают влияния на семантическую сеть.

Модуль идентифицируется за счет токена, передаваемого во всех запросах на изменение данных. Токен является глобальным уникальным идентификатором (GUID). Вопросы защиты токена не рассматриваются в работе.

В таблице 3.1 представлен Web API данного модуля

Таблица 3.1 – API методы модуля

Название метода	Тип	Путь	Пример запроса
GetConcepts	GET	[domain]\api\net\getconcepts	[domain]\api\net\getconcepts?concepts=География&concepts=Координата
MergeConcepts	POST	[domain]\api\net\mergeconcepts	–

Метод GetConcepts выполняет поиск по концептам в сети, и если запрашиваемые концепты найдены, выводит их в формате, представленном в листинге 3.1.

Листинг 3.1 – Формат ответа на запрос GetConcepts.

```
[{
    "concept":"Координата",
    "ingoingRelations":
    [
        {
            "sourceLemma":"Координата PART-OF География",
            "destinationLemma":"Координата",
            "type":"IS-A",
            "weight":0.8
        }
    ],
    "outgoingRelations":
    [
        {
            "sourceLemma":"Координата",
            "destinationLemma":"География",
            "type":"PART-OF",
            "weight":0.8
        }
    ]
},{
    "concept":"Координата PART-OF География",
    "ingoingRelations":
```

Продолжение листинга 3.1

```
    [
        {
            "sourceLemma":"Координата PART-OF География",
            "destinationLemma":"Координата",
            "type":"IS-A",
            "weight":0.8
        }
    ],
    "outgoingRelations":[]
},{
    "concept":"География",
    "ingoingRelations":
    [
        {
            "sourceLemma":"Координата",
            "destinationLemma":"География",
            "type":"PART-OF",
            "weight":0.8
        }
    ],
    "outgoingRelations":[]
}]
```

Метод MergeConcept выполняет слияние концептов, полученных извне с концептами, хранящимися в сети. Представим алгоритм слияния ниже.

1 Если сеть не содержит концепта, пришедшего в запросе на слияние, то добавляется новый полностью эквивалентный пришедшему в запросе концепт

2 Если сеть содержит концепт, пришедший в запросе, то происходит слияние его отношений.

2.1.1 Эквивалентные поля копируются без изменений.

2.1.2 Отношения, которых нет в сети, добавляются.

2.1.3 Для отношений, которые есть в сети происходит слияние.

2.1.3.1 Эквивалентные поля копируются без изменений.

2.1.4 В список источников, формирующих данное отношение, добавляется соответствующий источник, из которого пришел запрос на слияние концептов.

2.1.4.1 Происходит подсчет весового коэффициента по формуле нахождения среднего взвешенного (Формула 3.1).

$$X = \frac{\sum_{s=1}^{n} w_s \cdot x_s}{\sum_{s=1}^{n} w_s}, \qquad 3.1$$

где X – искомый вес отношения графа,

s – порядковый номер источника,

n – количество источников,

w_s – коэффициент доверия источника,

x_s – значение весового коэффициента.

Эквивалентными концептами считаются концепты, имеющие одно и тоже строковое представление (примеры строковых представлений, «Координата» и «Координата PART-OF География»). Эквивалентными отношениями считаются отношения, связывающие эквивалентные краевые концепты и имеющие эквивалентные типы. Типы считаются эквивалентными, если имеют одинаковое имя («IS-A» или «PART-OF», например).

MergeConcepts – POST метод, поэтому путь и пример использования совпадают. Однако для POST метода необходимо заполнять тело запроса (Body). Пример тела запроса представлен в листинге 3.2

Листинг 3.2 – Формат тела запроса MergeConcepts.

```
[
    "token":"53ea9c69-dbc6-4c25-a90b-030e448d7403",
    "concepts":
    {
        "concept":"Координата",
        "ingoingRelations":
        [
            {
                "sourceLemma":"Координата PART-OF География",
                "destinationLemma":"Координата",
                "type":"IS-A",
                "weight":0.8
            }
        ],
        "outgoingRelations":
        [
           {
                "sourceLemma":"Координата",
                "destinationLemma":"География",
                "type":"PART-OF",
                "weight":0.8
            }
        ]
    }
]
```

Представим диаграмму классов доменной модели на рисунке 3.3

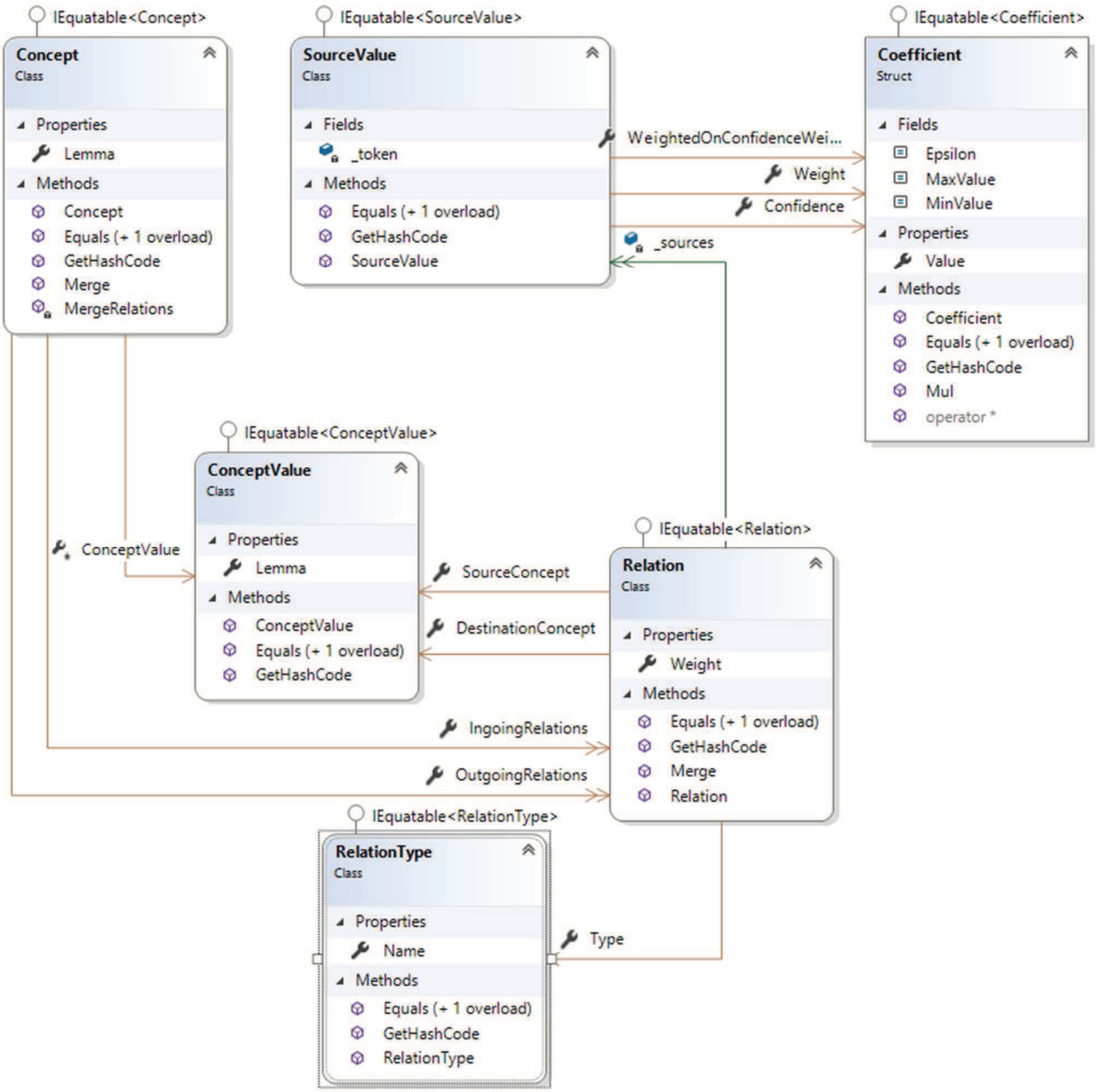

Рисунок 3.3 – Доменная модель семантической сети

Исходный код данного модуля представлен в приложении А.

3.6 Формирование семантической сети на основе анализа текстов

Укрупненный алгоритм формирования семантической сети на основе анализа текстов представлен на рисунке 3.4.

Рисунок 3.4 – Укрупненный алгоритм

Ниже кратко описан каждый элемент алгоритма.

1. Токенизация.

Токенизация – задача выделения различных блоков из текста. Как правило, токенизация предполагает выделение границ предложения и слов внутри этого предложения.

Однако при токенизации может потребоваться иная сегментация текста, разделение его на более крупные блоки, чем предложения – на абзацы, параграфы, разделы и прочее.

Возможными проблемами токенизации могут оказаться пропущенные точки, пробелы, грамматические и синтаксические ошибки.

В данной работе не рассматривается обработка ошибочных с точки зрения грамматики русского языка текстов.

Поэтому сегментизация на предложения выполняется так:

1. Окончание предложения выделяется на знаках «.», «!», «?» в случае, если первое за ним слово может являться началом следующего предложения (начинается с большой буквы).
2. Убрать признак окончания предложения, если точка идет после известных сокращений (ул., пр., т.д., проф., доц. и т.д.)
3. Предложение не может начинаться и заканчиваться в разных абзацах
4. Предложение не может начинаться за кавычками и заканчиваться внутри кавычек и наоборот.

В идеале известные сокращения должны быть описаны в семантической сети.

А токенизация на отдельные слова и символы производится на основе пробельных символов.

Результатом токенизации в разрабатываемом алгоритме является поток предложений, каждый из которых разделен на отдельные токены.

Исходя из вышесказанного приведем пример токенизации.

Исходный текст. «Токенизация – задача выделения различных блоков из текста. Как правило, токенизация предполагает выделение границ предложения и слов внутри этого предложения.»

Результат.

Первое предложение: «Токенизация – задача выделения различных блоков из текста.». Предложение состоит из токенов: «Токенизация», «–», «задача», «выделения», «различных», «блоков», «из», «текста», «.».

Второе предложение: «Как правило, токенизация предполагает выделение границ предложения и слов внутри этого предложения.». Предложение состоит из токенов: «Как», «правило», «,», «токенизация», «предполагает», «выделение», «границ», «предложения», «и», «слов», «внутри», «этого», «предложения».

2. Морфологический, морфемный анализ и лемматизация.

Морфологический анализ заключается в выделении таких признаков, как часть речи, число, род, падеж, время, склонение, спряжение и прочие в русском языке.

Морфемный анализ выделение основных частей слова – для русского языка – окончания, суффикса, корня, приставок, основа слова. На основе морфемного анализа строится лемма слова.

Одной из проблем морфологического анализа является омонимичность, как морфологическая, так и семантическая. Т.е. абсолютно эквивалентные слова могут иметь не только различное значение, но и состоять из отличающихся морфем, а также иметь отличающиеся морфологические характеристики.

Например, слово «гладь»:

– глагол в повелительном наклонении; существительное в именительном падеже единственном числе,
– существительное в винительном падеже единственном числе и т.д.

Или слово «коса»:

– коса, как инструмент;

– коса, как прядь волос;

– коса, как береговая линия.

То есть результатом морфологического анализа будет список всех возможных результатов морфемных и морфологических характеристик каждого токена.

Исходя из выполненного разбора, все данные кэшируются в морфологический словарь русского языка, в котором хранятся различные формы слов, с описанием их части речи, числа, рода, падежа, времени и прочего.

Результат морфологического разбора предложения «Убежищем потерпевших поражение в гражданской войне с коммунистами сторонников партии Гоминьдан стал островной Китай.» показан в таблице 3.1.

Таблица 3.2 – Результат морфологического разбора

Словоформа	Лемма	Морфологические варианты
Убежищем	Убежище	УБЕЖИЩЕ существ. ед.ч. ср.р. творит.
		УБЕЖИЩ собств.|фамилия ед.ч. муж.р. творит. (? 1)
потерпевших	Потерпевший	ПОТЕРПЕВШИЕ\ПОТЕРПЕВШИЙ существ. мн.ч. жен.р. родит.|винит.|предлож.
		ПОТЕРПЕВШИЕ\ПОТЕРПЕВШИЙ существ. мн.ч. муж.р. родит.|винит.|предлож. одуш.
		ПОТЕРПЕВШИЕ\ПОТЕРПЕВШИЙ прилаг. мн.ч. родит.|винит.|предлож.
		ПОТЕРПЕТЬ глагол мн.ч. родит.|винит.|предлож. п.вр. сов.в. дейст.з.
поражение	Поражение	ПОРАЖЕНИЕ существ. ед.ч. ср.р. именит.|винит. синоним.форма
в	В	В предлог
гражданской	Гражданский	ГРАЖДАНСКАЯ\ГРАЖДАНСКИЙ прилаг. ед.ч. жен.р. родит.|дател.|творит.|предлож.
войне	Война	ВОЙНА существ. ед.ч. жен.р. дател.|предлож.
с	С	С предлог
коммунистами	Коммунист	КОММУНИСТЫ\КОММУНИСТ существ. мн.ч. муж.р. творит. одуш.
сторонников	Сторонник	СТОРОННИКИ\СТОРОННИК существ. мн.ч. муж.р. родит.|винит. одуш.
партии	Партия	ПАРТИЯ существ. ед.ч. жен.р. родит.|дател.|предлож.
		ПАРТИИ\ПАРТИЯ существ. мн.ч. жен.р. именит.|винит.
Гоминьдан	Гоминьдан	ГОМИНЬДАН существ. ед.ч. муж.р. именит.|винит.
стал	Стать	СТАТЬ глагол ед.ч. муж.р. п.вр. сов.в. 1 л. 2 л. 3 л.

5. Графематический анализ.

Элементами графематического анализа является рассмотренный на этапе токенизации алгоритм разбиения на предложения, слова, разделители и прочее.

Также сюда относят выделение абзацев, заголовков, разделов, параграфов и другого.

В рамках представленной работе графематический анализ будет состоять в выделении составных токенов, в частности:

– чисел;

– дат;

– электронных адресов;

– ФИО;

– др.

Объединение токенов во время проведения графематического анализа происходит на основе словарей числительных, анализа римских цифр, инициалов, словарей единиц измерения и прочего.

Для морфологического анализа и частично для графематического анализа можно использовать сторонние решения, представленные в таблице 3.2.

Таблица 3.3 – Решения для морфологического анализа

Название	Метод	Лицензия	ЯП / платформа
AOT	Структурный	LGPL	C++ / Linux, Windows
Pymorpy	Структурный	MIT	Python / Python Interpreter
Tree Tagger	Структурный	MIT	C++ / Linux, Windows
Zamgi	Машинное обучение	Некоммерческая	C# (C++) / .Net Framework, Mono
Mystem-scals	Структурный	MIT	Java / Linux, Windows
Russian Morphology	Структурный	Apache License	Java / Java Virtual Machine
Pullenti SDK	Структурный	Бесплатно для некоммерческого использования	C# / .Net Framework, Mono

Согласно требованиям, необходимо решение с максимально открытой лицензией, имеющее API или библиотеку на языке C#. Исходя из этих требований лучше всего подходит Zamgi и Pullenti SDK. Если предположить написание собственной обертки над решениями на других языках, то наиболее популярными решениями являются Tree Tagger, AOT, Pymorphy.

Однако Pullenti SDK демонстрирует хорошие результаты морфологического анализа, имеет в своем составе множество словарей, которые помимо морфологического анализа оказываются полезными для графематического анализа. Также библиотека Pullenti SDK упрощает синтаксический анализ предложения, так как выполняет его частично.

Недостатком библиотеки Zamgi же является метод машинного обучения, который дает результаты со снятой омонимичной напряженности, однако полностью снять омонимичную напряженность на этапе морфологического анализа невозможно, так как конкретная морфологическая форма может зависеть от синтаксиса или семантики. Также, как и в целом алгоритмы машинного обучения дают результаты хуже хороших структурных алгоритмов. Задача морфологического анализа не является такой сложной, как синтаксический или семантический анализ и потому использование структурных методов на данном этапе может дать лучшие результаты.

В результате выполнения морфологического анализа SDK возвращает двусвязный список токенов. Изначально токены представлены словами с морфологическими характеристиками, кроме числовых токенов, которые объединяются автоматически в один MetaToken. Диаграмма классов токенов представлена на рисунке 3.2. [36]

Для выделения остальных типов MetaToken SDK Pullenti предоставляет возможности использования различных классов-помощников.

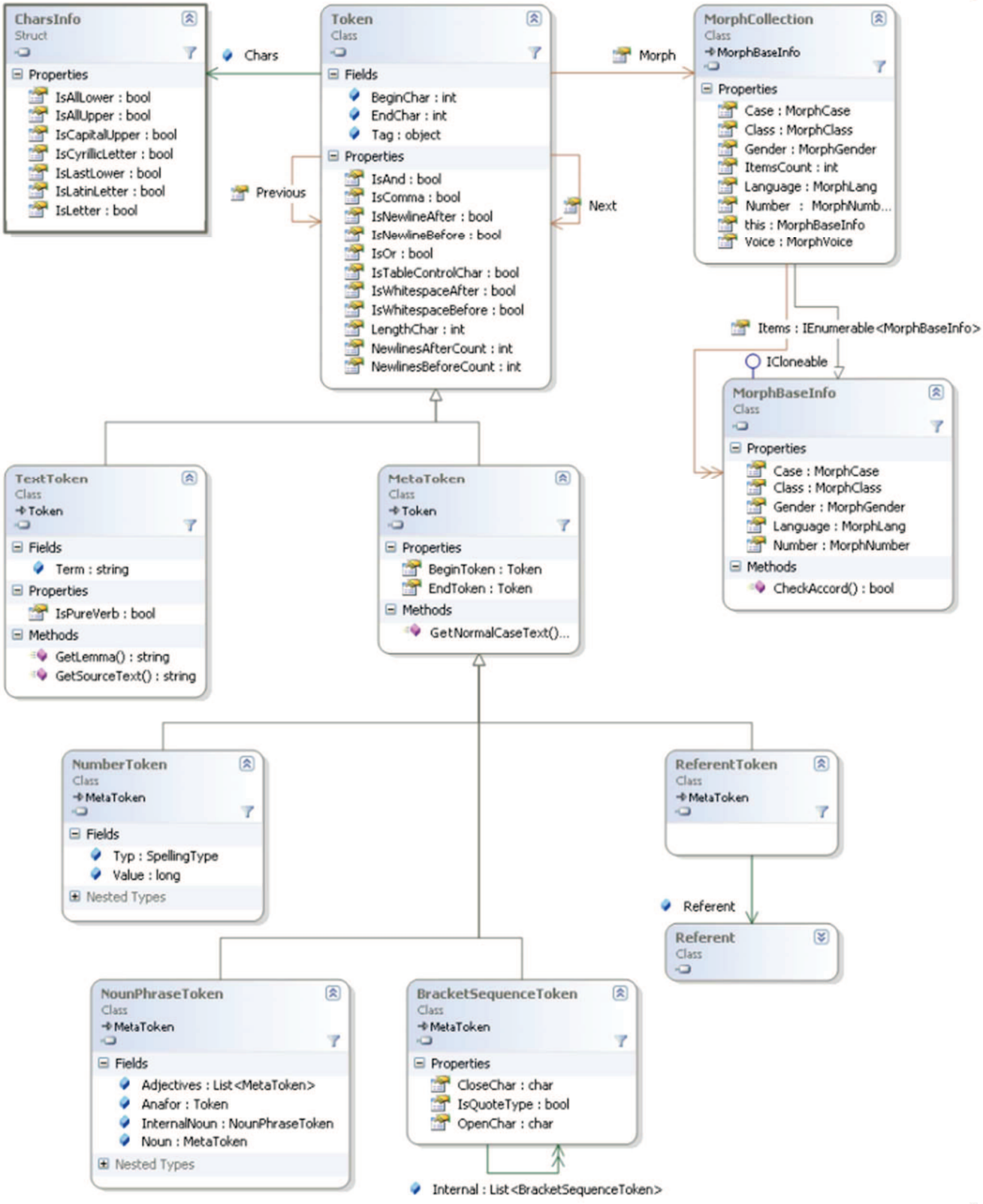

Рисунок 3.5 – Диаграмма токенов SDK Pullenti [36]

4. Синтаксический анализ.

Результатом синтаксического анализа является поток синтаксических групп:

– существительных с зависимыми прилагательными, причастиями, наречиями и частицами;

– лексемы, выделенные на этапе графематического анализа (даты, числа, ФИО и пр.)

– построение отрицаний

– и десятки других правил.

И построение на их основе синтаксического дерева.

Синтаксический анализ состоит из

– выделение клауз, т.е. простых предложений в составе сложных;

– построение синтаксических групп;

– построение синтаксического дерева на основе групп.

Из предложения выделяется клауза, если в предложении встречаются слова маркеры: «что», «когда», «где», «почему», «потому что» для сложноподчиненных предложений и союзы для сложносочиненных предложений, при условии наличия нескольких не согласованных главных членов предложения.

В таблице 3.3 представим решения для синтаксического анализа текстов на русском языке.

В таблице 3.3. представлены только самые основные решения, которые поддерживают русский язык. Также убраны многие коммерческие решения, за исключением ABBYY Compreno, так как парсер компании ABBYY стал лучшим в соревновании семантических парсеров, проходивших на Диалоге-2012 [37].

Другим популярным подходом для синтаксического анализа являются инструменты машинного обучения. Однако данные алгоритмы требуется обучать. Примером, такого синтаксического анализатора служит MaltParser [38].

Таблица 3.4 – Решения для синтаксического анализа.

Название	Метод	Лицензия	ЯП / платформа
AOT	Структурный	LGPL	C++ / Linux, Windows
MaltParser	Машинное обучение	Собственная лицензия (свободная для коммерческого и некоммерческого использования)	Python / Интерпретатор Python
ABBYY Compreno	Структурный	Коммерческая	C++ / Windows

Формат для обучения Malttab представлен в листинге 3.3. [39]

Листинг 3.3 – Формат представления предложения на Шведском для .malttab

```
Genom           pp                      3     ADV
skattereformen  nn.utr.sin.def.nom      1     PR
införs          vb.prs.sfo              0     ROOT
individuell     jj.pos.utr.sin.ind.nom  5     ATT
beskattning     nn.utr.sin.ind.nom      3     SUB
(               pad                     5     IP
särbeskattning  nn.utr.sin.ind.nom      5     APP
)               pad                     5     IP
av              pp                      5     ATT
arbetsinkomste  nn.utr.plu.ind.nom      9     PR
.               mad                     3     IP
```

В таком формате представлен результат морфологического и синтаксического анализа. Наибольшей полнотой, репрезентативностью и объемом, достаточным для обучения обладают национальные корпусы языка.

Крупнейшим национальным корпусом в России является НКРЯ – Национальный Корпус Русского Языка. [30]

НКРЯ составлялся на основе текстов, защищенных авторским правом, в результате чего его использование ограничено.

«Все результаты … доступны исключительно для некоммерческого использования в научно-исследовательских и учебных целях ... Они не предназначены ни для чтения/просмотра, ни для копирования, ни для иных видов использования: их можно использовать в режиме поиска как источники примеров (цитат), иллюстрирующих то или иное языковое явление. При цитировании примеров, полученных с помощью НКРЯ, необходимо ссылаться на НКРЯ как источник примеров, а также указывать имена всех авторов и название произведения, из которого заимствован пример».

Доступ к офлайн базе данных предоставляется только по запросу с копией паспорта и физической подписью лицензионного соглашения (изображение со сканера).

Существуют также полностью открытый корпус русского языка OpenCorpora [31]. Доступен бесплатно и в полном объеме под лицензией CC-BY-SA. Для его формирования использовались источники, являющиеся общественных достоянием, например, литература XX века. А также различные электронные ресурсы, находящиеся в общем доступе.

Следовательно, данный корпус имеет недостаточную репрезентативность. Проект находится на этапе развития, и на сегодняшний день его объем намного меньше объема НКРЯ. Представим разметку корпуса OpenCorpora, на примере одного приложения в листинге 3.4. [31]

Указание автора также обязательно для данной лицензии. Автор предложения, представленного в данном листинге, является Валентин Колесников ЧасКор: Медиа. – http://www.chaskor.ru/article/shkola_zlosloviya_uchit_prikusit_yazyk_21.

Таким образом, для синтаксического анализа будем использовать подход, предлагаемый АОТ (Автоматическая Обработка Текстов) [39] – подход, основанный на правилах.

На вход синтаксического анализатора подается поток токенов, прошедших морфологический и графематический анализ. Каждый поток должен представлять собой простое предложение. В случае наличия разных морфологических форм,

вследствие неснятой морфологической омонимии, то строится для каждого варианта морфологии строится своя синтаксическая структура.

Листинг 3.4 – Корпус русского языка OpenCorpora

```
<sentence id="2">
   <source>Сохранится ли градус дискуссии в новом сезоне?</source>
      <tokens>
         <token id="8" text="Сохранится">
            <tfr rev_id="834916" t="Сохранится">
               <v>
                  <l id="330994" t="сохранился">
                     <g v="VERB"/><g v="perf"/>
                     <g v="intr"/>
                     <g v="sing"/>
                     <g v="3per"/>
                     <g v="futr"/>
                     <g v="indc"/>
                  </l>
               </v>
            </tfr>
         </token>
         ...

      </tokens>
</sentence>
```

Поток токенов представляет собой классы, инкапсулирующие в себе токены предыдущего уровня

Для синтаксического анализатора описан отсортированный в заданном порядке список правил.

Пример синтаксического правила обнаружения подлежащего.

1 Существительное в именительном падеже.

2 Местоимение в именительном падеже.

3 Неопределенная форма глагола.

4 Если подлежащее уже найдено, то оно должно согласоваться.

4.1 Оба подлежащих местоимение или существительное, либо оба подлежащих сказуемое.

4.2 Между подлежащими есть союз либо запятая.

5 Сказуемое, если найдено, также должно согласоваться по числу и роду, либо для подлежащего – глагола в неопределенной форме быть глаголом в неопределенной форме.

В случае если пункты 4.1, 4.2 или 5 не выполняются, то данное поток токенов построен на неверной морфологической структуре. И можно переходить к следующему варианту результата синтаксического анализа.

5. Семантический анализ

Для каждого полученного результата синтаксического разбора выполняется алгоритм семантического анализа. Для этого выполняется еще один проход алгоритма, основанного на семантических правилах эквивалентного алгоритму синтаксического анализа.

Все выявленные в рамках одного семантического анализа отношения имеют вес, равный единице. Однако в случае, если потоков токенов в результате синтаксического анализа получилось несколько на основе морфологической неопределенности, тогда все результаты семантического анализа считаются равновероятными, соответственно, все веса ребер делятся на количество корректных результатов семантического разбора.

Пример одного из семантических правил обнаружения факта отношения «класс-подкласс».

1. Если текущий токен сказуемое и глагол, проверить на соответствие одной из лемм из списка: «являться», «есть» и прочее.
2. Подлежащее является существительным, а не дополнение не выбирается из списка «частью», «членом» и прочее.

При достижении определенного количества источников, формирующих вес ребра, отношение, которое это ребро отражает, становится приблизительно полным. Соответственно, факты, которые ей противоречат могут быть отвергнуты, как факты, полученные на основе ошибочного с лексической точки зрения морфологического или синтаксического разбора.

Структура проектов представлена на рисунке 3.6.

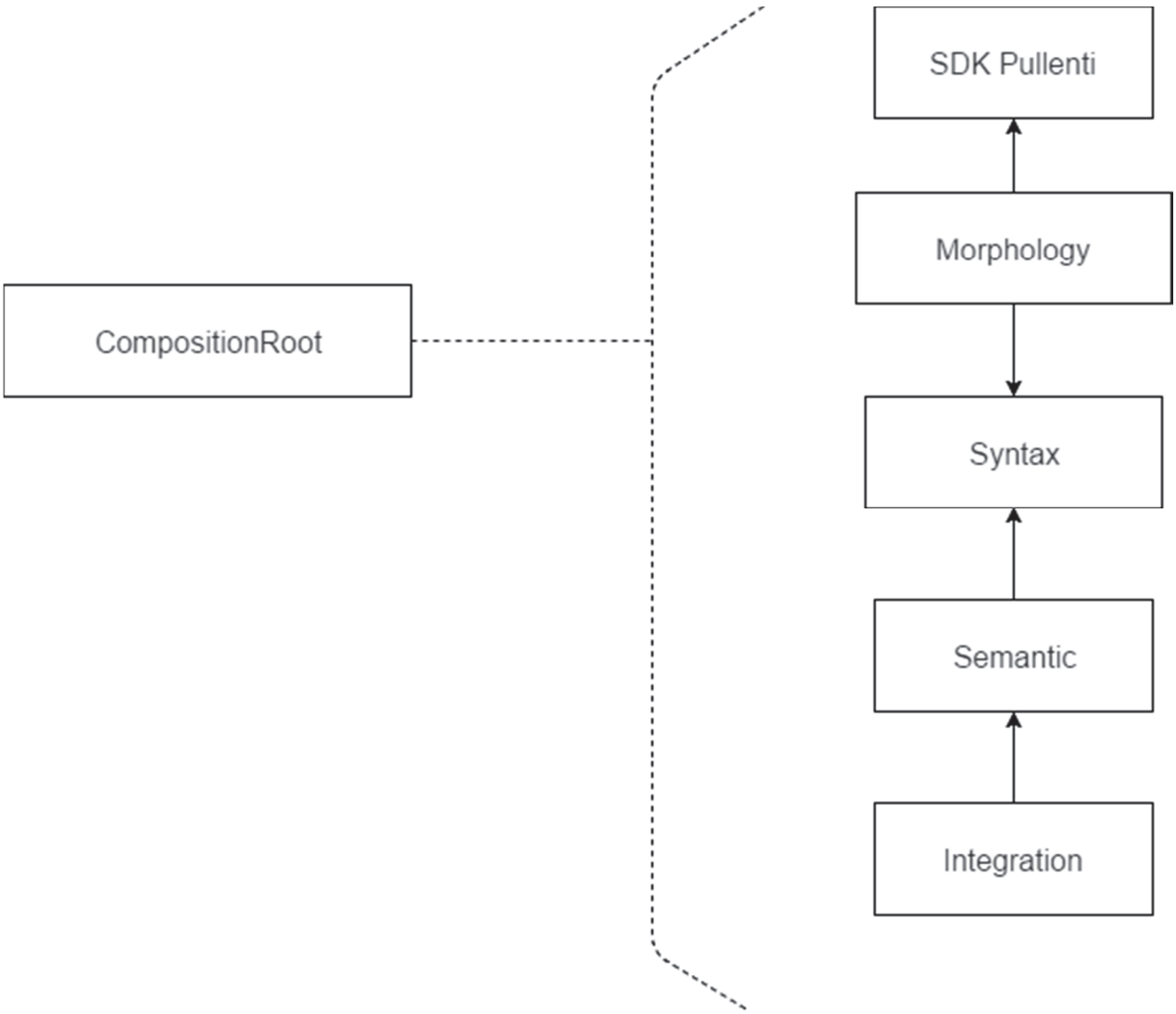

Рисунок 3.6 – Структура проектов модуля анализа текстов

На рисунке 3.6 зависимость Morphology – Syntax инвертирована для того, чтобы изолировать внешнюю зависимость SDK Pullenti в модули. Для этого используется принципе инверсии зависимости, паттерны внедрения зависимостей, а также паттерн проектирования Proxy. [40]

Для формирования весов отношений графа, воспользуемся вероятностью. Из предложения «Число – часть географической координаты.» можно выделить следующее отношение: «число part of географическая координата». На основе анализа данного отношения можно прийти к выводу, что «Число» также с некоторой вероятностью является частью понятия «Координата». Для вычисления

вероятности можно посмотреть, какие еще координаты существуют и есть ли еще такая связь.

Допустим, что связь «класс-подкласс» определена для таких координат, как «Историческая координата», «Географическая координата», «Математическая координата». И «Числа» связано отношение «часть-целоее» только с понятием «Географическая координата». Тогда можно говорить с вероятностью одна треть, что понятие «Число» - это часть понятия «Координата», соответственно, вес ребра будет равен 0,(3). Тот же алгоритм должен быть повторен для всех родительских концептов.

Предлагается выполнять данный алгоритм по таймеру на результирующей сети

Исходный код морфологического и графематического анализа представлен в Приложении Б.

Выводы по разделу три

В пункте 3.1 были сформулированы основные требования к разрабатываемой системе:

- Многомодульность для возможности расширения используемых методов формирования сети;
- Использование свободных библиотек, фреймворков и существующих словарей;
- Модуль хранения сети и модуль автоматизированного пополнения сети на основе анализа текстов на естественном языке;
- API модуля хранения семантической сети для возможности расширения внешними системами;
- Язык программирования C#
- Других особых требований в аппаратной, программной и лингвистической части нет;

В пункте 3.2 была описана структура семантической сети – гиперграфовая, для отображения межпредметных понятий и других словоформ, а также мультиграфовая для возможности отображения основных отношений, используемых повсеместно – отношений «класс-подкласс» и отношений «часть-целое», а также отношений синонимии и антономии. Граф имеет веса на отношениях для отображения отсутствия истинности знания, отображенного в семантической сети.

В пункте 3.3 описана общая архитектура многомодульного решения, преимущества такой архитектуры, и основные требования к организации модуля для обеспечения архитектурных премуществ.

В пункте 3.4 описаны основные технологии, применяемые при разработке системы: языки программирования, фреймворки, программные библиотеки, методологии и подходы программной инжерении, используемые при разработке.

В пункте 3.5 представлена структура проектов решения модуля хранения и отображения сети. Представлен разработанный API для обновления и получения концептов семантической сети, со связанными с ними отношениями. Также представлен алгоритм слияния концептов и их отношений, в случае пересечения множеств сохраняемых и уже существующих концептов.

В пункте 3.6 представлен концептуальный укрупненный алгоритм формирования семантической сети на основе лексического анализа текстов на русском языке. Выделены основные этапы алгоритма: токенизация, морфемный и морфологический анализ, графематический анализ, синтаксический анализ и выделение фактов, или семантический анализ. Описаны основные подэтапы каждого этапа, а также проблемы, которые они решают. Также были рассмотрены основные библиотеки для проведения морфологического и синтаксического анализа. Были рассмотрены возможности библиотеки SDK Pullenti. А также описаны проблемы использования корпусов русского языка и машинного обучения.

ЗАКЛЮЧЕНИЕ

В пункте 1.1 рассматривалась актуальность задачи. Были приведены примеры разработок в области анализа текстов на естественных языках, в области тезаурусов, а также технологий, выделяющих или использующих семантику для выполнения различных типов задач. В пункте 1.2 была сформулирована цель выпускной квалификационной работы. Были перечислены основные задачи, необходимые для достижения цели работы.

В пункте 2.1. были даны определения основных терминов, используемых в работе: семантическая сеть, тезаурус, корпус, синсет, а также лингвистическим типам отношений: гипоним, гипероним, холоним, мероним.

В пункте 2.2 был проведен обзор литературы и публикаций, представленные алгоритмы были обобщены в группы: вероятностные алгоритмы, алгоритмы анализа слабоструктурированных текстов, алгоритмы полного лексического анализа. Также было выделено машинное обучение, как метод, используемый в качестве вспомогательного для любой группы алгоритмов.

В пункте 2.3 были описаны существующие проблемы в области представления знаний в виде семантического графа, а также в области анализа текстов на естественных языках, в частности проблемы омономии. Были описаны случаи полной омонимии и морфологической омонимии. Также были рассмотрены модели представления межпредментых понятий и проблемы транзитивности и множественного наследования в семантических графах.

В пункте 2.4. был проведен обзор существующих семантических систем, как российских, так и зарубежных разработок. Были описаны основные особенности семантических графов WordNet, РуТез, YARN, ConceptNet. Концептуально были расмотрены методы их формирования.

В пункте 3.1 были сформулированы основные требования к разрабатываемой системе:

В пункте 3.2 была описана структура семантической сети – гиперграфовая, для отображения межпредметных понятий и других словоформ, а также мультиграфовая для возможности отображения основных отношений, используемых повсеместно – отношений «класс-подкласс» и отношений «часть-целое», а также отношений синонимии и антономии. Граф имеет веса на отношениях для отображения отсутствия истинности знания, отображенного в семантической сети.

В пункте 3.3 описана общая архитектура многомодульного решения, преимущества такой архитектуры, и основные требования к организации модуля для обеспечения архитектурных премуществ.

В пункте 3.4 описаны основные технологии, применяемые при разработке системы: языки программирования, фреймворки, программные библиотеки, методологии и подходы программной инжерении, используемые при разработке.

В пункте 3.5 представлена структура проектов решения модуля хранения и отображения сети. Представлен разработанный API для обновления и получения концептов семантической сети, со связанными с ними отношениями. Также представлен алгоритм слияния концептов и их отношений, в случае пересечения множеств сохраняемых и уже существующих концептов.

В пункте 3.6 представлен концептуальный укрупненный алгоритм формирования семантической сети на основе лексического анализа текстов на русском языке. Выделены основные этапы алгоритма: токенизация, морфемный и морфологический анализ, графематический анализ, синтаксический анализ и выделение фактов, или семантический анализ. Описаны основные подэтапы каждого этапа, а также проблемы, которые они решают. Также были рассмотрены основные библиотеки для проведения морфологического и синтаксического анализа. Были рассмотрены возможности библиотеки SDK Pullenti. А также описаны проблемы использования корпусов русского языка и машинного обучения.

В результате выполнения работы был проведен анализ и обобщение подходов, методов и алгоритмов семантического анализа текстов на естественном языке, рассмотрены существующие реализации тезаурусов. Разработана структура сети, модуль ее хранения, API для ее расширения и архитектура и минимально жизнеспособная реализация модуля на основе анализа текстов.

В ходе выполнения работы были сделаны следующие выводы:

1. Семантический анализ – трудоемкий процесс, требующий выделения и описания в виде кода большого количества правил, что не соответствует восприятию языка человеком.
2. Для дальнейшего развития алгоритма формирования семантической сети на основе анализа текстов, необходима совместная работа программных инженеров и лингвистов, для выявления и формализации правил семантического и синтаксического анализа.
3. Исследования [27] показывают связь методов дистрибутивного анализа, в частности, латентного семантического анализа, с развитием когнитивных способностей у детей, что делает данный подход хорошим способом для формирования ассоциативных отношений в семантической сети.
4. Для наиболее точного семантического анализа текстов на естественных языках, требуется формирование большого количества базовых концептов и отношений в сети. Также требуется определение более полного списка типов отношений. Например, представленного в сети ConceptNet [24] или книге Д.А. Поспелова «Ситуационное управление. Теория и практика» [29].
5. Для более точных результатов требуется построение свободных репрезентативных корпусов языка и использование методов машинного обучения на этапах синтаксического и семантического анализа.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Sowa, J. F. Semantic Networks / J.F. Sowa. – http://www.jfsowa.com/pubs/semnet.htm.

2 Лукашевич, Н.В. Тезаурусы в задачах инормационного поиска / Н.В. Лукашевич. – М.: Издательство Московского Университета, 2011. – 512 с.

3 Национальный корпус русского языка. Инструкция – http://www.ruscorpora.ru/instruction-main.pdf.

4 Гельфенбейн, И.Г. Автоматический перевод семантической сети WordNet на русский язык / И. Г. Гельфенбейн, А.В Гончарук, В.П. Лехельт, А.А. Липатов, В.В. Шило // «Компьютерная лингвистика и интеллектуальные технологии» Труды Международного семинара Диалог 2003: сб. науч. тр. – М.: Наука, 2003, С. 193–198.

5 Landauer, T.K An Indtroduction to Latent Semantic Analysis / T.K. Landauer, P.W. Folts, D. Laham. – http://lsa.colorado.edu/papers/dp1.LSAintro.pdf.

6 Landauer, T.K. A Solution to Plato' s Problem: The Latent Semantic Analysis Theory of Acquisition, Induction, and Representation of Knowledge / T.K. Landauer, S.T. Dumais. – http://www.welchco.com/02/14/01/60/96/02/2901.HTM.

7 Барановский, И.В. Методы и средства семантического анализа документов в системе делопроизводства / И.В. Барановский, В.Н, Комличенко // Актуальные вопросы современной науки: сб. науч. тр. – НИЦ «Наука и просвещение», 2015. – С. 58–68.

8 Алгоритм LSA для поиска похожих документов. – https://netpeak.net/ru/blog/algoritm-lsa-dlya-poiska-pohozhih-dokumentov/.

9 Hofmann, T. Probabilistic Latent Semantic Analysis / T. Hofmann. – http://www.iro.umontreal.ca/~nie/IFT6255/Hofmann-UAI99.pdf

10 Sahlgren, M. Automatic Bilingual Lexicon Acquisition Using Random Indexing of Parallel Corpora / M. Sahlgren, J. Karlgren // Journal of Natural Language Engineering, Special Issue on Parallel Texts. – 2005. – V.11, №3, P.327–341.

11 Морозова, Ю.И. Методика извлечения пословных переводных соответствий из параллельных текстов с применением моделей дистрибутивной семантики / Ю.И. Морозова, Е.В. Козеренко, М.М. Шарнин // Системы и средства информатики, 2014. – Т.24, №2. – С.131–142.

12 Письмак, А.Е. Метод автоматического формирования семантической сети из слабоструктурированных источников / А.Е. Письмак, А.Е. Харитонова, Е.А. Цопа, С.В. Клименков // Программные продукты и системы. – 2016. – Т.29, №3. – С. 74–78.

13 Письмак, А.Е. Оценка семантической близости предложений на естественном языке методами математической статистики / А.Е. Письмак, А.Е. Харитонова, Е.А. Цопа, С.В. Клименков // Научно-технический вестник информационных технологий, механики и оптики. – 2016. – Т. 16, №2. – С. 324–330.

14 Посевкин Р.В. Естественно-языковой пользовательский интерфейс диалоговой системы / Р.В. Посевкин, И.А Бессмертный // Программные продукты и системы, 2016. – Т.29, №3. – С. 5–8.

15 Черных И.А. Разработка методов лингвистического и семантического анализа для интеллектуальной обработки текстов, полученных в результате автоматического распознавания звучащей спонтанной русской речи / И.А. Черных, К.Е. Левин, И.П Меденников, В.И. Кабаров // Альманах научных работ молодых ученых XLV научной и учебно-методической конференции Университета ИТМО, 2016. – Т.4. – С. 211–214.

16 Hunter, S. A Novel Method of Network Text Analysis / S. Hunter // Open Journal of Modern Linguistics, 2014. – V.4. – P. 350–366.

17 R. Navigli BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network / Navigli R., Ponzetto S.P. // Journal Artificial Intelligence, 2012 – V.193. – P.217–250.

18 R Navigli BabelNet: Building a Very Large Multilingual Semantic Network / Navigli R., Ponzetto S.P. // Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, 2010 – P. 216–225.

19 J. Nivre Parsing the SYNTAGRUS Treebank of Russian / Nivre J., Boguslavsky I.M., Iomdin L.L.

20 H. Palangri Deep Learning of Grammatically-Interpretable Representations Through Question-Answering

21 Бленда, Н.А. Обзор русскоязычных тезаурусов для решения задачи расчета семантической близости между научными публикациями / Н.А. Бленда // Информационные технологии и системы: тр. Четвертой Меж-дунар. науч. конф, 2015 – С. 70–74.

22 Азарова, И. В. Принципы построения wordnet-тезауруса RussNet / И. В. Азарова, А. А. Синопальникова, М. В. Яворская. – http://www.dialog-21.ru/media/2595/sinopalnikova.pdf.

23 Браславский, П.И. – YARN: Начало / П.И. Браславский, М.Ю. Мухин, О.Н. Ляшевская, А.А. Бонч-Осмоловская, А.А. Крижановский, П. Егоров, 2013. – http://www.dialog-21.ru/digests/dialog2013/materials/pdf/BraslavskiyP_YARN.pdf.

24 ConceptNet Wiki. – https://github.com/commonsense/conceptnet5/wiki.

25 Рассел, С. Искусственный интеллект: современный подход / С. Рассел, П. Норвиг. – 2-е изд.; пер. с англ. – М.: Вильямс, 2006. – 1408 с.

26 Иванова, О.А. Формирование межпредметных понятий как метапредметных образовательных результатов / О.А. Иванова // Известия РГПУ им. А.И. Герцена, 2014. – №158. – С. 143–148.

27 Соловьев, А.Н. Использование латентно-семантического анализа в исследованиях и моделировании когнитивного развития детей. / А.Н. Соловьев. – http://www.dialog-21.ru/digests/dialog2014/materials/pdf/SolovyevAN.pdf.

28 Гаврилова, Т.А. Базы знаний интеллектуальных систем / Т.А. Гаврилова, В.Ф. Хорошевский. – СПб.: Питер, 2000. – 384 с.

29 Поспелов, Д.А. Ситуационное управление. Теория и практика / Д.А. Поспелов. – М.: Наука, 1986. – 288 с.

30 Национальный корпус русского языка. Что такое корпус? – http://www.ruscorpora.ru/corpora-structure.html

31 Бочаров, В.В. Программное обеспечение для коллективной работы над морфологической разметкой корпуса / В.В. Бочаров, Д.В. Грановский. – http://opencorpora.org/doc/articles/2011_CorpusLing.pdf

32 Эванс, Э. Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем / Э. Эванс. — М.: «Вильямс», 2011. — 448 с.

33 Osherove, R. The Art of Unit Testing: with examples in C# / R. Osherove. – Manning Publications Co., 2013. – 296 p.

34 Симан, М. Внедрение зависимостей в .NET / М. Симан. Изд-во: Питер, 2013. – 464 с.

35 «Диалог-2012»: соревнования по анализу тональности текстов и конкурс синтаксических анализаторов. – https://habrahabr.ru/company/abbyy/blog/147696/.

36 SDK Pullenti (2.58). Documentation. – http://www.pullenti.ru/DownloadFile.aspx?file=SDK_Pullenti.pdf.

37 MaltParser. User Guide. – http://www.maltparser.org/userguide.html.

38 Malt-XML, Malt-TAB and MaltConverter. – http://stp.lingfil.uu.se/~nivre/research/MaltXML.html.

39 АОТ. Синтаксический анализ. – http://aot.ru/docs/synan.html.

40 Тепляков, С. Паттерны проектирования на платформе .NET / С. Тепляков. Изд-во: Питер, 2016, – 320 с.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

Исходный код модуля хранения и отображения сети

Листинг А.1 – Исходный код класса Coefficient

```
using System;

namespace Domain
{
    public struct Coefficient : IEquatable<Coefficient>
    {
        public const double MinValue = 0;
        public const double MaxValue = 1;
        public const double Epsilon = 0.000001;

        public Coefficient(double value)
        {
            if (value > MaxValue) throw new ArgumentException(
                 "Коэффициент должнен находиться в диапазоне [0; 1]",
                 nameof(value));

            Value = value;
        }

        public double Value { get; }

        public Coefficient Mul(Coefficient other)
        {
            return new Coefficient(this.Value * other.Value);
        }

        public static Coefficient operator *(Coefficient left, Coefficient right)
        {
            return left.Mul(right);
        }

        public bool Equals(Coefficient other)
        {
            return Value - other.Value <= Epsilon;
        }

        public override bool Equals(object obj)
        {
            if (ReferenceEquals(null, obj)) return false;
            return obj is Coefficient && Equals((Coefficient) obj);
        }

        public override int GetHashCode()
        {
            return Value.GetHashCode();
        }
    }
}
```

Листинг А.2 – Исходный код класса Concept

```
using System;
using System.Collections.Generic;
using System.Collections.Immutable;
using System.Linq;

namespace Domain
{
    public class Concept : IEquatable<Concept>
    {
        public Concept(
            ConceptValue conceptValue,
            IReadOnlyCollection<Relation> outgoingRelations,
            IReadOnlyCollection<Relation> ingoingRelations)
        {
            ConceptValue = conceptValue ?? throw new
ArgumentNullException(nameof(conceptValue));
            OutgoingRelations = outgoingRelations ?? throw new
ArgumentNullException(nameof(outgoingRelations));
            IngoingRelations = ingoingRelations ?? throw new
ArgumentNullException(nameof(ingoingRelations));
        }

        protected virtual ConceptValue ConceptValue { get; }

        public virtual string Lemma => ConceptValue.Lemma;

        public virtual IReadOnlyCollection<Relation> OutgoingRelations { get; }

        public virtual IReadOnlyCollection<Relation> IngoingRelations { get; }

        public virtual Concept Merge(Concept other)
        {
            return new Concept(
                this.ConceptValue,
                MergeRelations(this.OutgoingRelations, other.OutgoingRelations),
                MergeRelations(this.IngoingRelations, other.IngoingRelations));
        }

        private IReadOnlyCollection<Relation> MergeRelations(IReadOnlyCollection<Relation>
old, IReadOnlyCollection<Relation> @new)
        {
            var oldHashSet = old.ToImmutableHashSet();
            var newHashSet = @new.ToImmutableHashSet();

            var notComonRelations = oldHashSet.SymmetricExcept(newHashSet);
            var commonRelations = oldHashSet.Intersect(newHashSet);

            var mergedRelations = commonRelations.Select(x =>
            {
                oldHashSet.TryGetValue(x, out var oldRelation);
                newHashSet.TryGetValue(x, out var newRelation);

                return oldRelation.Merge(newRelation);
            });

            return notComonRelations.Union(mergedRelations).ToList().AsReadOnly();
        }

        public virtual bool Equals(Concept other)
```

Продолжение листинга А.2

```
        {
            if (ReferenceEquals(null, other)) return false;
            if (ReferenceEquals(this, other)) return true;
            return Equals(ConceptValue, other.ConceptValue);
        }

        public override bool Equals(object obj)
        {
            if (ReferenceEquals(null, obj)) return false;
            if (ReferenceEquals(this, obj)) return true;
            if (obj.GetType() != this.GetType()) return false;
            return Equals((Concept) obj);
        }

        public override int GetHashCode()
        {
            return (ConceptValue != null ? ConceptValue.GetHashCode() : 0);
        }
    }
}
```

Листинг А.3 – Исходный код класса

```
using System;
namespace Domain
{
    public class ConceptValue : IEquatable<ConceptValue>
    {
        public ConceptValue(string lemma)
        {
            Lemma = lemma ?? throw new ArgumentNullException(nameof(lemma));
        }

        public string Lemma { get; }

        public bool Equals(ConceptValue other)
        {
            if (ReferenceEquals(null, other)) return false;
            if (ReferenceEquals(this, other)) return true;
            return string.Equals(Lemma, other.Lemma);
        }

        public override bool Equals(object obj)
        {
            if (ReferenceEquals(null, obj)) return false;
            if (ReferenceEquals(this, obj)) return true;
            if (obj.GetType() != this.GetType()) return false;
            return Equals((ConceptValue) obj);
        }

        public override int GetHashCode()
        {
            return (Lemma != null ? Lemma.GetHashCode() : 0);
        }
    }}
```

Листинг А.4 – Исходный код класса Relation

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace Domain
{
    public class Relation : IEquatable<Relation>
    {
        public Relation(ConceptValue sourceConcept, ConceptValue destinationConcept,
RelationType type, Coefficient weight, IReadOnlyCollection<SourceValue> sources)
        {
            SourceConcept = sourceConcept ?? throw new
ArgumentNullException(nameof(sourceConcept));
            DestinationConcept = destinationConcept ?? throw new
ArgumentNullException(nameof(destinationConcept));
            Weight = weight;
            _sources = sources;
            Type = type;
        }

        private readonly IReadOnlyCollection<SourceValue> _sources;

        public ConceptValue SourceConcept { get; }

        public ConceptValue DestinationConcept { get; }

        public Coefficient Weight { get; }

        public RelationType Type { get; }

        public Relation Merge(Relation other)
        {
            if (other == null) throw new ArgumentNullException(nameof(other));
            if (this != other) throw new ArgumentException("Возможно произвести слияние
только эквивалентных отношений", nameof(other));

            var newSources = this._sources.Union(other._sources).ToList().AsReadOnly();
            var newWeight = new Coefficient(newSources.Sum(s =>
s.WeightedOnConfidenceWeight.Value) / newSources.Sum(s => s.Confidence.Value));

            return new Relation(this.SourceConcept, this.DestinationConcept, this.Type,
newWeight, newSources);
        }

        public bool Equals(Relation other)
        {
            if (ReferenceEquals(null, other)) return false;
            if (ReferenceEquals(this, other)) return true;
            return Equals(SourceConcept, other.SourceConcept) &&
Equals(DestinationConcept, other.DestinationConcept) && Equals(Type, other.Type);
        }

        public override bool Equals(object obj)
        {
            if (ReferenceEquals(null, obj)) return false;
            if (ReferenceEquals(this, obj)) return true;
```

Продолжение листинга А.4

```
          if (obj.GetType() != this.GetType()) return false;
            return Equals((Relation) obj);
        }

        public override int GetHashCode()
        {
            unchecked {
                var hashCode = (SourceConcept != null ? SourceConcept.GetHashCode() : 0);
                hashCode = (hashCode * 397) ^ (DestinationConcept != null ?
DestinationConcept.GetHashCode() : 0);
                hashCode = (hashCode * 397) ^ (Type != null ? Type.GetHashCode() : 0);
                return hashCode;
            }
        }
    }
}
```

Листинг А.5 – Исходный код класса RelationType

```
using System;

namespace Domain
{
    public class RelationType : IEquatable<RelationType>
    {
        public RelationType(string name)
        {
            Name = name;
        }

        public string Name { get; }

        public bool Equals(RelationType other)
        {
            if (ReferenceEquals(null, other)) return false;
            if (ReferenceEquals(this, other)) return true;
            return string.Equals(Name, other.Name);
        }

        public override bool Equals(object obj)
        {
            if (ReferenceEquals(null, obj)) return false;
            if (ReferenceEquals(this, obj)) return true;
            if (obj.GetType() != this.GetType()) return false;
            return Equals((RelationType) obj);
        }

        public override int GetHashCode()
        {
            return (Name != null ? Name.GetHashCode() : 0);
        }
    }
}
```

Листинг А.6 – Исходный код класса ExpertAssociation

```
using System;

namespace Domain
{
    public class SourceValue : IEquatable<SourceValue>
    {
        private readonly Guid _token;

        public SourceValue(Guid token, Coefficient confidence, Coefficient weight)
        {
            if (token == null || token == Guid.Empty)
               throw new ArgumentException("Токен должен быть не пустым", nameof(token));
            _token = token;
            Confidence = confidence;
            Weight = weight;
        }

        public Coefficient Confidence { get; }

        public Coefficient Weight { get; }

        public Coefficient WeightedOnConfidenceWeight => Confidence * Weight;

        public bool Equals(SourceValue other)
        {
            if (ReferenceEquals(null, other)) return false;
            if (ReferenceEquals(this, other)) return true;
            return _token.Equals(other._token);
        }

        public override bool Equals(object obj)
        {
            if (ReferenceEquals(null, obj)) return false;
            if (ReferenceEquals(this, obj)) return true;
            if (obj.GetType() != this.GetType()) return false;
            return Equals((SourceValue) obj);
        }

        public override int GetHashCode()
        {
            return _token.GetHashCode();
        }
    }
}
```

Листинг А.7 – Исходный код класса ConceptDto

```
namespace Domain.DTOs
{
    public class ConceptDto
    {
        public string Concept { get; set; }
        public RelationDto[] IngoingRelations { get; set; }
        public RelationDto[] OutgoingRelations { get; set; }
    } }
```

Листинг А.7 – Исходный код класса RelationDto

```
namespace Domain.DTOs
{
    public class RelationDto
    {
        public string SourceLemma { get; set; }
        public string DestinationLemma { get; set; }
        public string Type { get; set; }
        public double Weight { get; set; }
    }
}
```

Листинг А.8 – Исходный код класса GetConceptsQuerySpecification

```
using System;
using System.Collections.Generic;

namespace Domain.Specifications
{
    public class GetConceptsQuerySpecification
    {
        public GetConceptsQuerySpecification(IReadOnlyCollection<string> concepts)
        {
            Concepts = concepts ?? throw new ArgumentNullException(nameof(concepts));
        }

        public IReadOnlyCollection<string> Concepts { get; }
    }
}
```

Листинг А.9 – Исходный код класса MergeConceptsCommandSpecification

```
using System;
using System.Collections.Generic;
using Domain.DTOs;

namespace Domain.Specifications
{
    public class MergeConceptsCommandSpecification
    {
        public MergeConceptsCommandSpecification(IReadOnlyCollection<ConceptDto> concepts,
Guid externalSystemToken)
        {
            if (externalSystemToken == Guid.Empty)
                throw new ArgumentException($"{externalSystemToken} не может быть пустым",
nameof(externalSystemToken));

            Concepts = concepts ?? throw new ArgumentNullException(nameof(concepts));
            ExternalSystemToken = externalSystemToken;
        }

        public IReadOnlyCollection<ConceptDto> Concepts { get; }

        public Guid ExternalSystemToken { get; }
    }
}
```

Листинг А.10 – Исходный код класса CommandHandler

```
namespace Domain.Infrastructure
{
    public abstract class CommandHandler<TSpecification>
    {
        public abstract void Handle(TSpecification specification);
    }
}
```

Листинг А.11 – Исходный код класса INetRepository

```
using System.Collections.Generic;

namespace Domain.Infrastructure
{
    public interface INetRepository
    {
        IReadOnlyCollection<Concept> GetByConceptValues(IReadOnlyCollection<ConceptValue>
conceptValues);

        void Add(Concept newConcept);

        void Update(Concept concept);
    }
}
```

Листинг А.12 – Исходный код класса IUnitOfWork

```
using System;

namespace Domain.Infrastructure
{
    public interface IUnitOfWork : IDisposable
    {
        void Commit();
    }
}
```

Листинг А.13 – Исходный код класса IUnitOfWorkFactory

```
namespace Domain.Infrastructure
{
    public interface IUnitOfWorkFactory
    {
        IUnitOfWork Create();
    }
}
```

Листинг А.14 – Исходный код класса QueryHandler

```
namespace Domain.Infrastructure
{
    public abstract class QueryHandler<TSpecification, TResult>
    {
        public abstract TResult Handle(TSpecification specification);
    }
}
```

Листинг А.15 – Исходный код класса Neo4JConnectionStringBuilder

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace Domain.Neo4j
{
    // TODO: issue #1
    public sealed class Neo4JConnectionStringBuilder
    {
        public Neo4JConnectionStringBuilder()
        {
            _params = new Dictionary<string, string>(StringComparer.OrdinalIgnoreCase)
            {
                { nameof(Server), null },
                { nameof(UserName), null },
                { nameof(Password), null }
            };
        }

        public static Neo4JConnectionStringBuilder CreateFromConnectionString(string
connectionString)
        {
            if (string.IsNullOrWhiteSpace(connectionString))
                throw new ArgumentException("Value cannot be null or whitespace.",
nameof(connectionString));

            var connectionStringBuilder = new Neo4JConnectionStringBuilder()
            {
                ConnectionString = connectionString
            };
            return connectionStringBuilder;
        }

        private readonly Dictionary<string, string> _params;

        private string _connectionString;
        public string ConnectionString
        {
            get
            {
                Validate();
                return _connectionString;
            }
            set
            {
                _connectionString = value;
```

Продолжение листинга А.15

```
            Clear();
            ParseString();
            Validate();
        }
    }

    public Uri ServerUri
    {
        get => new Uri(Server);
        set => Server = value.AbsoluteUri;
    }

    // TODO: issue #2
    public string Server
    {
        get => _params[nameof(Server)];
        set
        {
            _params[nameof(Server)] = value;
            Build();
        }
    }

    public string UserName
    {
        get => _params[nameof(UserName)];
        set
        {
            _params[nameof(UserName)] = value;
            Build();
        }
    }

    public string Password
    {
        get => _params[nameof(Password)];
        set
        {
            _params[nameof(Password)] = value;
            Build();
        }
    }

    public Neo4JConnectionStringBuilder SetupRequired(string server)
    {
        // if change this then change Validate method

        Server = server;

        return this;
    }

    public Neo4JConnectionStringBuilder SetUserName(string userName)
    {
        UserName = userName;

        return this;
```

Продолжение листинга А.15

```
}

        public Neo4JConnectionStringBuilder SetPassword(string password)
        {
            Password = password;

            return this;
        }

        private void Clear()
        {
            foreach (var key in _params.Keys.ToList()) {
                _params[key] = null;
            }
        }

        private void ParseString()
        {
            foreach (var param in _connectionString.Split(';').Where(x =>
!String.IsNullOrWhiteSpace(x))) {
                ParseParameter(param);
            }
        }

        private void ParseParameter(string param)
        {
            var keyValueParam = param.Split('=');
            if (keyValueParam.Length != 2) {
                throw new InvalidOperationException($"Parameter '{param}' is not valid");
            }
            var key = keyValueParam[0].Trim();
            var value = keyValueParam[1].Trim();

            if (!_params.Keys.Contains(key, StringComparer.OrdinalIgnoreCase))
            {
                throw new InvalidOperationException($"Unknow key '{key}' of connection
string '{_connectionString}'");
            }
            else
            {
                _params[key] = value;
            }
        }

        private void Validate()
        {
            // if change this then change SetupRequired method

            if (String.IsNullOrWhiteSpace(_params[nameof(Server)])) {
                throw new InvalidOperationException("Connection string is not valid.
Define required parameters.");
            }
        }

        private void Build()
        {
```

Продолжение листинга А.15

```
            _connectionString = String.Join(";", _params.Select(x =>
$"{x.Key}={x.Value}"));
        }
    }
}
```

Листинг А.16 – Исходный код класса Neo4JUnitOfWork

```
using System;
using Domain.Infrastructure;
using Neo4jClient;
using Neo4jClient.Transactions;

namespace Domain.Neo4j
{
    public class Neo4JUnitOfWork : IUnitOfWork
    {
        private readonly IGraphClient _graphClient;
        private readonly ITransaction _transaction;

        public Neo4JUnitOfWork(IGraphClientFactory graphClientFactory)
        {
            _graphClient = graphClientFactory?.Create() ?? throw new
ArgumentNullException(nameof(graphClientFactory));
            _graphClient.Connect();

            ITransactionalGraphClient transactionalClient = (ITransactionalGraphClient)
_graphClient;
            _transaction = transactionalClient.BeginTransaction();
            Neo4JRepository.GraphClient = _graphClient;
        }

        public void Commit()
        {
            if (!_transaction.IsOpen) {
                throw new InvalidOperationException(
             "Transaction was commited or rollbacked. Please, create new UnitOfWork.");
            }

            _transaction.Commit();
            Neo4JRepository.GraphClient = null;
        }

        public void Dispose()
        {
            if (!_transaction.IsOpen) {
                _transaction.Rollback();
            }
            _transaction.Dispose();

            _graphClient.Dispose();
            Neo4JRepository.GraphClient = null;
        }
    }
}
```

Листинг А.17 – Исходный код класса Neo4JUnitOfWorkFactory

```
using System;
using Domain.Infrastructure;
using Neo4jClient;

namespace Domain.Neo4j
{
    public class Neo4JUnitOfWorkFactory : IUnitOfWorkFactory
    {
        private readonly Neo4JConnectionStringBuilder _connectionStringBuilder;

        public Neo4JUnitOfWorkFactory(Neo4JConnectionStringBuilder
connectionStringBuilder)
        {
            _connectionStringBuilder = connectionStringBuilder ??
                                       throw new
ArgumentNullException(nameof(connectionStringBuilder));

            var configuration = NeoServerConfiguration.GetConfiguration(
                _connectionStringBuilder.ServerUri,
                _connectionStringBuilder.UserName,
                _connectionStringBuilder.Password);
            _factory = new GraphClientFactory(configuration);
        }

        private readonly GraphClientFactory _factory;

        public IUnitOfWork Create()
        {
            return new Neo4JUnitOfWork(_factory);
        }
    }
}
```

Листинг А.18 – Исходный код класса CommandConverterFacade

```
using System;
using Domain.DTOs;

namespace Domain.Services.Converters
{
    public class CommandConverterFacade
    {
        protected readonly IConverter<RelationDto, Relation> DtoToRelation;
        protected readonly IConverter<ConceptDto, Concept> DtoToConcept;

        public CommandConverterFacade(Guid externalSystemToken)
        {
            if (externalSystemToken == Guid.Empty)
                throw new ArgumentException($"{externalSystemToken} не должен быть
пустым");

            var relationConverter = new RelationConverter();
            relationConverter.SetToken(externalSystemToken);
            var conceptConverter = new ConceptConverter(relationConverter,
relationConverter);
```

Продолжение листинга А.18

```
            DtoToRelation = relationConverter;
            DtoToConcept = conceptConverter;
        }

        public Relation Convert(RelationDto dto)
        {
            return DtoToRelation.Convert(dto);
        }

        public Concept Convert(ConceptDto dto)
        {
            return DtoToConcept.Convert(dto);
        }
    }
}
```

Листинг А.19 – Исходный код класса ConceptConverter

```
using System;
using System.Linq;
using Domain.DTOs;

namespace Domain.Services.Converters
{
    public class ConceptConverter : IConverter<Concept, ConceptDto>,
IConverter<ConceptDto, Concept>
    {
        private readonly IConverter<Relation, RelationDto> _relationToDto;
        private readonly IConverter<RelationDto, Relation> _dtoToRelation;

        public ConceptConverter(
            IConverter<Relation, RelationDto> relationToDto,
            IConverter<RelationDto, Relation> dtoToRelation)
        {
            _relationToDto = relationToDto ?? throw new
ArgumentNullException(nameof(relationToDto));
            _dtoToRelation = dtoToRelation ?? throw new
ArgumentNullException(nameof(dtoToRelation));
        }

        public virtual ConceptDto Convert(Concept src)
        {
            return new ConceptDto()
            {
                Concept = src.Lemma,
                IngoingRelations = src.IngoingRelations.Select(r =>
_relationToDto.Convert(r)).ToArray(),
                OutgoingRelations = src.OutgoingRelations.Select(r =>
_relationToDto.Convert(r)).ToArray()
            };
        }

        public virtual Concept Convert(ConceptDto src)
        {
            return new Concept(
```

Продолжение листинга А.19

```
                new ConceptValue(src.Concept),
                src.IngoingRelations.Select(r =>
_dtoToRelation.Convert(r)).ToList().AsReadOnly(),

                src.OutgoingRelations.Select(r =>
_dtoToRelation.Convert(r)).ToList().AsReadOnly());
        }
    }
}
```

Листинг А.20 – Исходный код класса IConverter

```
namespace Domain.Services.Converters
{
    public interface IConverter<in TLeft, out TRight>
    {
        TRight Convert(TLeft src);
    }
}
```

Листинг А.21 – Исходный код класса RelationConverter

```
using System;
using Domain.DTOs;

namespace Domain.Services.Converters
{
    public class RelationConverter : IConverter<Relation, RelationDto>,
IConverter<RelationDto, Relation>
    {
        private Guid _queryToken;

        public void SetToken(Guid queryToken)
        {
            if (queryToken == Guid.Empty)
            {
                throw new ArgumentException("token не должен быть пустым",
nameof(queryToken));
            }
            _queryToken = queryToken;
        }

        private static TokenStore _tokenStore;
        internal static TokenStore TokenStore
        {
            get => _tokenStore ?? new TokenStore();
            set => _tokenStore = value;
        }

        public virtual RelationDto Convert(Relation src)
        {
            return new RelationDto()
            {
                SourceLemma = src.SourceConcept.Lemma,
                DestinationLemma = src.DestinationConcept.Lemma,
                Weight = src.Weight.Value,
```

Продолжение листинга А.21

```
                Type = src.Type.Name
            };
        }

        public virtual Relation Convert(RelationDto src)
        {
            if (_queryToken == Guid.Empty) {
                throw new InvalidOperationException("Перед конвертацией к доменому объекту
задайте токен");
            }

            return new Relation(
                new ConceptValue(src.SourceLemma),
                new ConceptValue(src.DestinationLemma),
                new RelationType(src.Type),
                new Coefficient(src.Weight),
                new[]
                {
                    new SourceValue(
                        _queryToken,
                        TokenStore.GetConfidenceCoefficientByToken(_queryToken),
                        new Coefficient(src.Weight))
                });
        }
    }
}
```

Листинг А.22 – Исходный код класса TokenStore

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;

namespace Domain.Services
{
    internal class TokenStore
    {
        private readonly ReadOnlyDictionary<Guid, Coefficient> _confidences = new
ReadOnlyDictionary<Guid, Coefficient>(new Dictionary<Guid, Coefficient>()
        {
            { new Guid("53ea9c69-dbc6-4c25-a90b-030e448d7403"), new Coefficient(0.6) }
        });

        internal virtual Coefficient GetConfidenceCoefficientByToken(Guid token)
        {
            if (_confidences.TryGetValue(token, out var confidence)) {
                return confidence;
            }
            else {
                throw new ArgumentException($"Клиент с токеном {token} не
зарегистрирован", nameof(token));
            }
        }
    }
}
```

Листинг А.23 – Исходный код класса GetConceptsQueryHandler

```
using System;
using System.Collections.Generic;
using System.Linq;
using Domain.DTOs;
using Domain.Infrastructure;
using Domain.Services.Converters;
using Domain.Specifications;

namespace Domain.Services
{
    public class GetConceptsQueryHandler : QueryHandler<GetConceptsQuerySpecification,
IReadOnlyCollection<ConceptDto>>
    {
        private readonly IConverter<Concept, ConceptDto> _conceptConverter;
        private readonly INetRepository _netRepository;

        public GetConceptsQueryHandler(INetRepository netRepository, IConverter<Concept,
ConceptDto> conceptConverter)
        {
            _conceptConverter = conceptConverter ?? throw new
ArgumentNullException(nameof(conceptConverter));
            _netRepository = netRepository ?? throw new
ArgumentNullException(nameof(netRepository));
        }

        public override IReadOnlyCollection<ConceptDto>
Handle(GetConceptsQuerySpecification specs)
        {
            var conceptValues = specs.Concepts.Select(c => new
ConceptValue(c)).ToList().AsReadOnly();

            var concepts = _netRepository.GetByConceptValues(conceptValues);

            return concepts.Select(c =>
_conceptConverter.Convert(c)).ToList().AsReadOnly();
        }
    }
}
```

Листинг А.24 – Исходный код класса MergeConceptsCommandHandler

```
using System;
using System.Collections.Generic;
using System.Collections.Immutable;
using System.Linq;
using Domain.Infrastructure;
using Domain.Services.Converters;
using Domain.Specifications;

namespace Domain.Services
{
    public class MergeConceptsCommandHandler :
CommandHandler<MergeConceptsCommandSpecification>
```

Продолжение листинга А.24

```
 {
        private readonly INetRepository _netRepository;

        public MergeConceptsCommandHandler(
            INetRepository netRepository)
        {
            _netRepository = netRepository ?? throw new
ArgumentNullException(nameof(netRepository));
        }

        public CommandConverterFacade GetConverterFacade(Guid externalSystemToken)
        {
            return new CommandConverterFacade(externalSystemToken);
        }

        public override void Handle(MergeConceptsCommandSpecification specification)
        {
            var converterFacade = GetConverterFacade(specification.ExternalSystemToken);
            var externalConcepts = specification.Concepts.Select(c =>
converterFacade.Convert(c)).ToImmutableHashSet();
            var netConcepts = ConceptsFromNet(specification).ToImmutableHashSet();

            foreach (var newConcept in externalConcepts.Except(netConcepts)) {
                _netRepository.Add(newConcept);
            }

            foreach (var concept in externalConcepts.Intersect(netConcepts)) {
                externalConcepts.TryGetValue(concept, out var externalConcept);
                netConcepts.TryGetValue(concept, out var netConcept);

                var conceptForUpdating = netConcept.Merge(externalConcept);
                _netRepository.Update(conceptForUpdating);
            }
        }

        private IReadOnlyCollection<Concept>
ConceptsFromNet(MergeConceptsCommandSpecification specification)
        {
            var allConceptsForQuery = specification.Concepts
                .Select(x => new ConceptValue(x.Concept))
                .ToList()
                .AsReadOnly();

            return _netRepository.GetByConceptValues(allConceptsForQuery);
        }
    }
}
```

Листинг А.25 – Исходный код класса MergeConceptsForm

```
using Domain.DTOs;

namespace WebApi.Controllers.Forms
{
    public class MergeConceptsForm
    {
        public ConceptDto[] Concepts { get; set; }
        public string Token { get; set; }
    }
}
```

Листинг А.26 – Исходный код класса MergeConceptsForm

```
using System;
using System.Collections.Generic;
using System.Linq;
using Domain.DTOs;
using Domain.Infrastructure;
using Domain.Specifications;
using Microsoft.AspNetCore.Mvc;
using WebApi.Controllers.Forms;

namespace WebApi.Controllers.Controllers
{
    [Route("api/[controller]")]
    public class NetController : Controller
    {
        private readonly IUnitOfWorkFactory _unitOfWorkFactory;
        private readonly CommandHandler<MergeConceptsCommandSpecification> _mergeCommand;
        private readonly QueryHandler<GetConceptsQuerySpecification,
IReadOnlyCollection<ConceptDto>> _getQuery;

        public NetController(
            IUnitOfWorkFactory unitOfWorkFactory,
            CommandHandler<MergeConceptsCommandSpecification> mergeCommand,
            QueryHandler<GetConceptsQuerySpecification, IReadOnlyCollection<ConceptDto>>
getCommand)
        {
            _unitOfWorkFactory = unitOfWorkFactory ?? throw new
ArgumentNullException(nameof(unitOfWorkFactory));
            _mergeCommand = mergeCommand ?? throw new
ArgumentNullException(nameof(mergeCommand));
            _getQuery = getCommand ?? throw new ArgumentNullException(nameof(getCommand));
        }

        [HttpGet("[action]")]
        public ConceptDto[] GetConcepts(params string[] concepts)
        {
            using (_unitOfWorkFactory.Create()) {
                var result = _getQuery.Handle(new
GetConceptsQuerySpecification(concepts));
                return result.ToArray();
            }
        }

        [HttpPost("[action]")]
```

Продолжение листинга А.26

```
        public void MergeConcepts([FromBody] MergeConceptsForm form)
        {
            using (_unitOfWorkFactory.Create()) {
                if (Guid.TryParse(form.Token, out var token)) {
                    _mergeCommand.Handle(new
MergeConceptsCommandSpecification(form.Concepts, token));
                }
                else {
                    throw new InvalidOperationException($"Token {token} не соответствует
формату глобального уникального идентификатора (GUID)");
                }
            }
        }
    }
}
```

Листинг А.27 – Исходный код класса Startup

```
using System.Collections.Generic;
using System.IO;
using System.Reflection;
using Domain;
using Domain.DTOs;
using Domain.Infrastructure;
using Domain.Neo4j;
using Domain.Services;
using Domain.Services.Converters;
using Domain.Specifications;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.SpaServices.Webpack;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using WebApi.Controllers.Controllers;

namespace WebUI
{
    public class Startup
    {
        public Startup(IHostingEnvironment env)
        {
            var builder = new ConfigurationBuilder()
                .SetBasePath(env.ContentRootPath)
                .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true)
                .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true)
                .AddEnvironmentVariables();
            Configuration = builder.Build();
        }

        public IConfigurationRoot Configuration { get; }
```

Продолжение листинга А.27

```
        // This method gets called by the runtime. Use this method to add services to the
container.
        public void ConfigureServices(IServiceCollection services)
        {
            // Add framework services.
            services.AddMvc()
                .AddApplicationPart(typeof(NetController).GetTypeInfo().Assembly)
                .AddControllersAsServices();

            services.AddScoped<INetRepository, Neo4JRepository>();
            services.AddSingleton<IUnitOfWorkFactory, Neo4JUnitOfWorkFactory>();
            services.AddSingleton<IConverter<RelationDto, Relation>, RelationConverter>();
            services.AddSingleton<IConverter<Relation, RelationDto>, RelationConverter>();
            services.AddSingleton<IConverter<Concept, ConceptDto>, ConceptConverter>();
            services.AddSingleton<IConverter<ConceptDto, Concept>, ConceptConverter>();
            services.AddScoped<CommandHandler<MergeConceptsCommandSpecification>,
MergeConceptsCommandHandler>();
            services.AddScoped<QueryHandler<GetConceptsQuerySpecification,
IReadOnlyCollection<ConceptDto>>, GetConceptsQueryHandler>();
            services.AddSingleton(CreateUnirOfWorkFactory());
        }

        // This method gets called by the runtime. Use this method to configure the HTTP
request pipeline.
        public void Configure(IApplicationBuilder app, IHostingEnvironment env,
ILoggerFactory loggerFactory)
        {
            loggerFactory.AddConsole(Configuration.GetSection("Logging"));
            loggerFactory.AddDebug();

            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
                app.UseWebpackDevMiddleware(new WebpackDevMiddlewareOptions {
                    HotModuleReplacement = true
                });
            }
            else
            {
                app.UseExceptionHandler("/Home/Error");
            }

            app.UseStaticFiles();

            app.UseMvc(routes =>
            {
                routes.MapRoute(
                    name: "default",
                    template: "{controller=Home}/{action=Index}/{id?}");

                routes.MapSpaFallbackRoute(
                    name: "spa-fallback",
                    defaults: new { controller = "Home", action = "Index" });
            });
        }

        private IUnitOfWorkFactory CreateUnirOfWorkFactory()
```

Продолжение листинга А.27

```
{
            Neo4JConnectionStringBuilder connectionStringBuilder;


#if DEBUG
            using (StreamReader file = File.OpenText(@"secret.json"))
            using (JsonTextReader reader = new JsonTextReader(file))
            {
                JObject jo = (JObject)JToken.ReadFrom(reader);
                connectionStringBuilder = new Neo4JConnectionStringBuilder()
                    .SetupRequired(jo["server"].ToString())
                    .SetUserName(jo["username"].ToString())
                    .SetPassword(jo["password"].ToString());
            }
#elif RELEASE
            var connectionString = Configuration.GetConnectionString("owcore");
            connectionStringBuilder =
Neo4JConnectionStringBuilder.CreateFromConnectionString(connectionString);
#endif
            return new Neo4JUnitOfWorkFactory(connectionStringBuilder);
        }
    }
}
```

ПРИЛОЖЕНИЕ Б

Исходный код модуля формирования семантической сети

Листинг Б.1 – Исходный код класса AggregatedToken

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace OW.Texts.Syntax.Tokens
{
    public abstract class AggregatedToken : Token
    {
        protected AggregatedToken(IReadOnlyCollection<Token> tokens)
        {
            Tokens = tokens ?? throw new ArgumentNullException(nameof(tokens));
        }

        public IReadOnlyCollection<Token> Tokens { get; }

        protected abstract string TokenName { get; }

        public override string ToString()
        {
            var builder = new StringBuilder();
            builder.AppendLine($"{TokenName}: ");
            builder.AppendLine($"{OriginalText}");
            foreach (var item in Tokens.Select((t, i) => new { TokenIndex = i, Token = t
})) {
                builder.AppendLine($"{item.Token}");
            }

            return builder.ToString();
        }

        public override string OriginalText => string.Join(" ", Tokens.Select(x =>
x.OriginalText));
    }
}
```

Листинг Б.2 – Исходный код класса NounPhraseToken

```
using System;
using System.Collections.Generic;
using EP.Text;

namespace OW.Texts.Syntax.Tokens
{
    public class NounPhraseToken : AggregatedToken
    {
        private readonly EP.Semantix.NounPhraseToken _baseToken;

        internal NounPhraseToken(EP.Semantix.NounPhraseToken baseToken,
IReadOnlyCollection<Token> tokens) : base(tokens)
        {
```

Продолжение листинга Б.2

```
            _baseToken = baseToken ?? throw new ArgumentNullException(nameof(baseToken));
        }

        public override string OriginalText => _baseToken.GetSourceText();

        protected override string TokenName => "Существительное с зависимыми словами";

        public MorphClass MorphClass => _baseToken.Noun.Morph.Class;

        public MorphCase MorphCase => _baseToken.Noun.Morph.Case;
    }
}
```

Листинг Б.3 – Исходный код класса NumberToken

```
using System;
using System.Collections.Generic;

namespace OW.Texts.Syntax.Tokens
{
    public class NumberToken : AggregatedToken
    {
        private readonly EP.Semantix.NumberToken _baseToken;

        internal NumberToken(EP.Semantix.NumberToken baseToken, IReadOnlyCollection<Token>
tokens) : base(tokens)
        {
            _baseToken = baseToken ?? throw new ArgumentNullException(nameof(baseToken));
        }

        public override string OriginalText => _baseToken.GetSourceText();

        protected override string TokenName => "Номер, год, число";
    }
}
```

Листинг Б.4 – Исходный код класса Sentence

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace OW.Texts.Syntax.Tokens
{
    public class Sentence : AggregatedToken
    {
        public Sentence(IReadOnlyCollection<Token> tokens) : base(tokens)
        {
            // ReSharper disable once SuspiciousTypeConversion.Global
            if (tokens.OfType<Sentence>().Any()) {
                throw new ArgumentException("Предложение не может состоять из других
предложений");
            }
        }
```

Продолжение листинга Б.4

```
        protected override string TokenName => "Предложение";
    }
}
```

Листинг Б.5 – Исходный код класса TextStructure

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace OW.Texts.Syntax.Tokens
{
    public class TextStructure
    {
        public TextStructure(IReadOnlyCollection<Sentence> sentences)
        {
            Sentences = sentences ?? throw new ArgumentNullException(nameof(sentences));
        }

        public IReadOnlyCollection<Sentence> Sentences { get; }

        public override string ToString()
        {
            var stringBuilder = new StringBuilder();
            foreach (var item in Sentences.Select((s, i) => new { SentenceIndex = i,
Sentence = s })) {
                stringBuilder.AppendLine();
                stringBuilder.Append($"{item.Sentence}");
            }

            return stringBuilder.ToString();
        }
    }
}
```

Листинг Б.6 – Исходный код класса Token

```
namespace OW.Texts.Syntax.Tokens
{
    public abstract class Token
    {
        public abstract string OriginalText { get; }
    }
}
```

Листинг Б.7 – Исходный код класса Token

```
using System;

namespace OW.Texts.Syntax.Tokens
{
    public class WordToken : Token
    {
        private readonly EP.Semantix.Token _baseToken;

        internal WordToken(EP.Semantix.Token token)
        {
            _baseToken = token ?? throw new ArgumentNullException(nameof(token));
        }

        public override string ToString()
        {
            return _baseToken.ToString();
        }

        public override string OriginalText => _baseToken.GetSourceText();
    }
}
```

Листинг Б.8 – Исходный код класса Converter

```
namespace OW.Texts.Syntax.Converters
{
    internal class Converter
    {
        public static IConverter<EP.Semantix.Token, OW.Texts.Syntax.Tokens.Token>
TokenConverter { get; } =
            new TokenConverter();

        public static IConverter<EP.Semantix.NumberToken, OW.Texts.Syntax.Tokens.Token>
NumberTokenConverter { get; } =
            new NumberTokenConverter();

        public static IConverter<EP.Semantix.NounPhraseToken,
OW.Texts.Syntax.Tokens.Token> NounPhraseTokenConverter { get; } =
            new NounPhraseTokenConverter();

        public static IConverter<EP.Semantix.Token, OW.Texts.Syntax.Tokens.Token>
WordTokenConverter { get; } =
            new WordTokenConverter();
    }
}
```

Листинг Б.9 – Исходный код класса IConverter

```
namespace OW.Texts.Syntax.Converters
{
    internal interface IConverter<in TSrc, out TDest>
    {
        TDest Convert(TSrc source);
    }
}
```

Листинг Б.10 – Исходный код класса NounPhraseTokenConverter

```
using System.Linq;

namespace OW.Texts.Syntax.Converters
{
    internal class NounPhraseTokenConverter : IConverter<EP.Semantix.NounPhraseToken,
OW.Texts.Syntax.Tokens.Token>
    {
        public OW.Texts.Syntax.Tokens.Token Convert(EP.Semantix.NounPhraseToken source)
        {
            return new OW.Texts.Syntax.Tokens.NounPhraseToken(
                source,
                new[]
{Converter.TokenConverter.Convert(source.Noun)}.ToList().AsReadOnly());
        }
    }
}
```

Листинг Б.11 – Исходный код класса NumberTokenConverter

```
using System.Collections.Generic;

namespace OW.Texts.Syntax.Converters
{
    internal class NumberTokenConverter : IConverter<EP.Semantix.NumberToken,
OW.Texts.Syntax.Tokens.Token>
    {
        public OW.Texts.Syntax.Tokens.Token Convert(EP.Semantix.NumberToken source)
        {
            var tokenList = new List<OW.Texts.Syntax.Tokens.Token>();
            for (var token = source.BeginToken; token.Previous != source.EndToken; token =
token.Next) {
                tokenList.Add(Converter.TokenConverter.Convert(token));
            }

            return new OW.Texts.Syntax.Tokens.NumberToken(source, tokenList.AsReadOnly());
        }
    }
}
```

Листинг Б.12 – Исходный код класса TokenConverter

```
namespace OW.Texts.Syntax.Converters
{
    public class TokenConverter : IConverter<EP.Semantix.Token,
OW.Texts.Syntax.Tokens.Token>
    {
        internal IConverter<EP.Semantix.NumberToken, OW.Texts.Syntax.Tokens.Token>
NumberTokenConverter { get; } =
            new NumberTokenConverter();

        internal IConverter<EP.Semantix.NounPhraseToken, OW.Texts.Syntax.Tokens.Token>
NounPhraseTokenConverter { get; } =
            new NounPhraseTokenConverter();
```

Продолжение листинга Б.12

```
internal IConverter<EP.Semantix.Token, OW.Texts.Syntax.Tokens.Token> WordTokenConverter {
get; } =
            new WordTokenConverter();

        public OW.Texts.Syntax.Tokens.Token Convert(EP.Semantix.Token source)
        {
            switch (source) {
                case EP.Semantix.NounPhraseToken noun:
                    return NounPhraseTokenConverter.Convert(noun);

                case EP.Semantix.NumberToken num:
                    return NumberTokenConverter.Convert(num);

                default:
                    return WordTokenConverter.Convert(source);
            }
        }
    }
}
```

Листинг Б.13 – Исходный код класса NumberTokenConverter

```
namespace OW.Texts.Syntax.Converters
{
    internal class WordTokenConverter : IConverter<EP.Semantix.Token,
OW.Texts.Syntax.Tokens.Token>
    {
        public OW.Texts.Syntax.Tokens.Token Convert(EP.Semantix.Token source)
        {
            return new OW.Texts.Syntax.Tokens.WordToken(source);
        }
    }
}
```

Листинг Б.14 – Исходный код класса SentenceBuilder

```
using System;
using System.Collections.Generic;
using OW.Texts.Syntax.Converters;
using EP.Semantix;

namespace OW.Texts.Syntax
{
    public class SentenceBuilder
    {
        private readonly EP.Semantix.Token _firstToken;
        private readonly int _sentenceEndChar;

        public SentenceBuilder(EP.Semantix.Token firstToken, int sentenceEndChar)
        {
            _firstToken = firstToken ?? throw new
ArgumentNullException(nameof(firstToken));
            if (sentenceEndChar <= 0)
                throw new ArgumentException(
                    $"Номер символа конца предложения ({sentenceEndChar}) не может быть
меньше или равен нулю",
                    nameof(sentenceEndChar));
```

Продолжение листинга Б.14

```
            _sentenceEndChar = sentenceEndChar;
        }

        public OW.Texts.Syntax.Tokens.Sentence Build()
        {
            var sentenceTokens = new List<OW.Texts.Syntax.Tokens.Token>();

            for (var token = _firstToken; token != null && token.EndChar <=
_sentenceEndChar; token = token.Next)
            {
                if (token is EP.Semantix.NumberToken numberToken)
                {

sentenceTokens.Add(Converter.NumberTokenConverter.Convert(numberToken));
                    continue;
                }

                // ReSharper disable BitwiseOperatorOnEnumWithoutFlags
                var attributes = NounPhraseParseAttr.AdjectiveCanBeLast |
                                 NounPhraseParseAttr.Multilines |
                                 NounPhraseParseAttr.ParseAdverbs |
                                 NounPhraseParseAttr.ParsePreposition |
                                 NounPhraseParseAttr.ParseVerbs |
                                 NounPhraseParseAttr.ReferentCanBeNoun;
                // ReSharper restore BitwiseOperatorOnEnumWithoutFlags

                var nounPhrase = NounPhraseHelper.TryParse(token, attributes);
                if (nounPhrase != null) {

sentenceTokens.Add(Converter.NounPhraseTokenConverter.Convert(nounPhrase));
                    token = nounPhrase.EndToken;
                    continue;
                }

                sentenceTokens.Add(Converter.WordTokenConverter.Convert(token));
            }

            return new OW.Texts.Syntax.Tokens.Sentence(sentenceTokens.AsReadOnly());
        }
    }
}
```

Листинг Б.15 – Исходный код класса SentenceBuilder

```
using System;
using EP;
using OW.Texts.Infrastructure;
using OW.Texts.Syntax.Tokens;

namespace OW.Texts.Syntax
{
    public class SyntaxAnalyser
    {
        public TextStructure Analyse(string text)
        {
            Logger.Current.Info("Выполнение морфологического и элементов синтаксического
анализа.");
```

Продолжение листинга Б.16

```
            Logger.Current.Info($"{Environment.NewLine}Текст: " +
                                $"{Environment.NewLine}{text}");
             var processor = new Processor();
             var result = processor.Process(new SourceOfAnalysis(text));
             Logger.Current.Info($"{Environment.NewLine}" +
                                $"{string.Join("\n", result.Log)}");

             var textSctructure = new TextStructureBuilder(result).Build();
             Logger.Current.Info($"{Environment.NewLine}" +
                                $"{textSctructure}");

             return textSctructure;
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using EP;
using EP.Semantix;
using OW.Texts.Syntax.Tokens;

namespace OW.Texts.Syntax
{
    public class TextStructureBuilder
    {
        private readonly AnalysisResult _result;

        public TextStructureBuilder(AnalysisResult result)
        {
            _result = result ?? throw new ArgumentNullException(nameof(result));
        }

        public TextStructure Build()
        {
            var textStructure = new
TextStructure(Sentences(_result).ToList().AsReadOnly());

            return textStructure;
        }

        public IEnumerable<Sentence> Sentences(AnalysisResult result)
        {
            var firstTokenOfSentence = result.FirstToken;

            for (var token = result.FirstToken; token != null; token = token.Next) {
                if (token != firstTokenOfSentence &&
                    (token.GetSourceText().Equals(".") ||
                    token.Next == null)) {
                    if (token.Next == null) {
                        yield return new SentenceBuilder(firstTokenOfSentence,
token.EndChar).Build();
                        yield break;
                    }
```

Продолжение листинга Б.16

```
                    token = token.Next;
                    if (MiscHelper.CanBeStartOfSentence(token)) {
                        yield return new SentenceBuilder(firstTokenOfSentence,
token.Previous.EndChar).Build();
                        firstTokenOfSentence = token;
                    }
                }
            }
        }
    }
}
```