

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

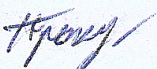
Кафедра

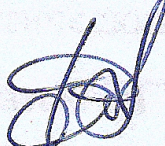
«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

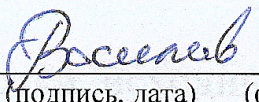
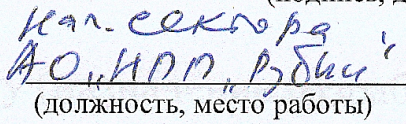
Профиль подготовки: «Вычислительные машины, комплексы, системы и сети»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ  
на тему  
**СЕМАНТИЧЕСКИЙ АНАЛИЗАТОР  
ТЕКСТОВОЙ ИНФОРМАЦИИ**

Студент  31.05.2022 (подпись, дата) Прокуров Дмитрий Алексеевич

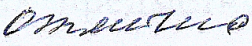
Руководитель  31.05.2022 (подпись, дата) Федюнин Р.Н.

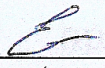
Нормоконтролёр  31.05.2022 (подпись, дата) А.С. Бычков

Рецензент  31.05.2022 (подпись, дата) (фамилия, инициалы)  (должность, место работы)

Работа допущена к защите (протокол зас. кафедры от 02.06.22 г. № 13)

Заведующий кафедрой  (подпись) М.А. Митрохин

Работа защищена с отметкой  (протокол заседания ГЭК от 14.06.22 № 4)

Секретарь ГЭК  (подпись) Е.Н. Журина

Пенза 2022

Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

*Факультет*  
вычислительной техники

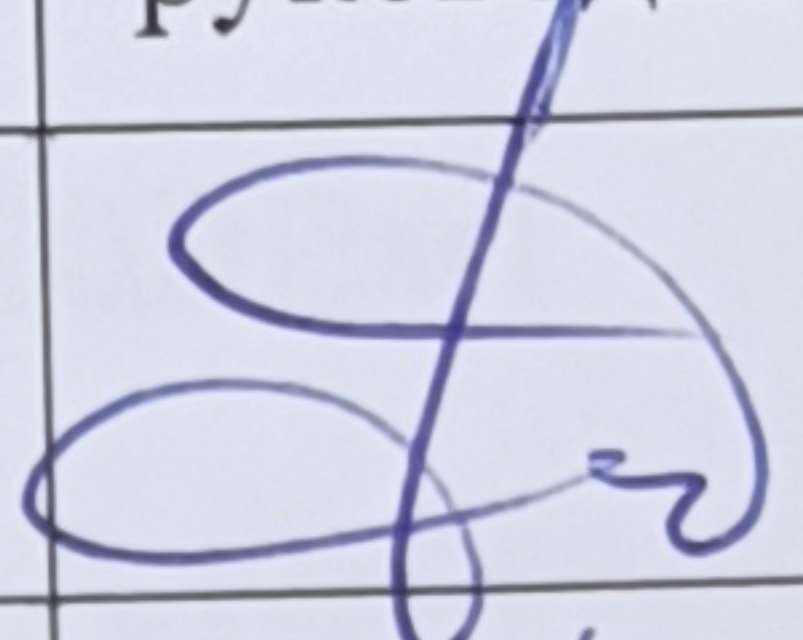
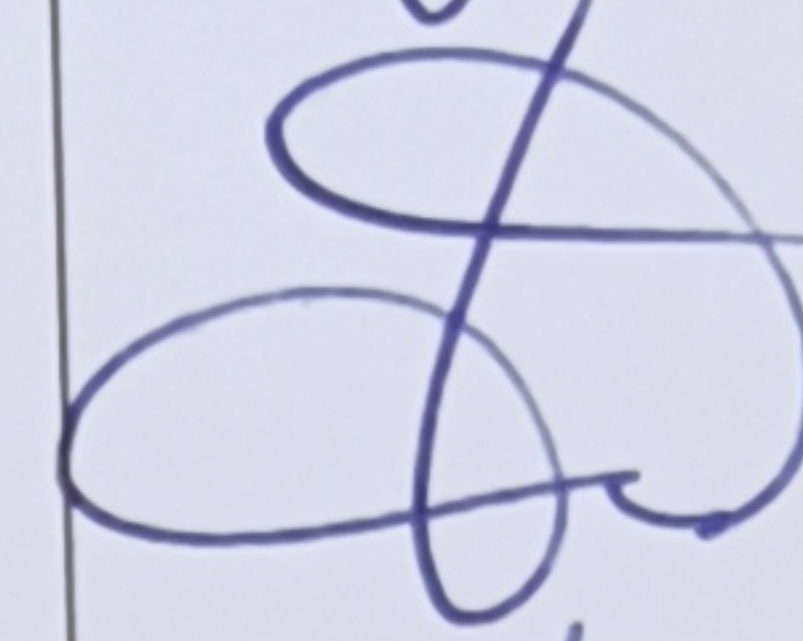
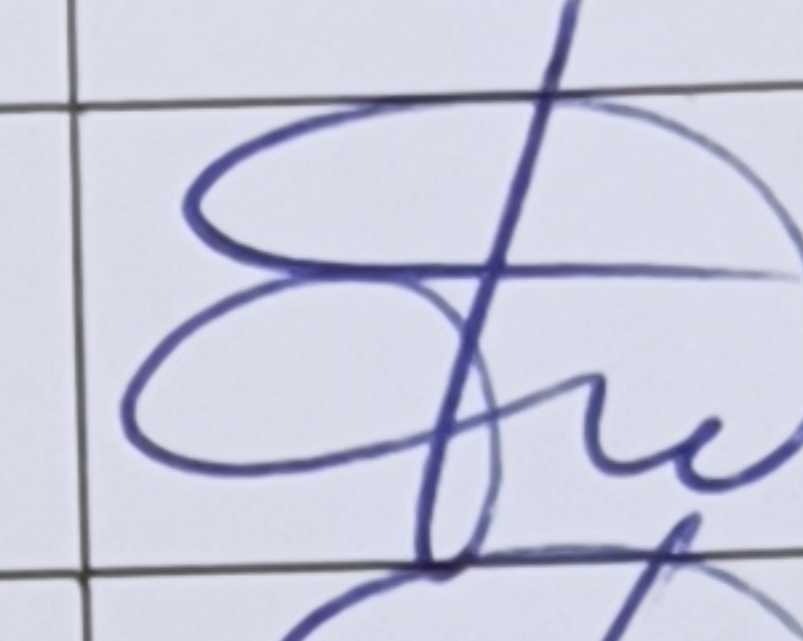
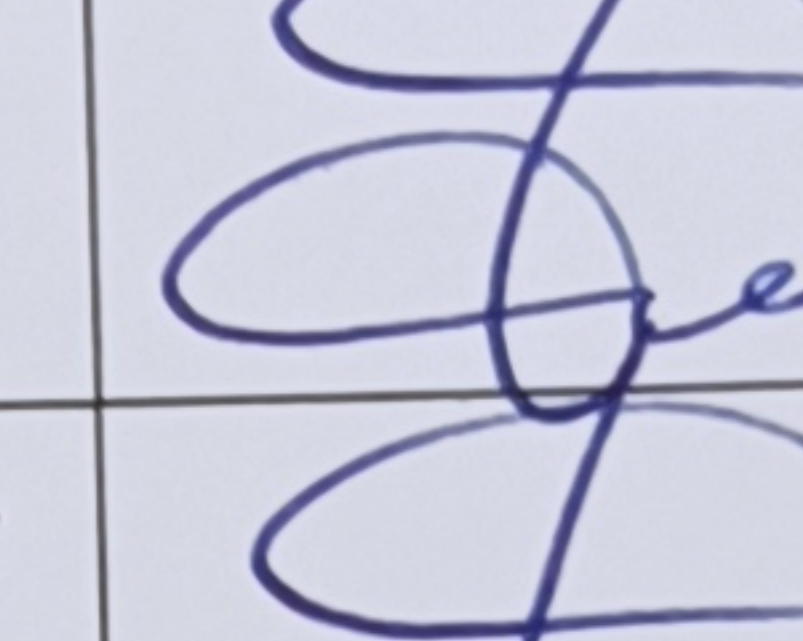
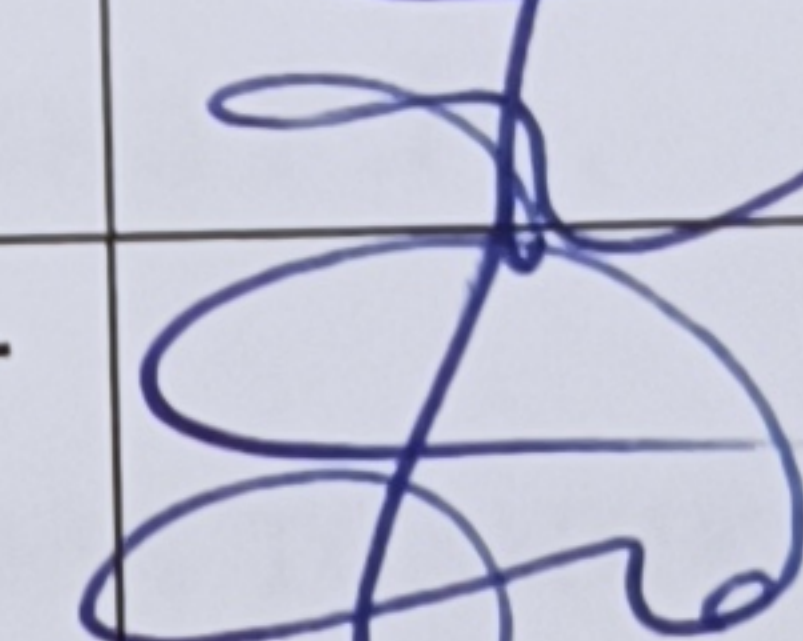
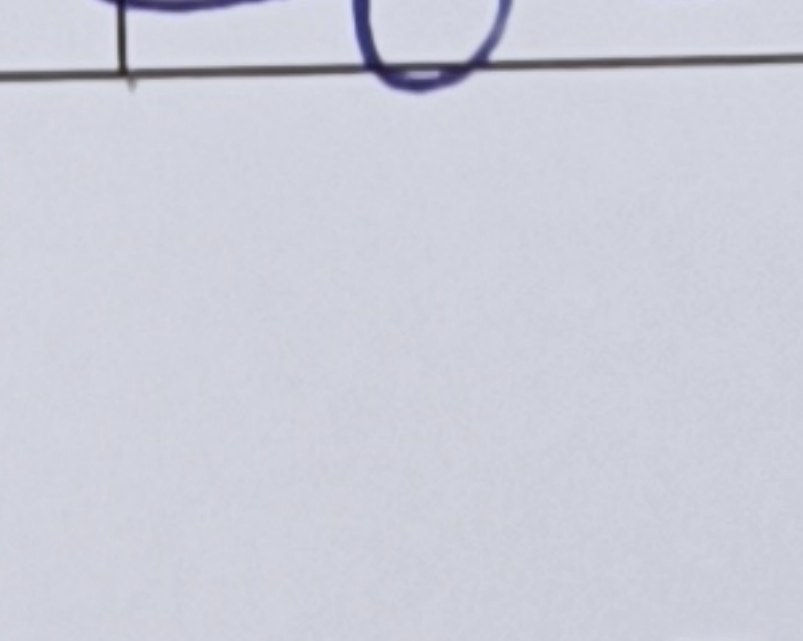
*Кафедра*  
«Вычислительная техника»

«Утверждаю»  
Заведующий кафедрой  
\_\_\_\_\_ М.А. Митрохин  
«\_\_» \_\_\_\_\_ 20\_\_ г.

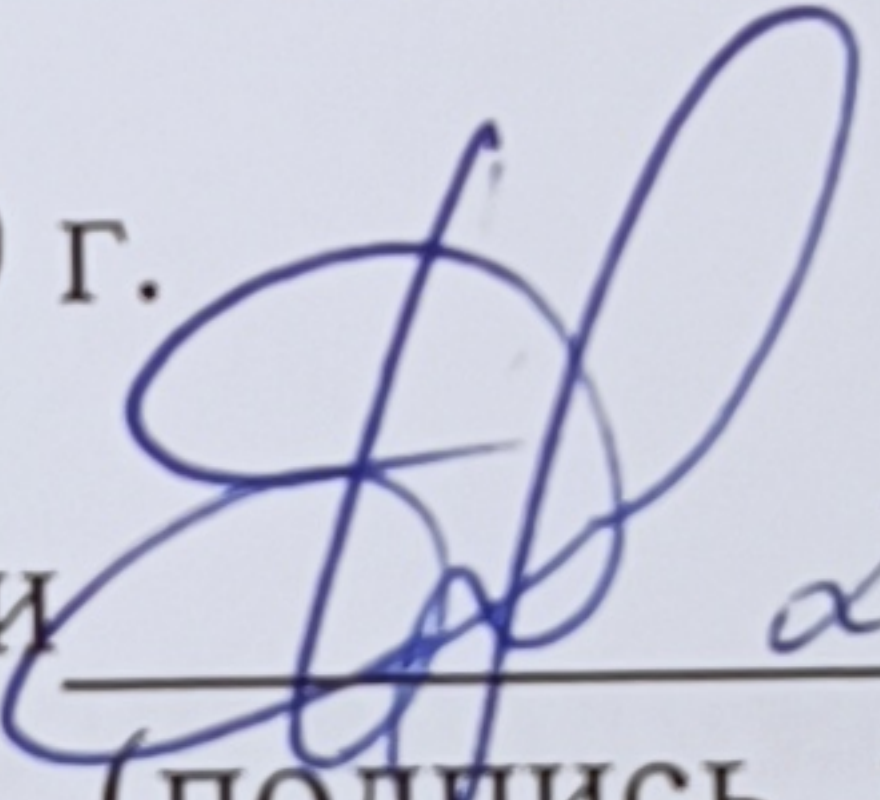
**ЗАДАНИЕ  
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ**

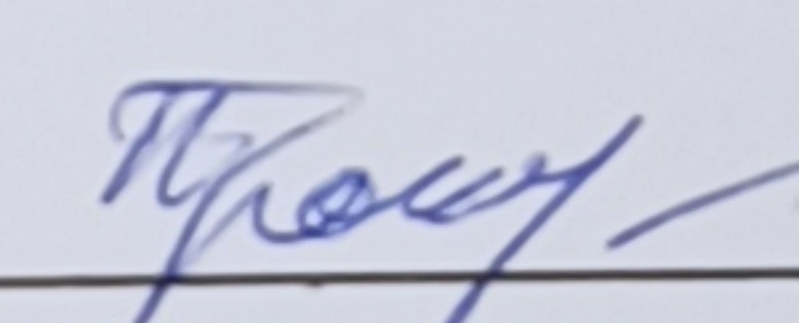
1. Студент Прокуров Дмитрий Алексеевич гр. 20ВВм1 факультета вычислительной техники.
2. Тема работы «Семантический анализатор текстовой информации».  
Тема утверждена приказом ПГУ № 656/о от "21" сентября 2020 г.
3. Руководитель магистерской диссертации к.т.н., доцент каф ВТ ПГУ Федюнин Р.Н.
4. Задание на работу (назначение разработки, исходные данные и т.п.).  
Назначение разработки- приложение, предназначенное для семантического анализа текстовой информации.
5. Перечень подлежащих разработке вопросов:
  1. Проведение анализ предметной области оптического распознавания текста и возможностей их применения для разработки десктопных приложений
  2. Определение архитектуры приложения, разработка алгоритма семантического анализа.
  3. Разработка и тестирование приложения для семантического анализа текстовой информации.

# 6. Календарный график выполнения работы

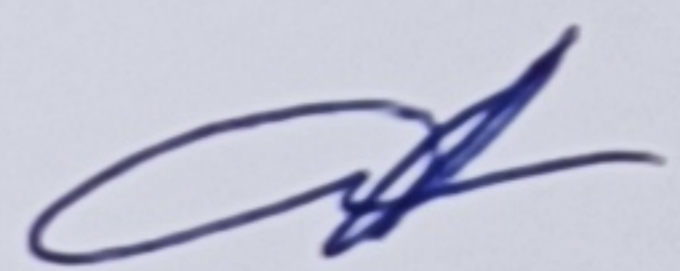
| № п/п | Наименование этапов работы                                                    | Объем работы | Срок выполнения       | Подпись руководителя                                                                  |
|-------|-------------------------------------------------------------------------------|--------------|-----------------------|---------------------------------------------------------------------------------------|
| 1.    | Анализ предметной области                                                     | 10           | 21.09.2020-21.11.2020 |    |
| 2.    | Выбор архитектуры приложения, языков программирования, способа взаимодействия | 10           | 22.11.2020-22.01.2021 |    |
| 3.    | Разработка интерфейса                                                         | 20           | 23.01.2021-23.04.2021 |    |
| 4.    | Разработка алгоритмов                                                         | 30           | 24.04.2021-29.01.2022 |   |
| 5.    | Тестирование и отладка                                                        | 10           | 30.01.2022-30.03.2022 |  |
| 6     | Оформления выпускной работы                                                   | 20           | 31.03.2022-31.05.2022 |  |

Дата выдачи задания "21" сентября 2020 г.

Руководитель магистерской диссертации  21.09.20 Федюнин Р.Н.  
(подпись, дата)

Задание к исполнению принял студент  21.09.20 Прокуров Д.А.  
(подпись, дата)

Работу к защите допустить  
Декан факультета

 Фионова Л.Р.  
(подпись, дата)  
09.06.2022

## РЕФЕРАТ

Магистерская работа содержит 80 листов, 50 рисунков, 2 таблицы, 2 приложения.

ТОКЕНИЗАТОР, ЛЕКСЕР, СЕМАНТИЧЕСКАЯ СЕТЬ, РАСПОЗНАВАТЕЛЬ, МОРФОЛОГИЧЕСКИЙ РАЗБОР, ЛЕКСЕМА

Цель работы – разработать приложение для семантического анализа текстовой информации, позволяющее пользователю проводить семантический анализ текстовой информации, содержащейся в графических изображениях, а также спроектировать графический интерфейс, представляющий результаты анализа в наиболее удобном виде.

Объектом разработки является приложение, представляющее собой программное средство семантического анализа текстовой информации.

В результате проведенной работы реализован проект, который удовлетворяет условию поставленной задачи, а именно, создано приложение для считывания и отображения семантической информации.

Созданное приложение может быть внедрено в алгоритмы социальных сетей, анализирующие оригинальность контента.

|           |      |               |                                                                                     |      |                                                                                  |                            |      |        |  |
|-----------|------|---------------|-------------------------------------------------------------------------------------|------|----------------------------------------------------------------------------------|----------------------------|------|--------|--|
|           |      |               |                                                                                     |      | ПГУ 1.090401.06.001 ПЗ                                                           |                            |      |        |  |
| Изм.      | Лист | № докум.      | Подпись                                                                             | Дата | Синтаксический анализатор<br>телеметрической информации<br>Пояснительная записка | Лит.                       | Лист | Листов |  |
| Разраб.   |      | Прокуров Д.А. |  |      |                                                                                  |                            |      |        |  |
| Провер.   |      | Федюнин Р.Н.  |  |      |                                                                                  |                            | 2    | 80     |  |
| Н. Контр. |      | Бычков А.С.   |  |      |                                                                                  | ФВТ, каф. ВТ<br>гр. 20ВВм1 |      |        |  |
| Утверд.   |      | Митрохин М.А. |  |      |                                                                                  |                            |      |        |  |

## СОДЕРЖАНИЕ

|                                                                                                          |    |
|----------------------------------------------------------------------------------------------------------|----|
| ВВЕДЕНИЕ .....                                                                                           | 5  |
| 1 Анализ предметной области для решения поставленной задачи.....                                         | 7  |
| 1.1 Постановка задачи .....                                                                              | 7  |
| 1.2 Актуальность применения средств семантического анализа.....                                          | 7  |
| 1.3 Обзор программ автоматической обработки текста.....                                                  | 8  |
| 1.3.1 Семантические вопросно-ответные поисковые системы AskNet .....                                     | 8  |
| 1.3.2 Комплекс программ для синтаксического и морфологического<br>анализа Russian Context Optimizer..... | 9  |
| 1.3.3 Программа синтаксического анализа предложений Link Grammar<br>Parser for Russian .....             | 10 |
| 1.3.4 Программа построения семантической сети TextAnalyst 2.0 .....                                      | 11 |
| 1.4 Определение архитектуры приложения .....                                                             | 12 |
| 1.5 Выбор языка программирования и среды разработки .....                                                | 15 |
| 1.6 Выбор средства оптического распознавания символов.....                                               | 16 |
| 1.7 Выводы.....                                                                                          | 16 |
| 2 Разработка приложения .....                                                                            | 20 |
| 2.1 Разработка алгоритма семантического анализа.....                                                     | 22 |
| 2.1.1 Токенизация.....                                                                                   | 22 |
| 2.1.2 Морфологический разбор.....                                                                        | 26 |
| 2.2 Разработка визуального представления семантического дерева .....                                     | 29 |
| 2.3 Разработка модели семантического дерева.....                                                         | 35 |
| 2.4 Выводы.....                                                                                          | 35 |
| 3 Описание работы приложения.....                                                                        | 38 |
| 3.1 Руководство пользователя.....                                                                        | 38 |
| 3.1.1 Назначение «Семантический анализатор» .....                                                        | 38 |
| 3.1.2 Условия выполнения «Семантический анализатор».....                                                 | 38 |
| 3.1.3 Состав «Семантический анализатор» .....                                                            | 38 |
| 3.1.4 Подготовка к работе .....                                                                          | 38 |
| 3.1.5 Описание работы с программой «Семантический анализатор». ....                                      | 39 |

|                                            |    |
|--------------------------------------------|----|
| 3.2 Тестирование приложения .....          | 52 |
| 3.2.1 Текст на русском языке .....         | 53 |
| 3.2.2 Отсутствие словаря .....             | 54 |
| 3.2.3 Изображение не содержит текста ..... | 55 |
| 3.2.4 Текст на другом языке .....          | 56 |
| 3.3 Выводы .....                           | 52 |
| ЗАКЛЮЧЕНИЕ .....                           | 58 |
| СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ .....       | 59 |
| ПРИЛОЖЕНИЕ А Листинг программы .....       | 60 |
| Файл «DrawingObject.cs» .....              | 60 |
| Файл «MainWindow.cs» .....                 | 71 |
| ПРИЛОЖЕНИЕ В Презентация .....             | 74 |

## **ВВЕДЕНИЕ**

В начале развития сети Интернет данные во всемирной паутине представляли из себя, в основном, текстовую информацию, которая легко поддаётся интерпретации машиной, например, с целью создания поисковых систем.

Однако, по мере процесса совершенствования протоколов передачи данных, а также аппаратных и программных средств их реализации, стало возможно массово использовать не только текстовую, но и графическую информацию. Особенно широкое распространение последняя получила с появлением в сети Интернет большого числа социальных сетей и мессенджеров, где люди активно взаимодействуют друг с другом с использованием различного рода мультимедиа-контента.

Таким образом, для формирования, поисковых, экспертных систем, баз данных и аналитических алгоритмов встал вопрос распознавания и анализа информации в графических файлах.

На данный момент в этой области существуют два основных направления: распознавание образов в графической информации, и распознавание символов в графической информации.

Для решения проблем во втором направлении, а именно оптическое распознавание символов (OCR, Optical character recognition) в настоящее время создано множество библиотек для различных сред программирования. Наиболее известными среди них являются Tesseract, Google Text Recognition API, Anyline. Все они обладают рядом ограничений и недостатков, основным из которых является низкая точность считывания рукописного текста. Поэтому создание новых систем обработки графической информации является важной проблемой для развития современных информационных технологий.

Другим важным вопросом остаются алгоритмы автоматического анализа, осуществляющие такие функции, как фильтрация, классификация и верификация контента в сети Интернет. В настоящее время существует достаточно много программных средств, предназначенных для

семантического анализа текста, таких как система AskNet, Russian Text Optimizer, Ontos и другие. Данные системы не предназначены для семантического анализа текстовой информации в графическом виде, поэтому разработка средства семантического анализа информации, представленной в текстовом виде, является актуальной и важной задачей, позволяющей повысить эффективность алгоритмов анализа интернет-контента.

Разработанное в данной работе приложение предназначено выполнять следующие задачи: считывание текста с графических изображений, формирование семантической модели, представление её в виде графа и дерева и сохранение семантической сети в формат XML.

## **1 Анализ предметной области для решения поставленной задачи**

### **1.1 Постановка задачи**

Необходимо разработать приложение для семантического анализа текстовой информации, представленной в графической форме. Приложение должно обладать простым и интуитивно понятным пользовательским интерфейсом и реализовывать функционал загрузки изображения, считывания текста, его семантического анализа и сохранения полученной модели в формате XML. Ниже представлен список условий для работы приложения:

- ОС Windows,
- среда разработки: Visual Studio 2019,
- язык разработки: C#.

### **1.2 Актуальность применения средств семантического анализа**

Семантический анализ представляет собой один из этапов алгоритма автоматического понимания текстов. Целью семантического анализа является выделение семантических отношений и формирование семантического представления текстов. Последнее является семантической сетью (графом), отражающим бинарные отношения между двумя смысловыми единицами текста, которые можно принять за узлы графа.

Разработки в области семантического анализа текста связаны с областью искусственного интеллекта, делающей акцент на смысловом понимании текста. Несмотря на значимость данного направления, существует множество нерешенных проблем в области семантического анализа. В настоящее время успехи в этом направлении достаточно ограничены. Разработанные семантические анализаторы обладают высокой вычислительной сложностью и неоднозначностью выдаваемых результатов.

В настоящее время семантический анализ широко распространён в таких динамично развивающихся сферах информационных технологий, как чат-боты, поисковые системы, задачи анализа тональности текста. Вкупе с существованием множества нерешённых проблем в области семантического

анализа, разработка средства семантического анализа текста представляется перспективной и актуальной.

### **1.3 Обзор программ автоматической обработки текста**

Рассмотрим альтернативные средства автоматической обработки текста, для того чтобы лучше понять, что необходимо для реализации поставленной задачи.

#### **1.3.1 Семантические вопросно-ответные поисковые системы**

##### **AskNet**

Семантические вопросно-ответные поисковые системы AskNet и инструментарий разработчика, реализующий полный лингвистический анализ текстов на русском и (или) английском языках. Модули лингвистического анализа включают в себя морфологию (словарную и бессловарную), синтаксис, семантику (включая толково-комбинаторные словари). Имеется модуль семантической рубрикации текстов. Программные продукты представлены коробочными версиями корпоративной, сайтовой и персональной поисковой системы. Вопросно-ответный поиск по Интернету реализован на базе метапоисковой системы [www.asknet.ru](http://www.asknet.ru). Разрабатывается аналитическая поисковая система AQUA, позволяющая находить семантические ответы на основе автоматического обобщения системой текстовой информации и проведения логического вывода. Программы и SDK распространяются на коммерческой основе. Уровни лингвистического анализа, поддерживаемые системой AskNet:

- графематический,
- морфологический,
- синтаксический,
- семантический.

На рисунке 1 представлен скриншот основной страницы сайта, на котором расположена система AskNet.

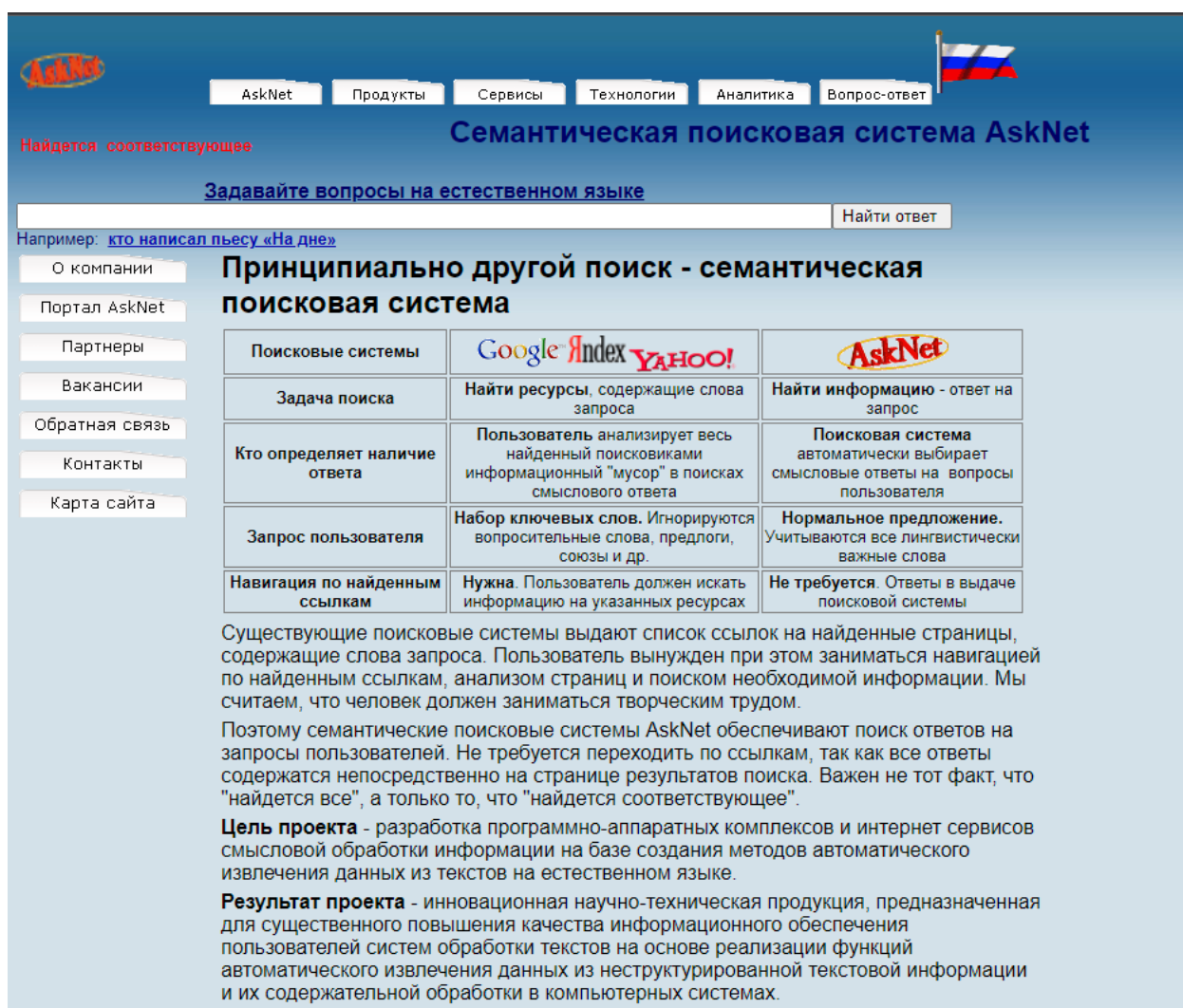


Рисунок 1 – Скриншот сайта asknet.ru

### 1.3.2 Комплекс программ для синтаксического и морфологического анализа Russian Context Optimizer

Комплекс программ для синтаксического и морфологического анализа русскоязычных текстов. Модули лингвистического анализа включают морфологические словари, словари графем, синтаксические правила, семантический классификатор, словари синонимов. Инструментарий разработчика представлен модулями морфологического и фактографического анализа. Используется морфологический словарь объемом ~ 115000 лексем. Программы разработаны для Windows, Oracle, SDK работает на платформах Windows и Unix. Программы и SDK распространяются на коммерческой основе.

Уровни лингвистического анализа, поддерживаемые системой Russian Context Optimizer:

- графематический,
- морфологический,
- синтаксический,
- семантический.

На рисунке 2 представлен скриншот основной страницы сайта компании.

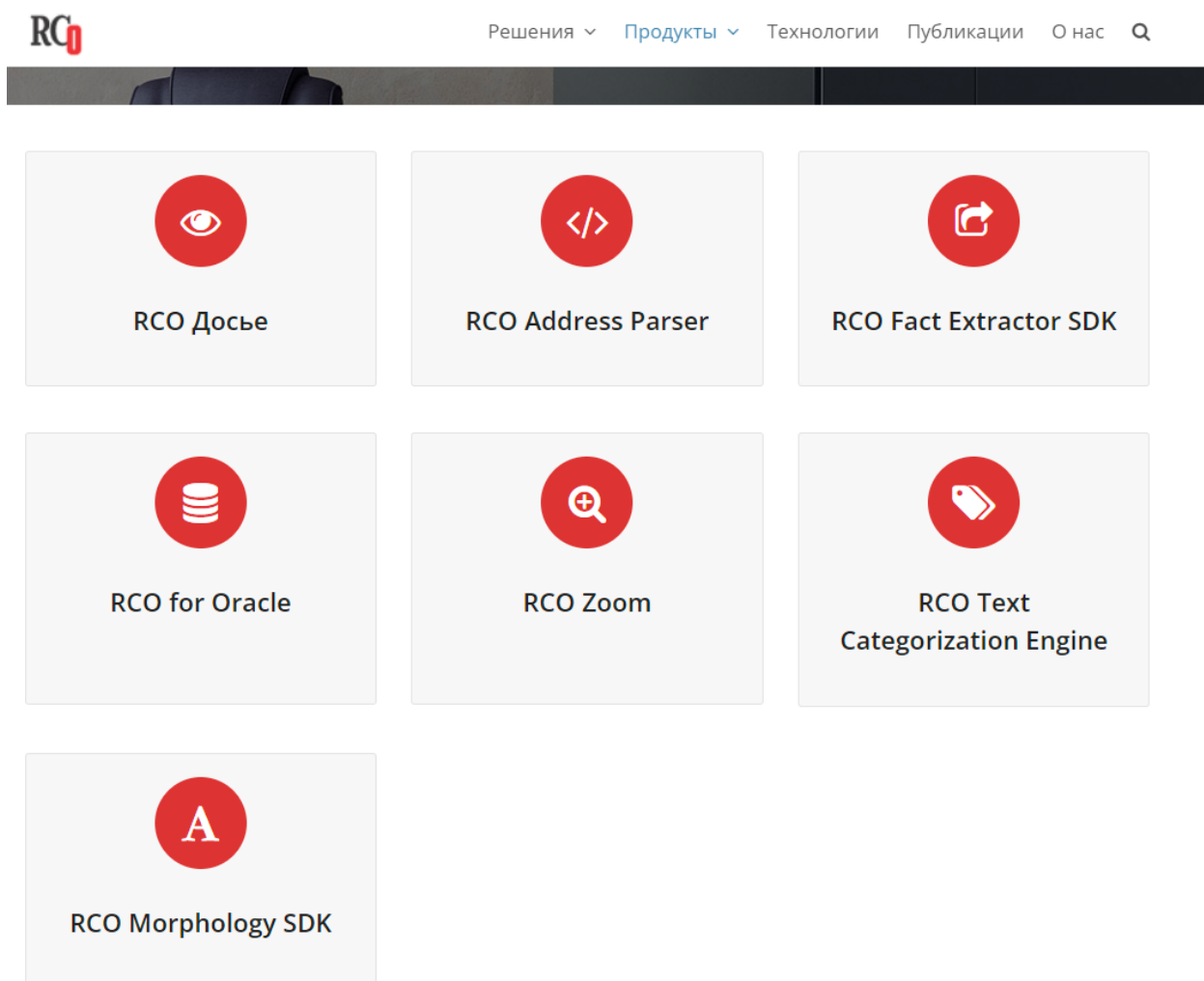


Рисунок 2 – Скриншот списка продуктов компании RCO

### 1.3.3 Программа синтаксического анализа предложений Link Grammar Parser for Russian

Online программа синтаксического анализа предложений русского языка. Создана по образцу Link Grammar Parser. Алгоритм работы



частотных списков понятий, поиска слов в контексте. TextAnalyst Lib реализует создание гипертекстовых связей выявляемых понятий. На рисунке 4 представлен скриншот главной страницы сайта TextAnalyst 2.0.



Рисунок 4 – Скриншот главной страницы сайта TextAnalyst 2.0

## 1.4 Определение архитектуры приложения

Исходя из требования к задаче, была выбрана архитектура MVC. Это одна из самых распространенных архитектур. На её основе построено множество крупных фреймворков — Java EE, Drupal, Express. Самый известный пример этой архитектуры — это сетевая модель OSI.

MVC расшифровывается как модель-представление-контроллер (от англ. model-view-controller). Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения. Архитектура MVC включает в себя следующие компоненты:

- модель,
- представление,
- контроллер.

Модель предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.

Представление отвечает за отображение данных модели пользователю, реагирует на изменения модели.

Контроллер интерпретирует действия пользователя, оповещая модель о необходимости изменений.

К достоинствам архитектуры MVC относят:

- единую концепцию архитектуры приложения,
- упрощение механизмов отладки приложения.

Несомненным преимуществом MVC является единая глобальная архитектура приложения. Даже в сложных системах, разработчик может без труда ориентироваться в программных блоках. Например, если возникла ошибка в логике обработки данных, разработчик не нуждается в том, чтобы производить отладку двух блоков программы (controller и view), поскольку ошибка с наибольшей вероятностью расположена в третьем блоке (model). Кроме того, в связи с тем, что весь механизм визуализации сконцентрирован в одном программном блоке, механизмы опционального вывода графических элементов значительно упростились.

Не смотря на вышеперечисленные достоинства, архитектурный паттерн MVC обладает рядом недостатков:

- использование большего количества ресурсов,
- усложнение механизма разделения программы на модули,
- усложнение процесса расширения функционала.

Сложность применения данного подхода обусловлена тем, что все три фундаментальных блока являются абсолютно независимыми и взаимодействуют между собой исключительно путем передачи данных. Controller должен всегда загрузить (и при необходимости создать) все возможные комбинации переменных и передать их в Model. Model, в свою очередь, должен загрузить все данные для визуализации и передать их во View. Например, в модульном подходе, модуль может напрямую обрабатывать переменные окружения и визуализировать данные без загрузки их в отдельные секции памяти.

Следующей проблемой паттерна MVC является усложнение механизма разделения программы на модули. Поскольку концепция схемы предполагает разделение каждого функционального модуля на три блока, существует необходимость функционального разделения логических модулей программы на предусмотренные шаблоном блоки. Схожие проблемы возникают и при добавлении в приложение нового функционала.

Несмотря на перечисленные выше недостатки, шаблон MVC широко применяется в следующих областях разработки ПО:

- для создания приложений, где модули относительно независимы друг от друга,
- для разработки многоуровневых приложений, основанных на идее разделения обязанностей,
- для разработки приложений, поддерживающих множественно представлений одной и той же модели.

Для разработки многоуровневых приложений, основанных на идее разделения обязанностей. Уровень Model не знает бизнес-логики приложения и сохраняет только структуру данных и предоставляет интерфейс для операций с данными, что позволяет сделать процесс обработки событий более прозрачным а, значит, существенно упростить отладку приложения.

Требования к программе предусматривают несколько не связанных друг с другом вариантов представления одной и той же модели, что хорошо подходит под концепцию MVC. Уровень Model не имеет доступа к бизнес-логике приложения и сохраняет только структуру данных и предоставляет интерфейс для операций с данными, что позволяет сделать процесс обработки событий более прозрачным а, значит, существенно упростить отладку приложения.

Исходя из вышеизложенного, паттерн MVC является наиболее оптимальным вариантом для проектирования архитектуры приложения.

## 1.5 Выбор языка программирования и среды разработки

Исходя из требований к задаче, необходимо выбрать язык программирования для реализации редактора. После чего на основе выбранного языка нужно определить среду разработки.

Приложение должно обладать графическим интерфейсом и уметь реагировать на действия со стороны пользователя. Т.к. это является одним из ключевых требований, то необходимо выбрать язык программирования, который позволяет относительно легко создавать графический интерфейс и поддерживать механизм событийности. Также необходимо, чтобы была возможность использовать классы при написании программы. Это повысит читаемость исходного кода и позволит выделить сущности необходимые для грамотной реализации программы. В свою очередь, это облегчит поддержку и развитие приложения.

С учетом выше сказанного был выбран язык программирования C#. C# является объектно-ориентированным языком программирования, в котором присутствует такое понятие как классы. Класс – это конструкция языка, которая позволяет создавать собственные типы данных за счет группировки переменных других типов, методов и событий. Классы позволяют скрыть данные и определить набор операций, выполняемых над ними. За счет этого можно относительно легко разбить программу на сущности, которые будут представлять собой независимые части. Это повысит наглядность и легкость восприятия кода программы, что в свою очередь поможет проще обнаружить и локализовать возможные ошибки в исходном коде программы. Помимо вышеуказанных преимуществ язык C# является быстрым в работе, что подтверждают исследования, проведённые авторами книги «Совершенный код», надёжным, простым в освоении. Также следует отметить схожесть синтаксиса C# с такими популярными языками программирования, как C++, Java и прочими объектно-ориентированными языками программирования. Кроме того, реализованная в языке C# сборка мусора позволяет не

беспокоиться о необходимости автоматического удаления более не используемых объектов.

Графический интерфейс приложения можно реализовать с помощью средств платформы Microsoft Visual Studio, таких как набор библиотек Windows.Forms. Данный инструмент позволяет относительно легко создавать графические интерфейсы за счет того, что имеется графический редактор формы, который позволяет просмотреть её вид на этапе разработки непосредственно до запуска программы. Данный редактор автоматически генерирует весь код, отвечающий за внешний вид, для каждого элемента формы, что позволяет ускорить разработку интерфейса и избежать нежелательных ошибок в коде, отвечающем за интерфейс. Помимо этого, редактор Windows.Forms позволяет использовать в создании графического интерфейса пользовательские компоненты интерфейса, что позволит значительно упростить разработку интерфейса.

Выбор среды разработки пал на продукт Microsoft – Visual Studio 2019. Официальная, бесплатная и самая функциональная IDE для разработки на C#. Является самой распространенной среди C# разработчиков. Visual Studio полностью совместима с TFS, имеет огромное количество встроенных библиотек и инструментов, поддерживающих компиляцию, отладку, профилирование, юнит-тестирование. Также стоит отметить, что Visual Studio поддерживает подключение сторонних плагинов, что необходимо для подключения библиотеки OCR.

### **1.6 Выбор средства оптического распознавания символов**

Для решения проблем оптического распознавания символов (OCR, Optical character recognition) в настоящее время создано огромное количество библиотек для различных сред программирования. Наиболее известными среди них являются Tesseract, Google Text Recognition API, Anyline. Однако они, так или иначе обладают рядом ограничений и недостатков, основным из которых является низкая точность считывания рукописного текста. Поэтому создание новых систем обработки графической информации является

достаточно важной проблемой для развития современных информационных технологий.

Одним из примеров является API от компании Google, а именно Google Text Recognition API. Данная библиотека позволяет обнаружить текст в изображениях и видео потоках и распознать содержащийся в нем текст. После обнаружения распознаватель определяет фактический текст в каждом блоке и разбивает его на слова и строки. К достоинствам можно отнести большое количество поддерживаемых языков, возможность распознавания в режиме реального времени, высокую скорость распознавания, а также небольшой размер библиотеки. Недостатком является большой размер файлов с обученными данными.

OCR библиотека Tesseract имеет открытый исходный код и распространяется бесплатно. Благодаря открытому коду достаточно легко обучить OCR распознавать нужные шрифты и повысить качество распознаваемой информации. После обучения библиотеки качество результатов распознавания стремительно возрастает. Также поддерживает большое количество языков. Недостатком является то, что для распознавания текста в реальном времени требуется дополнительная обработка полученного изображения.

Anyline — многоплатформенный SDK, который позволяет разработчикам легко интегрировать функции OCR в приложения. Данная OCR библиотека привлекательна тем, что имеет многочисленные возможности по настройке параметров распознавания и предоставляет модели для решения конкретных прикладных задач. Позволяет распознавать текст в реальном времени, а также помимо текста библиотека способна распознавать штрихкоды и QR-коды.

Iron OCR — это простая в установке, полная и хорошо документированная программная библиотека .NET. Выбор IronOCR позволяет достичь точности оптического распознавания текста более 99% без использования каких-либо внешних веб-сервисов. Iron OCR отлично подходит

при работе с реальными изображениями и несовершенными документами, такими как фотографии, или сканированные изображения с низким разрешением, которые могут иметь цифровой шум или дефекты.

В таблице 1 приведены сравнительные характеристики библиотек оптического распознавания текста.

Таблица 1 – Сравнительные характеристики OCR-библиотек

|                          | <b>Tesseract</b> | <b>ABBYY<br/>FineReader</b> | <b>IronOCR</b> | <b>Весовые<br/>оценки</b> |
|--------------------------|------------------|-----------------------------|----------------|---------------------------|
| Платность                | 10               | 0                           | 10             | 0,3                       |
| Уровень функциональности | 3                | 10                          | 10             | 0,2                       |
| Наличие поддержки        | 0                | 10                          | 10             | 0,15                      |
| Качество                 | 6                | 10                          | 10             | 0,25                      |
| Скорость                 | 10               | 8                           | 5              | 0,1                       |

Для каждого критерия определён коэффициент, так что бы сумма всех коэффициентов была равна единице. Ниже приведены полученные критерии с соответствующими коэффициентами:

- платность = 0,3,
- уровень функциональности = 0,2,
- наличие поддержки = 0,15,
- качество = 0,25,
- скорость = 0,1.

Для выбора наиболее подходящей для выполнения задания библиотеки был применён метод свёртки по аддитивному критерию. Полученные результаты представлены в таблице 2.

Таблица 2 – Свёртка по аддитивному критерию

|                            | <b>Tesseract</b> | <b>ABBYY<br/>FineReader</b> | <b>IronOCR</b> |
|----------------------------|------------------|-----------------------------|----------------|
| Кол-во языков              | 3                | 0                           | 3              |
| Качество перевода          | 0,6              | 2                           | 2              |
| Максимальная длина текста  | 0                | 1,5                         | 1,5            |
| Интерфейс                  | 1,5              | 2,5                         | 2,5            |
| Дополнительные инструменты | 1                | 0,8                         | 0,5            |
| Сумма                      | 6,1              | 6,8                         | 9,5            |

Таким образом, исходя из вышеизложенного обзора, наилучшей библиотекой OCR для выполнения задачи является IronOCR.

### **1.7 Выводы**

В рамках этого раздела были достигнуты следующие результаты: произведён исчерпывающий анализ предметной области, в рамках которой планируется вести разработку приложения семантического анализа текстовой информации, была выбрана наиболее подходящая архитектура, а также были выбраны языки и средства разработки всех компонентов системы.

## 2 Разработка приложения

После запуска программы отображается интерфейс приложения. В момент запуска в приложение ещё не загружено изображение, поэтому пользователь должен загрузить файл изображения, содержащего текст на английском или русском языке. После загрузки файла, необходимо нажать кнопку «Преобразовать изображение в текст», после чего программа производит считывание текстовой информации с изображения. Если в ходе считывания возникает ошибка, то программа предупреждает пользователя какого эта ошибка рода. В случае, если считывание произошло без ошибок, в средней части приложения отображается текст. Пользователь имеет следующие возможности работы с полученным текстом: создание модели в виде дерева, создание модели в виде графа и сохранение модели в виде XML-документа. В случае, если текст не является связанным и согласованным, программа выведет сообщение, предупреждающее пользователя о невозможности совершения подобных действий. Созданная модель отображается в правой части приложения в виде графа или дерева.

В соответствии с выбранной архитектурой MVC, программу можно разделить на три логические части:

- models,
- views,
- controllers.

Models содержит описание элементов семантической модели, а также их специфических свойств.

Views представляет собой библиотеку классов для визуального отображения элементов модели, а также элемент управления GraphView, на котором размещаются экземпляры вышеупомянутых классов в соответствии с их назначением в семантической модели.

Controllers – библиотека классов, позволяющая оперировать с текстом и элементами семантической модели.

Отдельным элементом программы является класс MainWindow, в котором реализована обработка действий пользователя и взаимодействие с UI.

UML-диаграмма последовательности приложения представлена на рисунке 5.



Рисунок 5 – UML-диаграмма последовательности

UML-диаграмма вариантов использования представлена на рисунке 6.

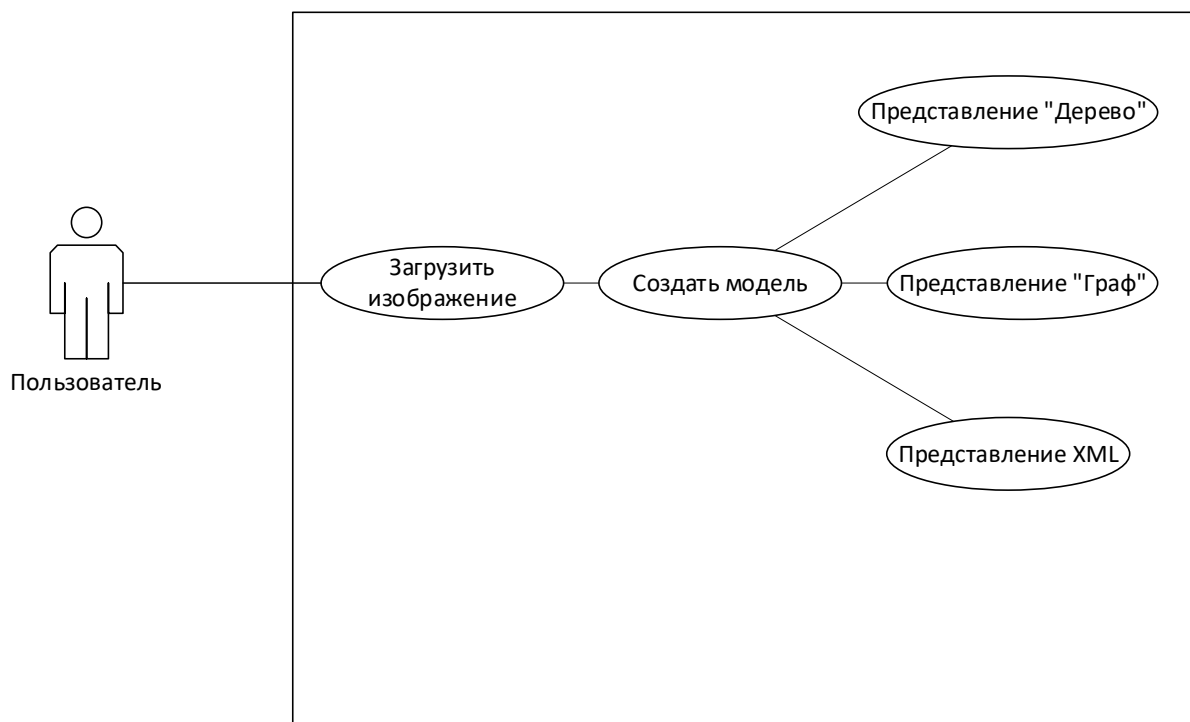


Рисунок 6 – UML-диаграмма вариантов использования

## 2.1 Разработка алгоритма семантического анализа

Рассмотрим подробнее процесс преобразования графического представления текста в семантический граф. Первым этапом является оптическое распознавание текста средствами библиотеки IronOCR. Перед началом обработки текста необходимо задать такие параметры, как язык, необходимость автоматического выравнивания текста, а также удаление шума. Результатом оптического преобразования является текст. После получения текстовой информации необходимо определить границы слов в предложении с учетом разнообразных правил для русского языка или английского языка. Этот этап называется токенизация и осуществляется в одном из классов, реализующих интерфейс *ITokenizer* (*RTokenizer*, *ETokenizer*), содержащем метод *Tokenize()*.

### 2.1.1 Токенизация

На этапе токенизации задача программы – выделить в тексте основные структурные единицы – лексемы и распознать их, сопоставив со словарными формами или другими морфологическими образцами. Токенизатор разработан для обработки реальных текстов, содержащих различные грамматические

ошибки и опечатки. Поэтому кроме пассивного распознавания слов в исходной цепочке он может также активно модифицировать исходную цепочку, объединяя и расщепляя лексемы, а также добавляя новые. В результате работы токенизатора (лексера) получается сложная структура данных – граф токенизации, который будет рассмотрен далее.

Граф токенизации является исходным материалом для работы синтаксического парсера. Лексер появляется в системе обработки текста в результате декомпозиции задачи парсинга. Он упрощает реализацию морфологического и синтаксического анализаторов, так как позволяет им работать с более крупными единицами – лексемами. Вводимое таким способом упрощение неявно ограничивает общность всей системы, так как сама по себе идея разбивки текста на независимые лексемы сочетается не со всеми языками. Более того, даже для языков с естественным выделением слов на письме в звуковом представлении появляются сложные эффекты слияния слов в более крупные единицы.

Токенизатор может работать без явно выделенных правил, используя только ту информацию, которая содержится в словаре выбранного языка. Обязательными требованиями к работе лексера являются задание типа границ слов и список символов-разделителей. Для каждого языка можно независимо задать свой набор разделителей предложений. Для европейских языков, включая русский и английский, таковыми символами обычно являются точка, знаки вопроса и восклицания. Если в качестве разделителей используется точка, то она обрабатывается особым образом, в отличие от знаков «?» и «!». Дело в том, что некоторые слова могут содержать точку, и это не должно вызывать обрыва предложения. Типичный пример – сокращения типа «т.е.» или аббревиатуры наподобие «Р.С.Ф.С.Р.». Аналогично особо обрабатываются числа с десятичной точкой «3.141».

Кроме определения границ лексем, лексер также выполняет предварительное распознавание морфологических атрибутов слов, превращая лексемы в токены. Для этого лексер использует информацию в лексиконе и

правила распознавания несловарных лексем, а также ряд вспомогательных алгоритмов, включая нечеткое распознавание. При распознавании слова определяются такие характеристики, как принадлежность к определенной части речи и набор грамматических атрибутов. Для словарных лексем также определяется словарная статья, формой которой является лексема. Результат работы лексера – направленный граф, не имеющий циклических путей. На рисунке 7 представлена простейшая цепочка токенов без ветвлений:

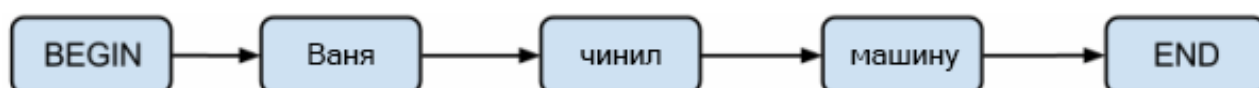


Рисунок 7 – Простейший граф токенизации

В более сложных случаях появляются альтернативные пути токенизации, предполагающие разветвления графа. В узлах графа находятся токены. Каждый токен хранит информацию о местоположении извлеченного слова в исходной среде (в случае печатного текста это индекс первого символа и количество символов в слове), само слово и результаты его распознавания.

Слева всегда находится специальный токен, обозначающий начало предложения. Каждый лист графа является специальным токеном, обозначающим конец предложения. Каждый путь в графе заканчивается специальным токеном. Для большинства случаев этот токен обозначает правую границу предложения, но для некоторых случаев, особенно при работе с беспробельными языками, может появляться токен-терминатор, который обозначает невозможность продолжить выделение слов. Таким образом, синтаксический анализатор имеет возможность учитывать близость слов к границам высказывания, что может быть полезным для оптимизации некоторых правил фильтрации токенов.

Лексер работает в очень тесном взаимодействии с парсером текста – синтаксическим анализатором. Иными словами, лексер и парсер работают как единое целое. Распознаваемые лексером слова подтверждают или опровергают выдвигаемые парсером гипотезы о синтаксической структуре текста. Парсер, в свою очередь, на основе текущего контекста выдвигает

новые гипотезы, которые прерывают или продолжают отдельные пути токенизации в графе.

Создаваемый граф токенизации может содержать произвольное количество ветвлений. Кроме того, отдельные пути в этом графе могут иметь разное количество узлов. Другими словами, лексер может разбивать исходное предложение множеством альтернативных путей. Выбор одного из путей ложится на синтаксический анализатор. Фактически, в ленивом режиме лексер и синтаксический анализатор работают как один алгоритм. Таким образом, разбивка на слова зависит не только от формальных правил выделения границ слов (для соответствующих языков), но и от синтаксиса. Более того, при обработке реальных текстов даже на языках с «пробельной» сегментацией лексер может выполнить принудительную разбивку двух слов, написанных слитно (*Пример: «я незнаю»*), или слить две части одного слова (*Пример: «что ли»*).

Так как ветвления не являются экстраординарным явлением и могут сохраняться даже в результатах синтаксического разбора, то схема работы лексера не сводится к алгоритму с возвратами.

Способность (и даже склонность) данного лексера генерировать развилки в графе токенизации отличает его от лексеров, обычно применяемых для разбора текстов на языках программирования. Старые языки, такие как Фортран или PL/I, в которых лексический анализ был осложнен отсутствием пробелов (Фортран) или отсутствием зарезервированных ключевых слов, представляют в упрощенном виде те сложности, с которыми приходится сталкиваться при разборе текста на естественном языке. Ветвления могут создаваться либо по инициативе самого лексера, например для работы с многословными лексемами, либо генерироваться вручную заданными правилами.

Возможность создавать альтернативные пути в графе токенизации принципиально важна для алгоритмов коррекции ошибок. Эти алгоритмы работают на уровне отдельных слов и переключают выбор одного из

альтернативных наборов модификаций исходных токенов на синтаксический анализатор.

### 2.1.2 Морфологический разбор

Следующим этапом семантического анализа является морфологический разбор полученного графа токенизации. Морфологический анализатор — это набор алгоритмов, которые занимается сопоставлением отдельных слов и словоформ в словаре (лексиконе, если быть точным) и выяснением грамматических характеристик слов.

Синтаксический парсер в ходе разбора предложения активно использует результаты морфологического анализа слов, хотя он может работать и без распознавателя. Тем не менее, разметка исходного текста грамматической информацией очень облегчает составление правил синтаксического анализа.

В качестве примера можно взять правило, которое определяет связь прилагательного и существительного. В соответствии с правилами русской грамматики, прилагательное, находящее слева от существительного, связывается с ним, если они стоят в одном и том же падеже (считая, конечно, локатив разновидностью предложного, а партитив разновидностью родительного падежей), числе и роде (для единственного числа). (*Пример: «На крутом (предложный, ед., муж.) берегу (локатив, ед., муж.)»*).

Это правило не является единственно возможным основанием для связывания прилагательного и существительного, но оно описывает наиболее частый случай.

Любая попытка формализации морфологии русского языка неизбежно сталкивается с такой проблемой, как совпадение разных грамматических форм. Например, глядя на слово «книги», нельзя однозначно сказать, является ли это слово множественным числом именительного падежа «старые книги лежат на столе», винительного падежа «выкинь эти старые книги», или единственным числом родительного падежа «начало книги». В число задач распознавателя входит нахождение всех возможных вариантов распознавания грамматической формы слова, а также исключение тех вариантов

распознавания, которые явно и однозначно запрещены в данном контексте. Данная проблема снятия омонимии может быть разбита на несколько отдельных задач, которые решаются своими специализированными алгоритмами.

Последняя задача распознавателя, которой будет уделено очень много времени, состоит в определении возможных грамматических признаков неизвестных слов. То есть, если анализируемое слово не найдено в лексиконе, то распознаватель должен сделать предположения о возможных вариантах его грамматической роли. Для решения этой задачи используется много разной информации, в том числе правила распознавания, которые будут подробно рассмотрены далее.

Результатом морфологического анализа слова является набор из следующих данных:

- к какой части речи может относиться слово,
- грамматической формой какой словарной статьи может являться слово,
- набор грамматических признаков для распознанной формы, например:

- 1) падеж для русских прилагательных и существительных,
- 2) время для русских глаголов,
- 3) глагольная форма для английских глаголов,
- 4) те грамматические атрибуты, которые определяются через атрибуты словарной статьи, если выполнено словарное распознавание.

- для нечеткой проекции вычисляется также достоверность распознавания.

Словарная морфология хороша тем, что обеспечивает высоконадежные результаты очень быстро, просто и точно. Главный ее недостаток – ограниченный набор распознаваемых слов. Если взять большой текстовый корпус и построить частотную гистограмму для слов в нем, то получится весь примечательное распределение - острый «пик» для небольшого количества

высокочастотных слов и очень длинный «хвост» из слов с частотностью около единицы. В логарифмической шкале, изображённой на рисунке 8, это выглядит так, по оси Y отложена частота слова, по X - отсортированные в порядке убывания частоты слова:

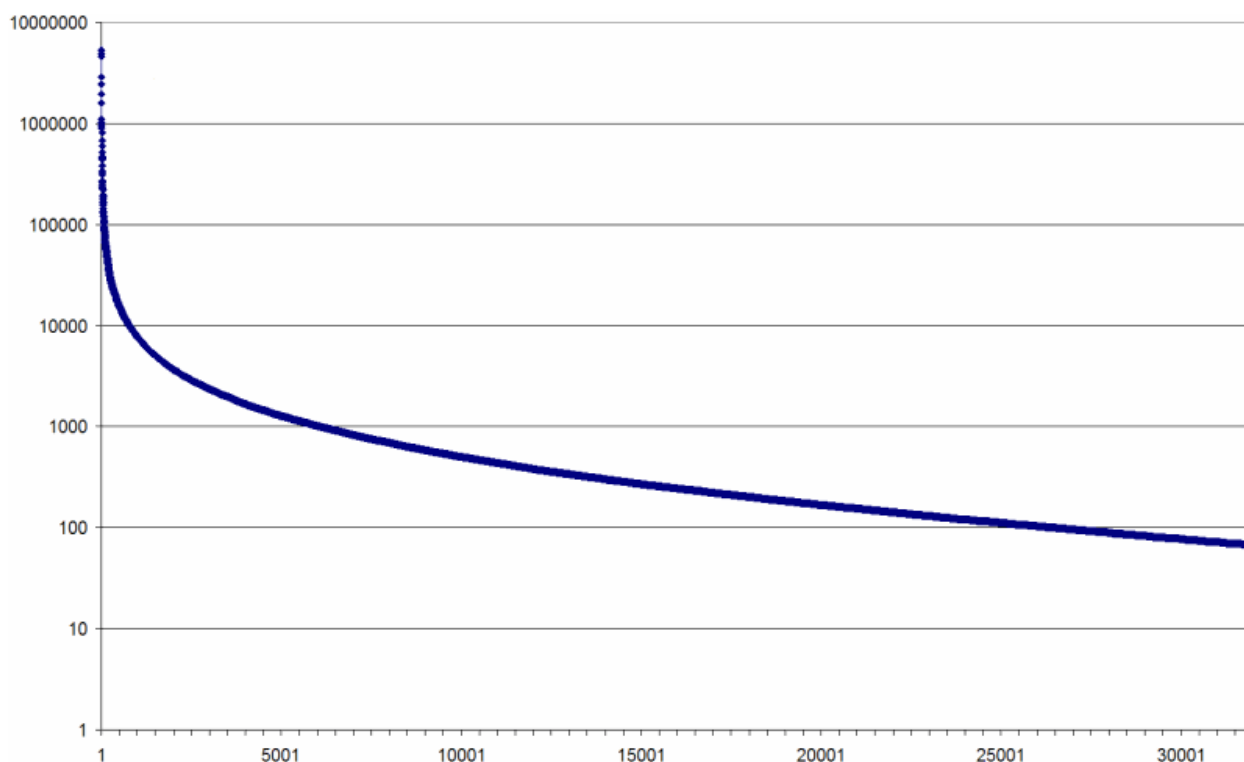


Рисунок 8 – График частот встречаемости словоформ русского языка

Местоимения обеспечивают примерно 15% общего числа форм. Словарная морфология для русского словаря с 3 миллионами форм успешно распознаёт в среднем примерно 97% для обычного (не узкоспециализированного) текста. Распознаванием оставшихся 3% занимается целый набор алгоритмов, в том числе правила распознавания по текстовому содержанию, например по окончаниям.

Результатом морфологического разбора является семантический граф.

Для разрешения омонимии в рамках данного предложения использован механизм ассоциации.

На основе вышеописанного алгоритма семантического анализа разработан программный функционал, осуществляющий функции

семантического анализа. На рисунке 9 приведена UML-диаграмма классов модуля.

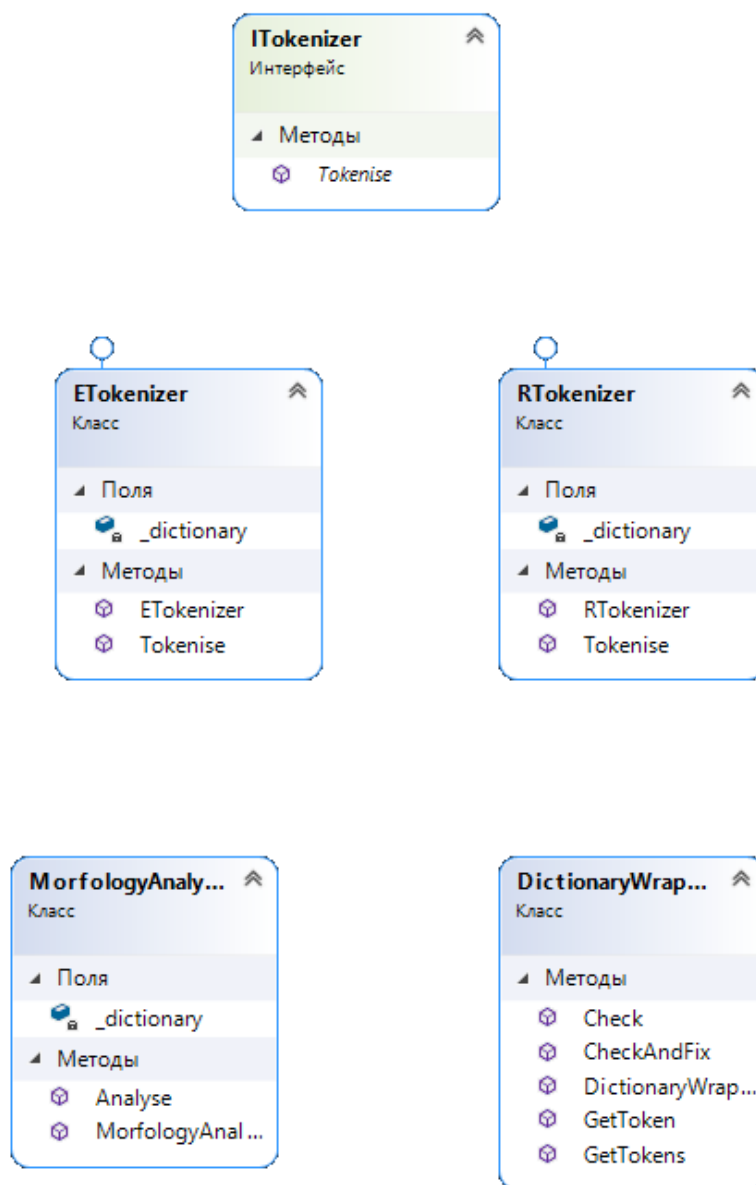


Рисунок 9 – Диаграмма классов модуля семантического анализа

## 2.2 Разработка визуального представления семантического дерева

После того, как завершён семантический анализ, и текстовая информация получена, необходимо осуществить её представление в удобном для пользователя виде. В соответствии с ранее созданной моделью графического представления семантической информации необходимо

разработать элементы управления Windows.Forms (Control), внутри которых будет производиться масштабирование, позиционирование и отображение графических объектов. Было решено выбрать две наиболее удобные формы графического представления – дерево и граф, а также осуществить возможность сохранения модели в формате XML-документа определённого формата для дальнейшей обработки сторонними приложениями. С целью экономии пространства и во избежание излишней перегруженности пользовательского интерфейса было решено разместить оба представления на правой панели программы. Переключение между ними осуществляется с использованием элемента пользовательского интерфейса «ComboBox», в котором пользователь может выбрать удобную ему форму представления.

Для отображения модели в виде дерева наиболее подходящим решением было использовать стандартный элемент управления Windows.Forms TreeView. Данный объект пользовательского интерфейса предоставляет возможность скрывать и раскрывать ветви дерева, перемещаться по узлам дерева с помощью клавиатуры, а также даёт наглядное и ясное представление семантической структуры модели. Чтобы отобразить модель в виде дерева, необходимо загрузить в программу изображение, содержащее текст, выбрав соответствующий пункт в разделе «Файл» в главном меню, затем, выбрав язык из выпадающего меню в правом нижнем углу окна, нажать кнопку «Преобразовать изображение в текст», дождаться отображения текста в панели «Текст», а затем, убедившись, что в правом нижнем углу в панели «Вид» выбран пункт «Дерево», нажать кнопку «Генерировать модель». Пример представления в виде дерева приведён на рисунке 10.

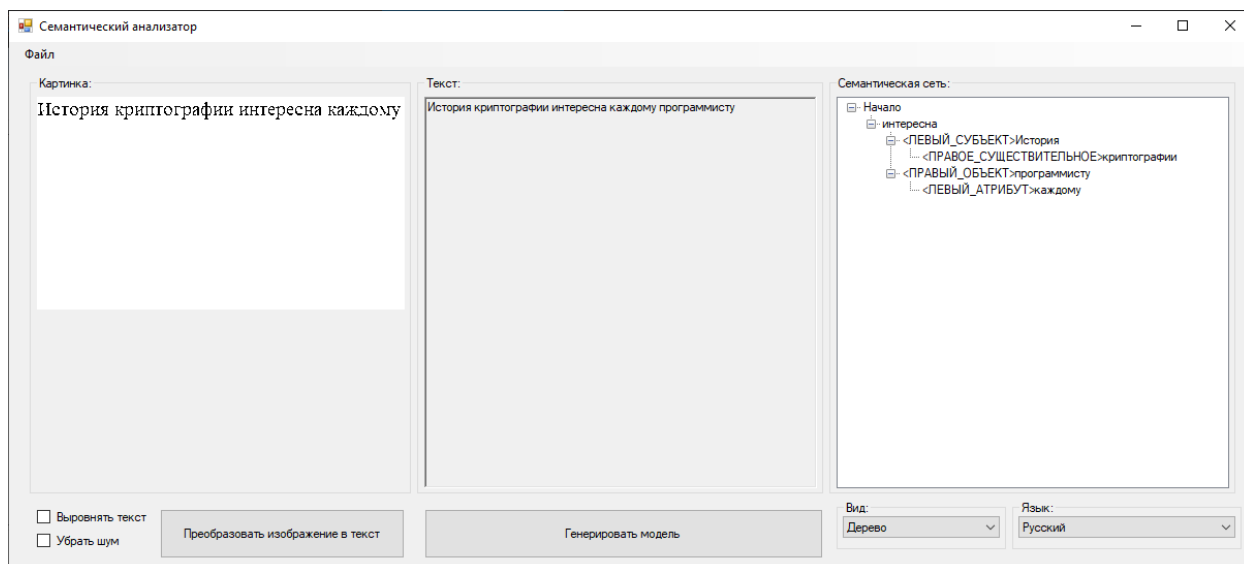


Рисунок 10 – Представление "Дерево"

При двойном клике левой кнопки мыши по любому узлу дерева, кроме корневого, в отдельном окне отображаются свойства элементов семантической сети. Поскольку набор свойств для каждого узла дерева различается, для их динамического отображения используется элемент управления DataGridView, позволяющий получить наглядное представление этих свойств в виде таблицы. Пример свойств слова «программисту» представлен на рисунке 11.

| Свойства       |                 |
|----------------|-----------------|
| Название       | Значение        |
| Слово          | программисту    |
| Часть речи     | СУЩЕСТВИТЕЛЬНОЕ |
| Падеж          | ДАТ             |
| Число          | ЕД              |
| Род            | МУЖ             |
| Одушевлённость | ОДУШ            |

Рисунок 11 – Представление свойств слова

Для отображения модели в виде графа необходимо разработать новый элемент управления, унаследованный от класса Control. GraphView поддерживает необходимый функционал по автоматическому размещению, масштабированию и отображению элементов семантической модели. Данный

элемент пользовательского интерфейса позволяет отображать свойства, в отдельном окне, аналогично рисунку 2, с использованием табличного представления DataGridView. Чтобы отобразить модель в виде графа, необходимо загрузить в программу изображение, содержащее текст, выбрав соответствующий пункт в разделе «Файл» в главном меню, затем, выбрав язык из выпадающего меню в правом нижнем углу окна, нажать кнопку «Преобразовать изображение в текст», дождаться отображения текста в панели «Текст», а затем, убедившись, что в правом нижнем углу в панели «Вид» выбран пункт «Граф», нажать кнопку «Генерировать модель». Пример представления в виде графа приведён на рисунке 12.

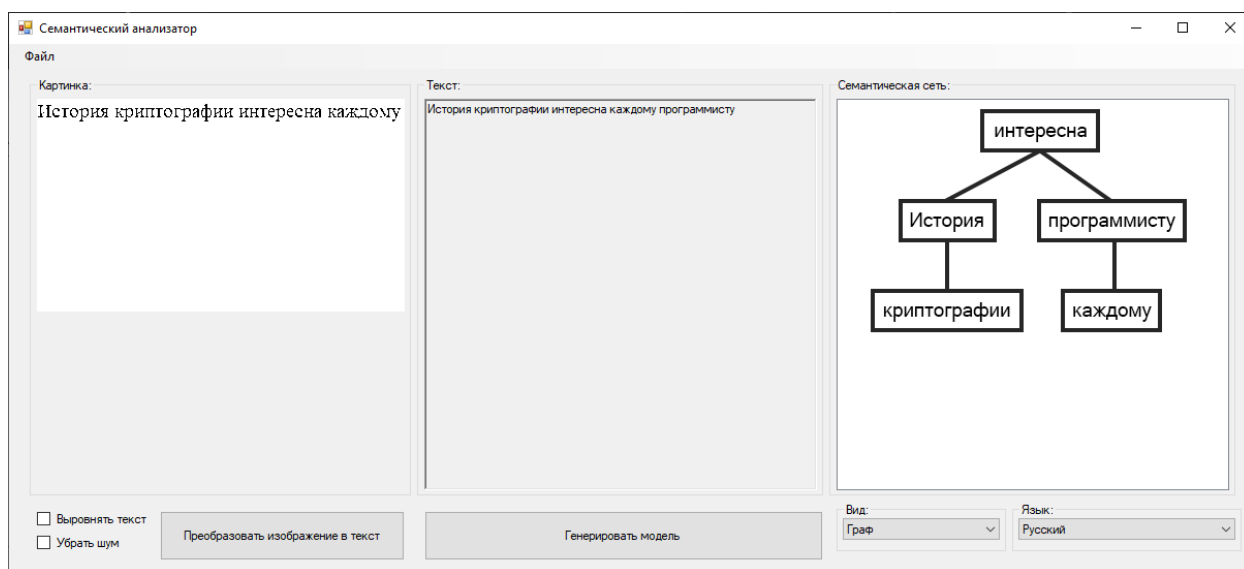


Рисунок 12 – Представление "Граф"

Поскольку применение большого количества элементов управления ведёт к росту числа используемых GDI-объектов, что уменьшает скорость работы алгоритмов, увеличивает затраты оперативной памяти и может повлечь за собой ошибки в работе программы, было решено создать библиотеку графических объектов, позволяющих использовать один GDI-объект Graphics, получаемый при вызове метода Invalidate() класса GraphView. Это позволило существенно уменьшить использование GDI-объектов, ускорить работу алгоритмов обработки пользовательских действий, а также снизить нагрузку на графическое ядро.

Базовым классом библиотеки графических элементов является класс `DrawingObject`. Данный класс включает в себя функционал, который позволяет:

- динамически обрабатывать взаимодействие пользователя с элементом управления,
- перехватывать события,
- изменять положение и масштаб отрисовки представления узла семантического дерева.

В рамках реализации визуальной модели наследником класса `DrawingObject` является класс `WordView`, расширяющий его функционал с возможностью отрисовки содержащейся в нём лексемы. За размещение представлений лексем отвечает класс `ViewAdapter`, в котором реализован функционал работы с объектами типа `WordView` такой, как:

- методы автоматического размещения,
- масштабирования
- позиционирования.

Помимо вышеупомянутых классов также добавлен класс `LinkView`, связывающий представления лексем друг с другом. Диаграмма классов библиотеки визуализации семантической модели представлена на рисунке 13.

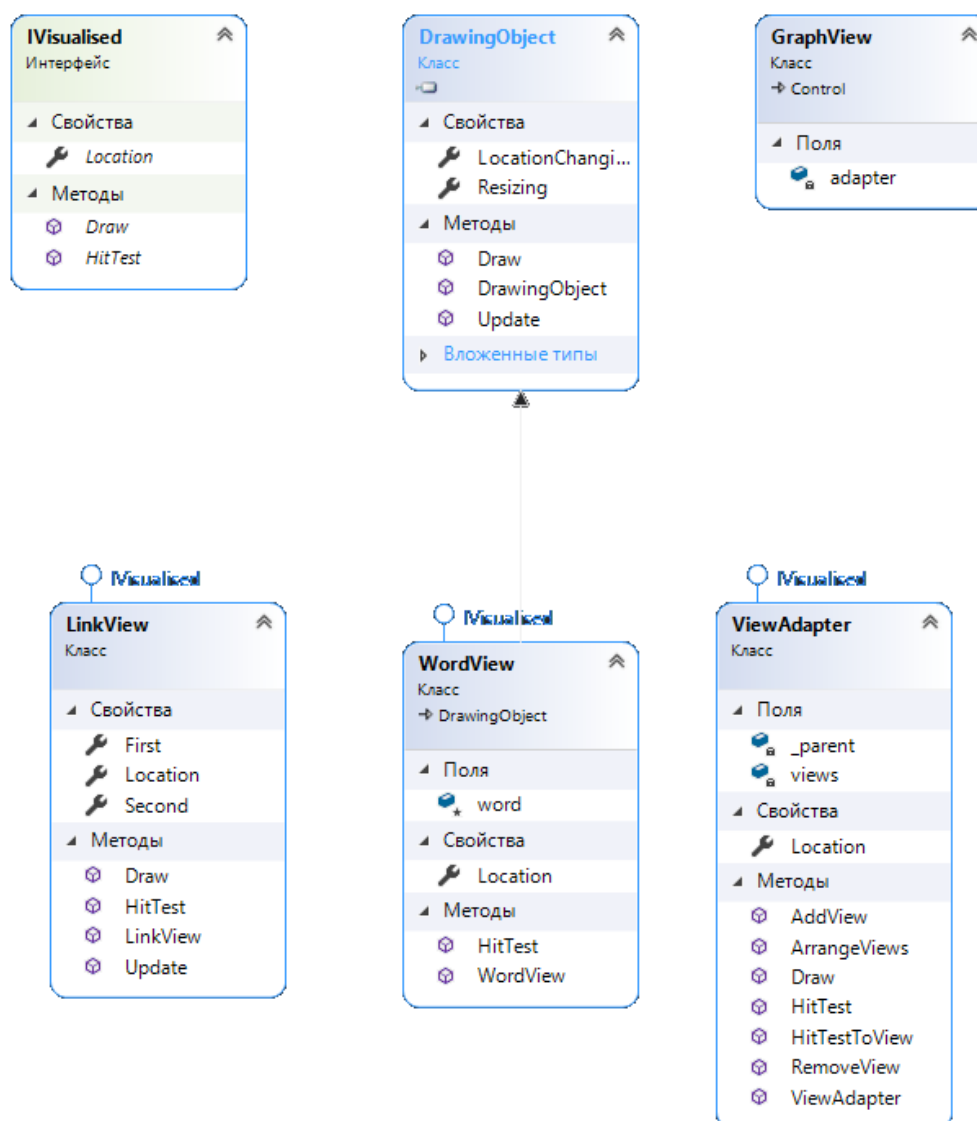


Рисунок 13 – Диаграмма классов библиотеки графического представления графа

Ещё одним способом представления семантической информации является XML-документ. Формат XML позволяет представлять иерархические структуры в одинаково удобном для чтения как пользователем, так и различными программными средствами виде. Иерархическая структура семантической модели хорошо подходит для сериализации в виде XML-документа, который в дальнейшем может обрабатываться другими программами. Пример XML-документа, полученного в результате сохранения приведённой ранее в примерах модели представлен на рисунке 14.

```

1 <?xml version="1.0"?>
2 <model>
3   <predicate type="verb" val="интересна" case="NOM" number="SIN" kind="FEM">
4     <subject type="noun" val="История" case="NOM" number="SIN" kind="FEM" anim="0" enum="1">
5       <noun type="noun" val="криптографии" case="GEN" number="SIN" kind="FEM" anim="0" enum="0"/>
6     </subject>
7     <object type="noun" val="программисты" case="DAT" number="SIN" kind="MAS" anim="1" enum="1">
8       <attribute type="adjective" val="каждому" case="DAT" number="SIN" kind="MAS"/>
9     </object>
10   </predicate>
11 </model>

```

Рисунок 14 – XML-документ модели

## 2.3 Разработка модели семантического дерева

Полученное в конце морфологического разбора семантическое дерево, представляет собой список объектов, реализующих интерфейс IWord. Данный интерфейс является абстрактным представлением узла семантического дерева и содержит в себе единственное поле «Text» типа «string», возвращающее строковое содержание данной лексемы. Все объекты данной иерархии можно разделить на два категории: «Связь» и «Лексема».

Объекты типа «Связь» («Relation») имеют достаточно простую структуру, отличающуюся от родительского интерфейса лишь полем перечисления, отвечающим за тип связи, и ссылками на связываемые лексемы. Поэтому необходимость создания разветвлённых иерархий классов для них отсутствует, и реализация интерфейса «IWord» является наилучшим возможным решением.

Для объектов типа «Лексема» характерно наличие большого числа как общих (род, падеж, число и пр.), так и специфических (наклонение, время и пр.). Следовательно, необходимо создать для них соответствующие классы. В качестве родительского для всех слов создан интерфейс «ILexem», в свою очередь реализующий интерфейс «IWord». Поскольку разные части речи могут иметь сходные свойства, решено создать иерархию интерфейсов, унаследованных от «IProperty», которые будут реализовывать объекты, реализующие интерфейс «ILexem». Данная архитектура позволит осуществлять поиск по словам, содержащим то или иное свойство.

Для осуществления операций над объектами модели используется класс «HelperFunctions». Операции выполняются с использованием поведенческого паттерна «Команда», позволяющим производить отмену ошибочно

произведённых ранее действий. Такой подход оставляет широкие возможности для дальнейшего расширения и модифицирования программы, например, в области анализа грамматик других естественных языков. Помимо прочего, данный класс реализует возможность сериализации узлов семантического дерева, что делает возможным сохранение модели в XML-документ для последующей автоматической обработки другими приложениями, такими как рекомендательные алгоритмы.

Диаграмма классов, описывающая представленную модель, приведена на рисунке 15.

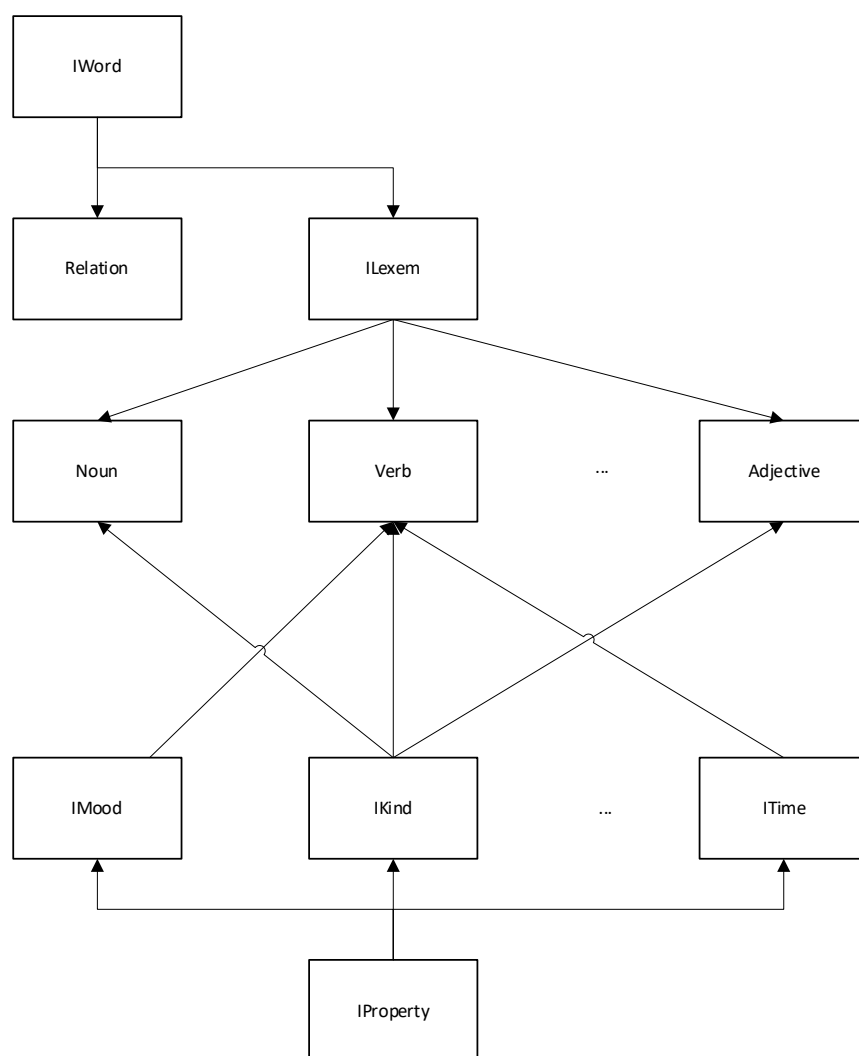


Рисунок 15 – Диаграмма классов модели

## 2.4 Выводы

В рамках работ, описанных в этом разделе, была проведена поэтапная разработка всех компонентов реализуемого приложения, были подробно

описаны алгоритмы токенизации и морфологического разбора, а также подобраны необходимые средства реализации графического интерфейса, отвечающие функциональным требованиям к приложению.

### **3 Описание работы приложения**

#### **3.1 Руководство пользователя**

##### **3.1.1 Назначение «Семантический анализатор»**

Приложение «Семантический анализатор» предназначено для семантического анализа графического изображения, содержащего текст, и его представление в виде дерева, графа или XML-документа для последующей обработки сторонними приложениями.

##### **3.1.2 Условия выполнения «Семантический анализатор»**

ПО «Семантический анализатор» может быть использовано самостоятельно. В ПО предусмотрены возможности для создания XML-документа, описывающего семантическую модель, а также различные возможности считывания, представления и обработки семантической информации.

Для выполнения «Семантический анализатор» необходима операционная среда Windows XP, 7, 8 или 10.

Изображения, указанные на рисунках, приведены в качестве примера.

Для работы программы необходимы следующие минимальные характеристики ПК:

- частота процессора: 1 ГГц,
- ОЗУ: 1024 Мб,
- 4.5 Гб свободного места на жёстком диске.

##### **3.1.3 Состав «Семантический анализатор»**

В состав приложения «Семантический анализатор» входят следующие элементы:

- ПО «Семантический анализатор»,
- данное руководство пользователя.

##### **3.1.4 Подготовка к работе**

Для подготовки ПО к работе необходимо загрузить ОС Windows и запустить исполняемый файл приложения “SemanticAnalyser.exe”.

### 3.1.5 Описание работы с программой «Семантический анализатор»

Ниже приводится описание пользовательского интерфейса.

Работа оператора с ПО «Семантический анализатор» ведется с помощью мыши и клавиатуры.

На рисунке 16 показано основное окно программы после запуска. Стандартный размер основного окна: 1221:553 пикселей.

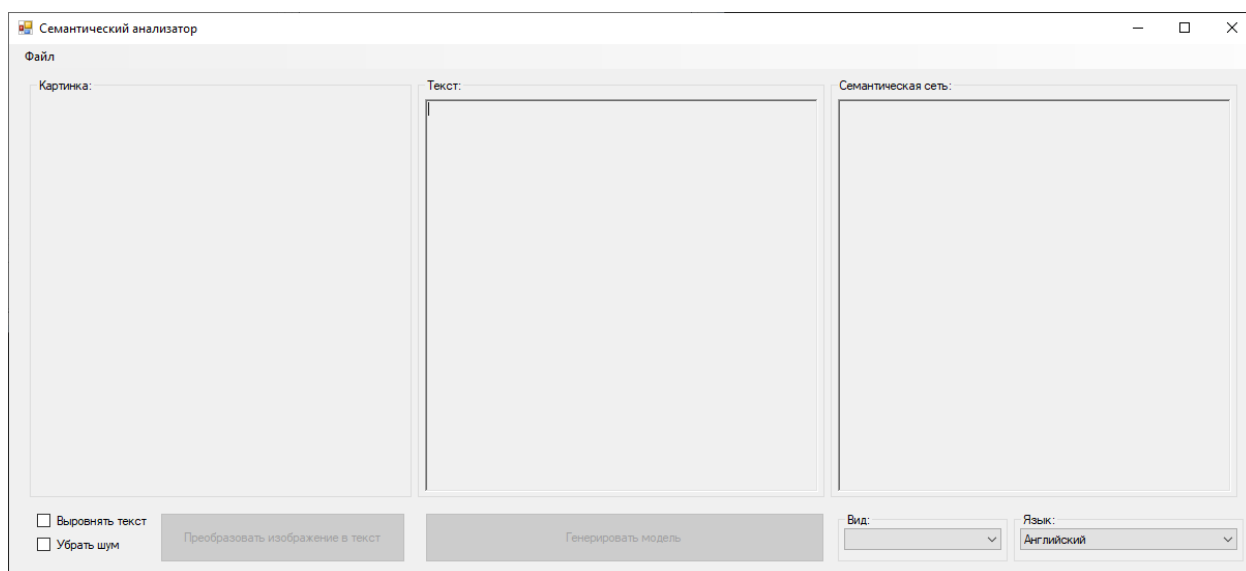


Рисунок 16 – Стандартный размер окна программы

На экране окна программы расположены:

- панель меню,
- панель параметров улучшения качества текста,
- кнопки «Преобразовать изображение в текст» и «Генерировать модель»,
- панель изображения,
- панель текста,
- панель модели семантической сети,
- панель параметров вида и языка.

Панель меню расположена в верхней части главного окна приложения (рисунок 17).

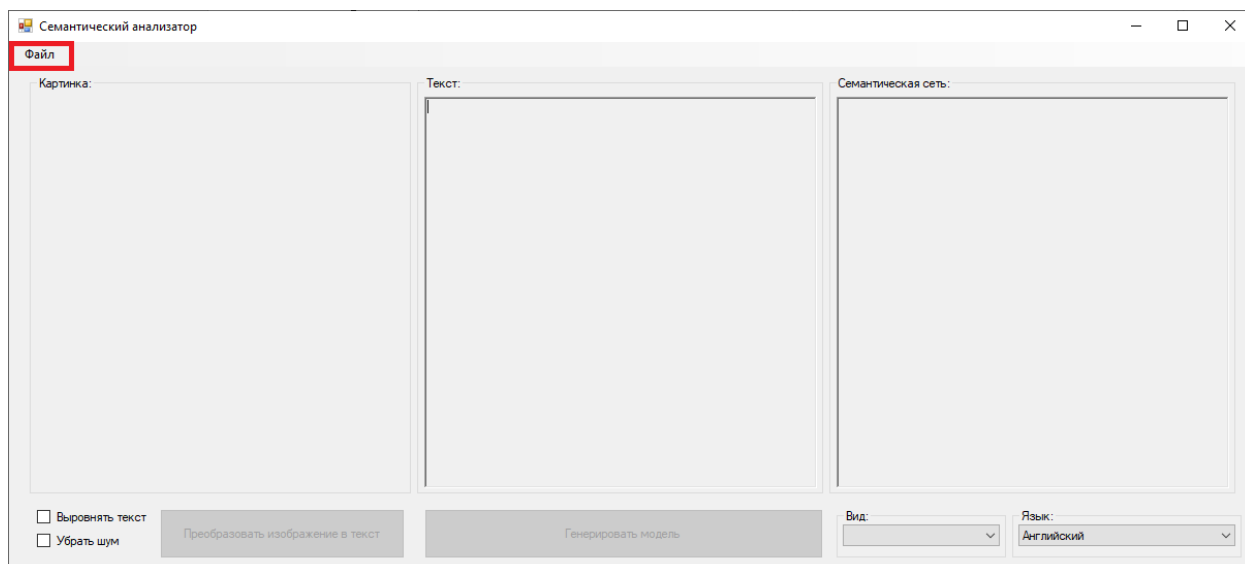


Рисунок 17 – Панель меню

Панель состоит из пункта меню «Файл» (рисунок 18).

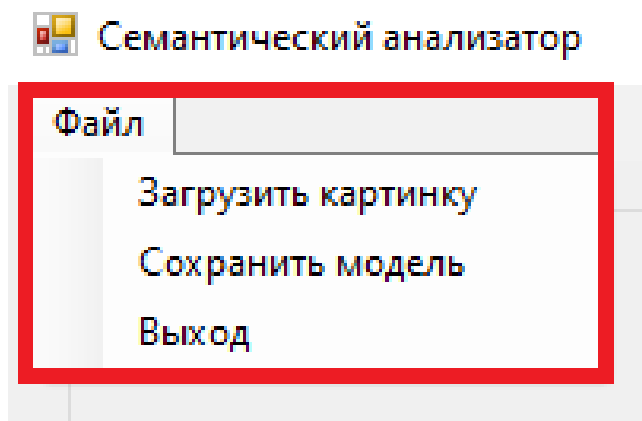


Рисунок 18 – Функции меню для работы с файлом

Подпункты меню «Файл» выполняют соответственно следующие функции:

- загрузка изображения,
- сохранение модели в виде XML-документа,
- выход из приложения.

Панель параметров улучшения текста расположена в нижней левой части главного окна приложения ниже панели изображения (рисунок 19).

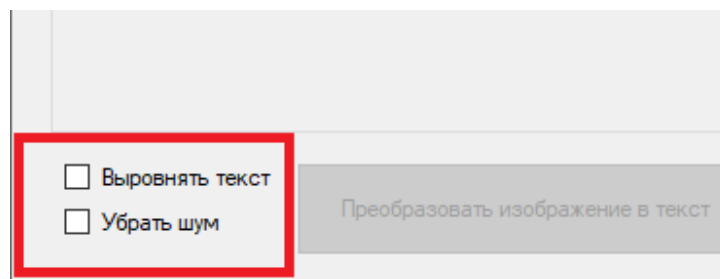


Рисунок 19 – Панель параметров улучшения текста

Панель состоит из следующих элементов:

- выровнять текст,
- убрать шум.

Кнопки «Преобразовать изображение в текст» и «Генерировать модель» расположены справа от панели параметров улучшения текста (рисунок 20). При запуске приложения они неактивны, но после загрузки изображения, становится активной кнопка «Преобразовать изображение в текст», а при наличии текста в панели текста становится активной кнопка «Генерировать модель».

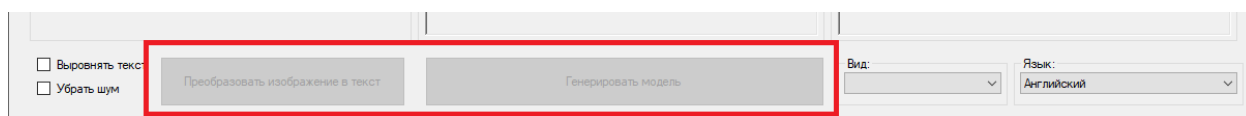


Рисунок 20 – Кнопки «Преобразовать изображение в текст» и «Генерировать модель»

Панель изображения расположена в левой части окна приложения выше панели параметров улучшения качества текста (рисунок 21). В панели отображается загруженное в программу изображение.

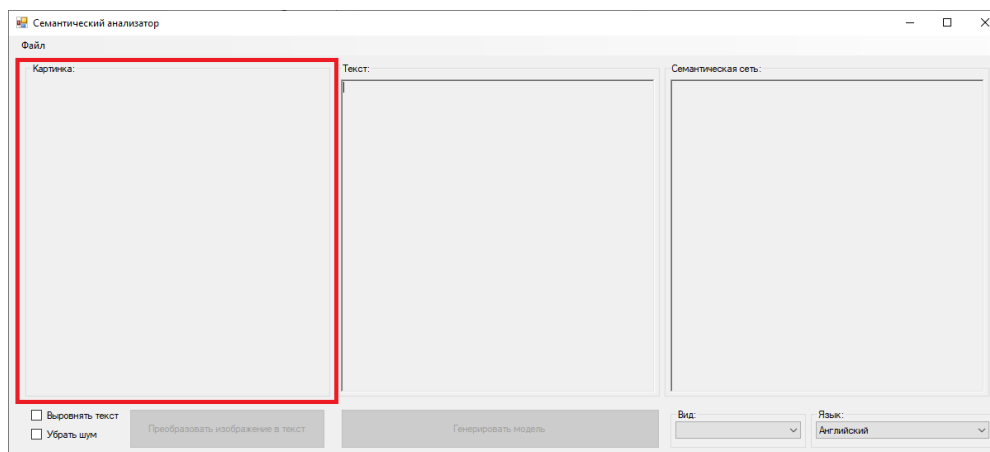


Рисунок 21 – Панель изображения

Панель текста (рисунок 22) расположена справа от панели изображения и служит для отображения текста, полученного в результате работы библиотеки IronOCR. Программа не поддерживает ручной ввод текста, поэтому эта панель является доступной только для чтения.

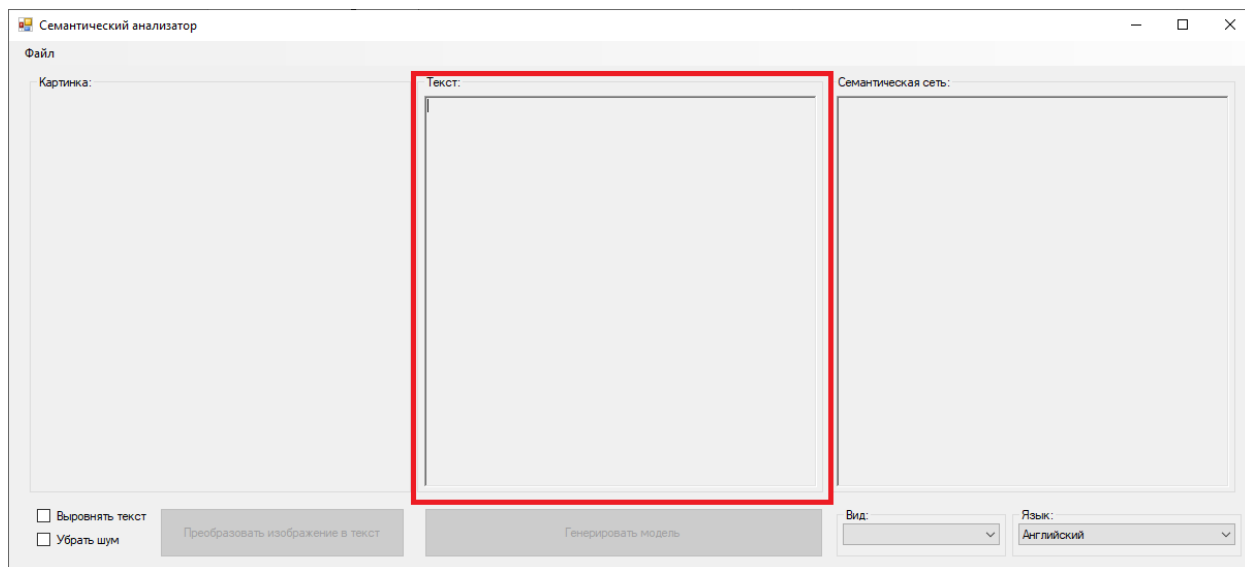


Рисунок 22 – Панель текста

Полученная в результате семантического анализа модель отображается в панели семантической сети (рисунок 23) в правой части главного окна приложения. В зависимости от выбранного в списке «Вид» варианта вывода, она может быть представлена в виде дерева или графа.

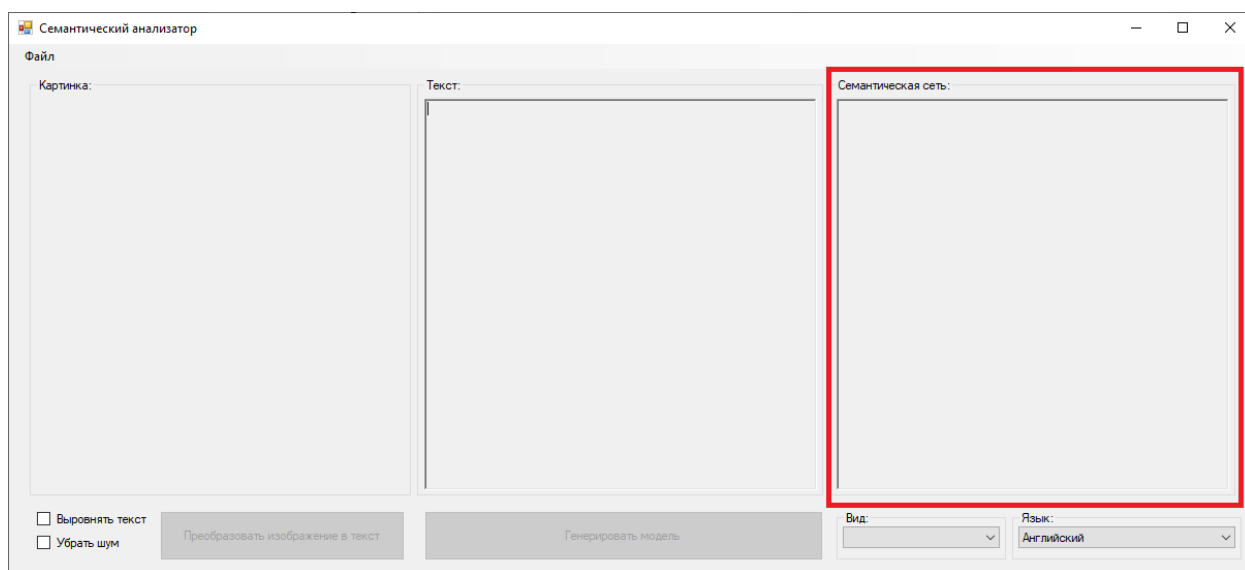


Рисунок 23 – Панель семантической сети

Элементы панели параметров вида и языка отвечают за способ представления семантической модели и выбор языка для оптического

распознавателя текста. Панель представляет собой два элемента управления ComboBox, позволяющие пользователю выбрать одну из нескольких доступных опций.

Опции элемента «Вид» отвечают за форму представления семантической сети и могут принимать следующие значения:

- дерево,
- граф.

Опции элемента «Язык» отвечают за то, на каком языке будет считываться и обрабатываться текстовая информация.

На рисунке 24 изображена панель параметров вида и языка:

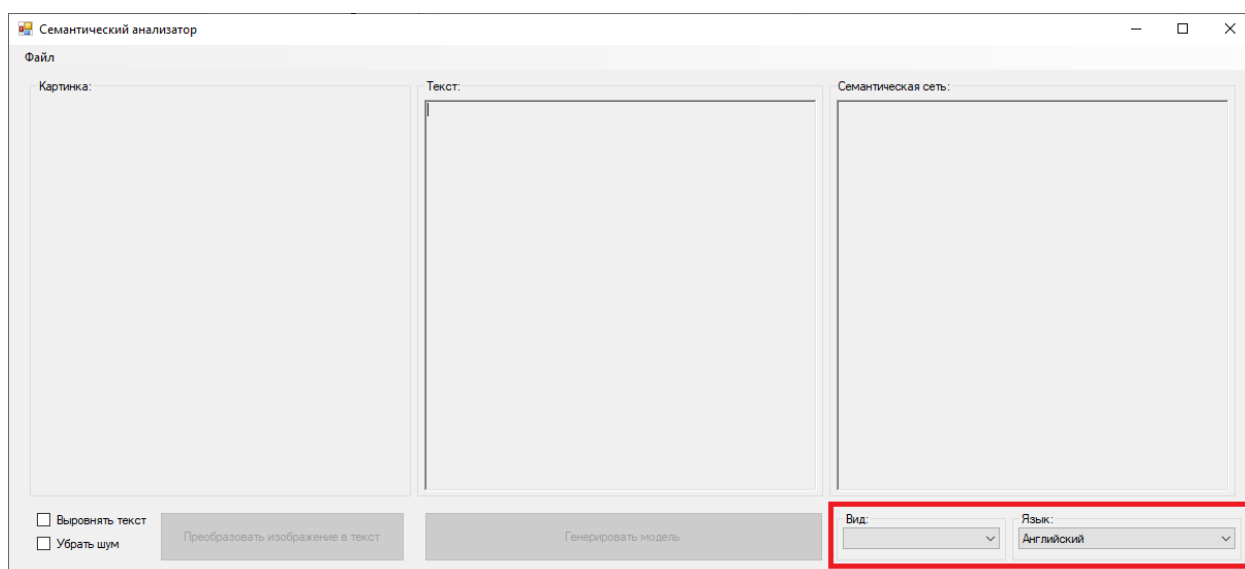


Рисунок 24 – Панель параметров вида и языка

Работа с программой «Семантический анализатор» осуществляется следующим образом.

Запуск программы «Семантический анализатор» осуществляется запуском файла «SemanticAnalyser.exe». После успешного запуска программы на экране появится главное окно программы «Семантический анализатор» (рисунок 25).

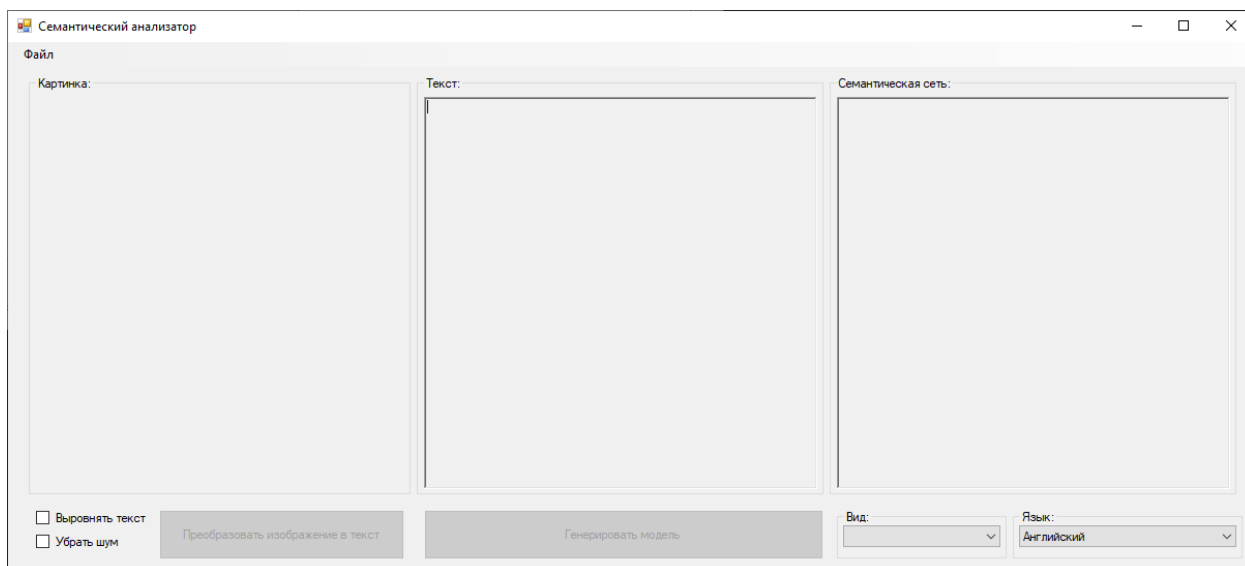


Рисунок 25 – Главное окно программы «Семантический анализатор»

Попытка сохранить пустую семантическую модель не повлияет на работу ПО.

Чтобы начать работу непосредственно с графическим файлом, необходимо загрузить файл формата .bmp, содержащий текстовую информацию.

Для загрузки изображения необходимо выбрать соответствующий пункт в разделе «Файл» в панели меню, после чего в открывшемся окне выбора файла (рисунок 26) выбрать необходимое изображение. Результат загрузки изображения приведён на рисунке 27. После загрузки изображения кнопка «Преобразовать изображение в текст» станет активной.

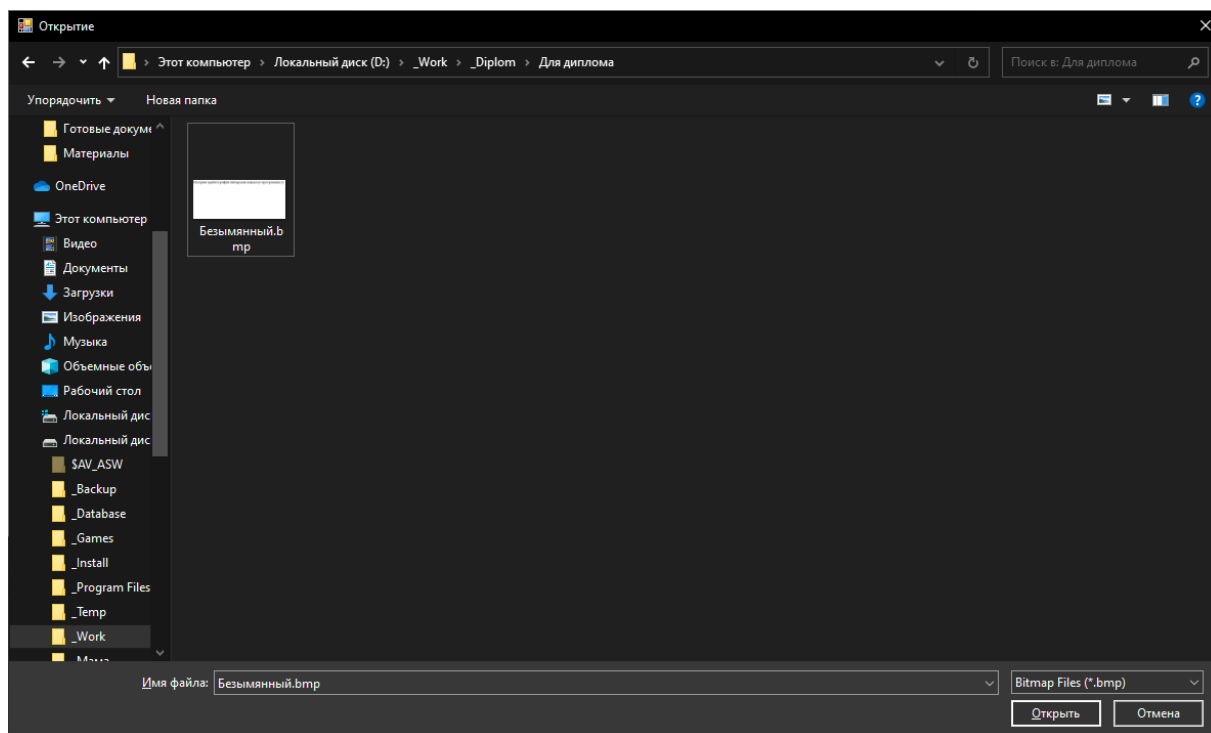


Рисунок 26 – Окно выбора файла

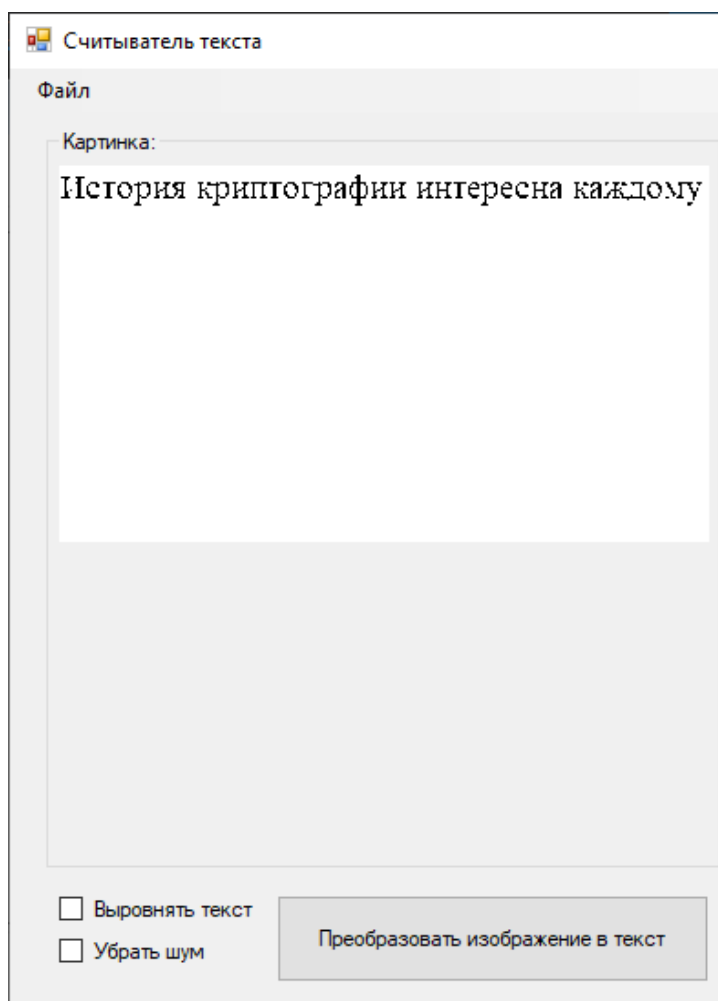


Рисунок 27 – Загрузка изображения

Рассмотрим преобразование графической информации в текстовую.

После загрузки изображения необходимо считать с него текстовую информацию. Для этого нужно:

- при необходимости выбрать необходимые опции в панели параметров улучшения качества текста,
- в панели параметров вида и языка выбрать язык текста,
- нажать на кнопку «Преобразовать изображение в текст» в нижней части окна.

В результате выполнения этой операции в средней части окна будет представлен результат работы оптического распознавателя текста. Результат считывания текста с изображения, приведённого на рисунке 27, показан на рисунке 28.

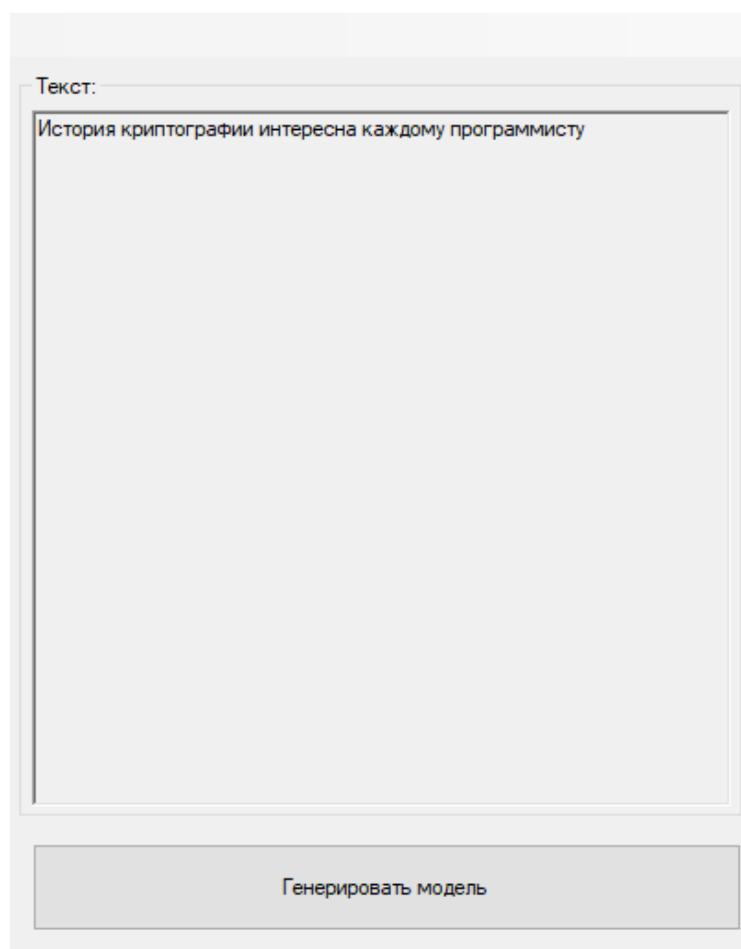


Рисунок 28 – Считанный с изображения текст

Создание модели осуществляется следующим образом.

После того, как был считан текст, станет доступна возможность создания модели. Для этого необходимо предварительно выбрав форму представления этой модели в панели параметров вида и языка нажать на кнопку «Генерировать модель» в нижней части окна. В результате выполнения этой операции в правой части экрана отобразится представление модели в виде графа (рисунок 29) или дерева (рисунок 30), в зависимости от выбранной опции вида представления. Для сохранения модели необходимо выбрать соответствующий пункт в разделе «Файл» в панели меню, после чего в открывшемся окне сохранения файла (рисунок 31) указать путь, где должен быть сохранён XML-документ, описывающий модель. В случае сохранения пустой модели на экран будет выведено сообщение об ошибке.



Рисунок 29 – Представление модели в виде графа

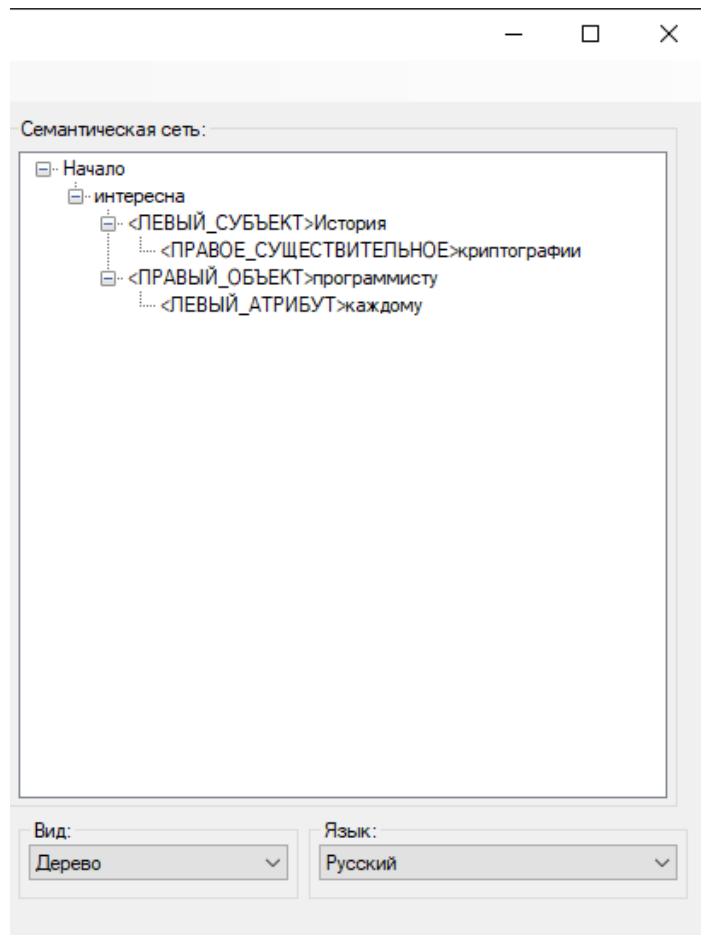


Рисунок 30 – Представление модели в виде дерева

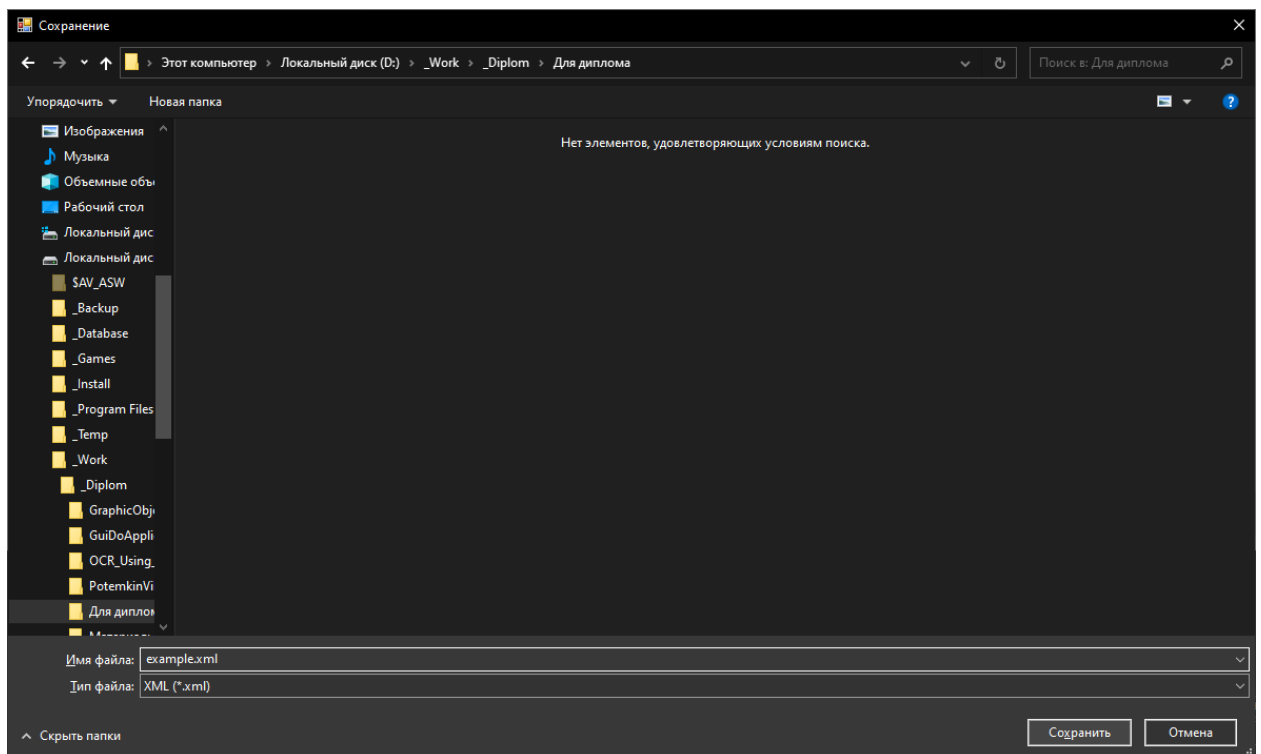
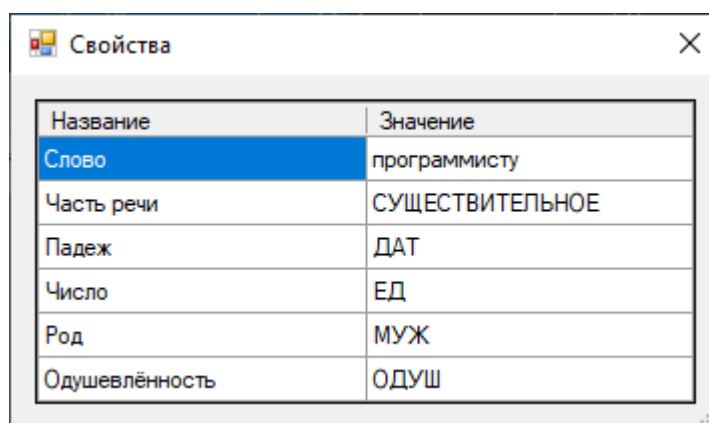


Рисунок 31 – Окно сохранения файла

Свойства узлов семантической сети представлены следующим образом.

Приложение поддерживает возможность просмотра свойств каждого узла семантической сети (кроме узлов «BEGIN» и «END»). Для того, чтобы увидеть свойства узла, необходимо в представлении «Граф» дважды кликнуть по узлу графа, или в представлении «Дерево» дважды кликнуть по элементу дерева. В открывшемся окне будет отображена таблица свойств выбранного узла. Пример окна свойств представлен на рисунке 32.



| Название       | Значение        |
|----------------|-----------------|
| Слово          | программисту    |
| Часть речи     | СУЩЕСТВИТЕЛЬНОЕ |
| Падеж          | ДАТ             |
| Число          | ЕД              |
| Род            | МУЖ             |
| Одушевлённость | ОДУШ            |

Рисунок 32 – Таблица свойств узла «программисту»

Ниже рассмотрены сообщения об ошибках, возникающих в процессе работы приложения.

В ходе работы приложения на разных этапах работы могут возникнуть ошибки. Их можно разделить на три типа:

- 1) Ошибки, возникшие при считывании текста
- 2) Ошибки, возникающие в процессе токенизации
- 3) Ошибки, возникающие в процессе сохранения XML-документа

Ошибки, возникающие при считывании текста

Примеры ошибок приведены на рисунках 33-35.

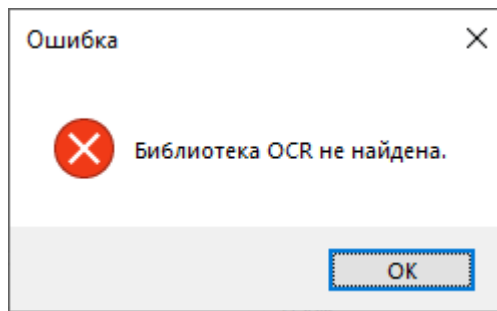


Рисунок 33 – Ошибка при считывании текста – библиотека OCR не найдена

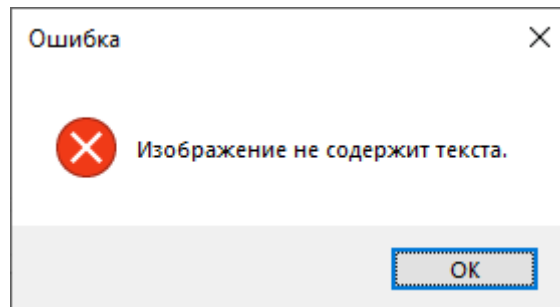


Рисунок 34 – Ошибка при считывании текста – изображение не содержит текста

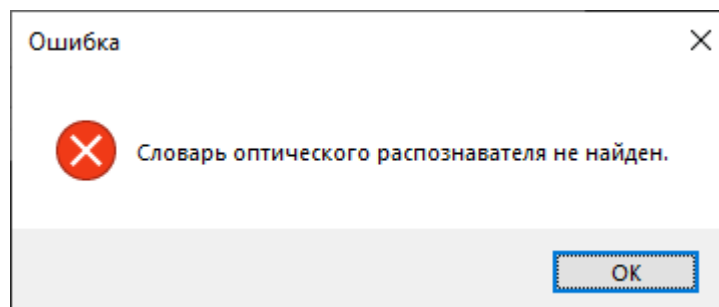


Рисунок 35 – Ошибка при считывании текста – библиотека OCR не найдена

Рассмотрим ошибки, возникающие в процессе токенизации.

Примеры ошибок приведены на рисунках 36-38.

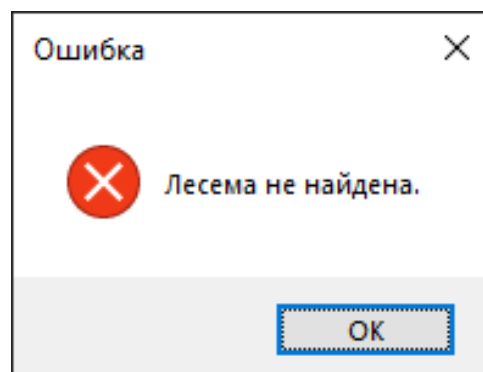


Рисунок 36 – Ошибка токенизации – лексема не найдена

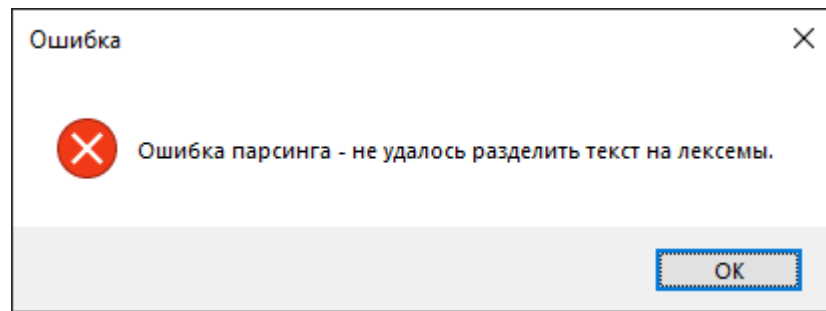


Рисунок 37 – Ошибка токенизации – не удалось разбить текст на лексемы

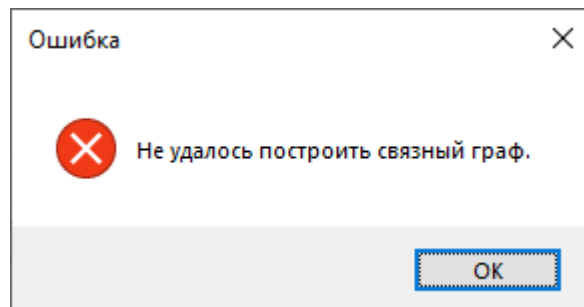


Рисунок 38 – Ошибка токенизации – не удалось построить связный граф

Ошибки, возникающие в процессе сохранения XML-документа, предсталены ниже

Примеры ошибок приведены на рисунках 39, 40.

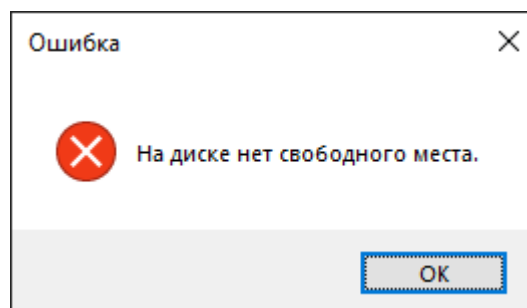


Рисунок 39 – Ошибка сохранения XML-документа – нет свободного места на диске

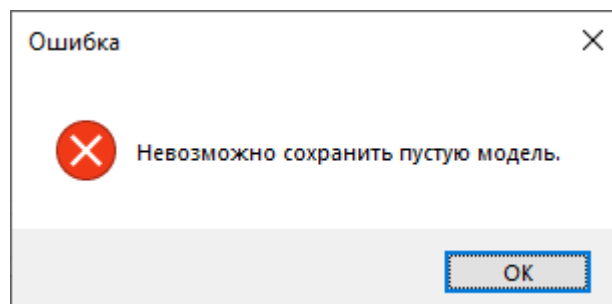


Рисунок 40 – Ошибка сохранения XML-документа – сохранение пустой модели

Диалоговые сообщения, возникающие в ходе работы приложения, представлены ниже.

Во время работы с программой могут возникнуть ситуации, в которых от пользователя потребуется вручную добавлять в словарь лексера (токенизатора) новые лексемы. Часто возникает ситуация, когда при считывании текста одно или несколько слов оказываются написанными с грамматическими ошибками. В таких случаях на экран выводится диалоговое окно (рисунок 41), содержащее текстовое поле с незнакомой словарю лексемой, список возможных вариантов замены, а также кнопки «Принять» и «Отмена».

Пользователю необходимо выбрать, является ли это слово одной из предложенных лексем, или определяет новую лексему. Для того, чтобы исправить грамматическую ошибку, пользователь должен выбрать один из предложенных вариантов и нажать кнопку «Принять». Для того, чтобы добавить новую лексему, следует исправить грамматическую ошибку в текстовом поле (если она есть) и нажать кнопку «Отмена».

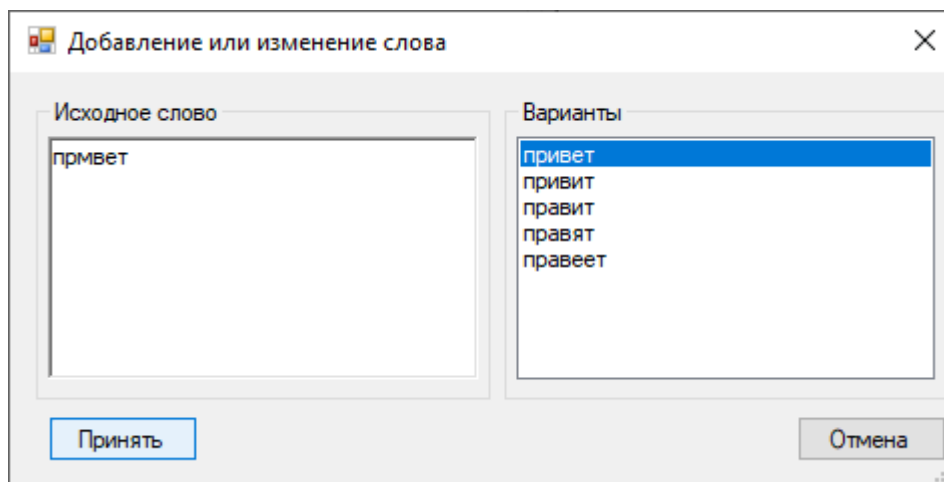


Рисунок 41 – диалоговое окно добавления или изменения лексемы

### 3.2 Тестирование приложения

Для того, чтобы убедиться, что приложение не содержит ошибок, необходимо проверить его работоспособность на различных наборах тестовых данных. Для этого было проведено функциональное тестирование приложения.

### 3.2.1 Текст на русском языке

В качестве примера изображения, содержащего текст на русском языке, было использовано изображение, приведённое на рисунке 42:

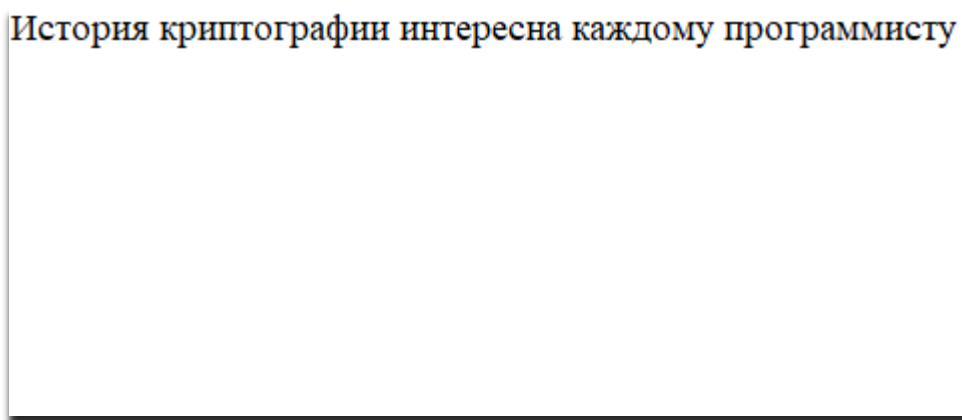


Рисунок 42 – Пример изображения, содержащего текст на русском языке

В результате загрузки, считывания текста и генерации семантической модели в виде графа и дерева, были получены результаты, представленные на рисунках 43,44.

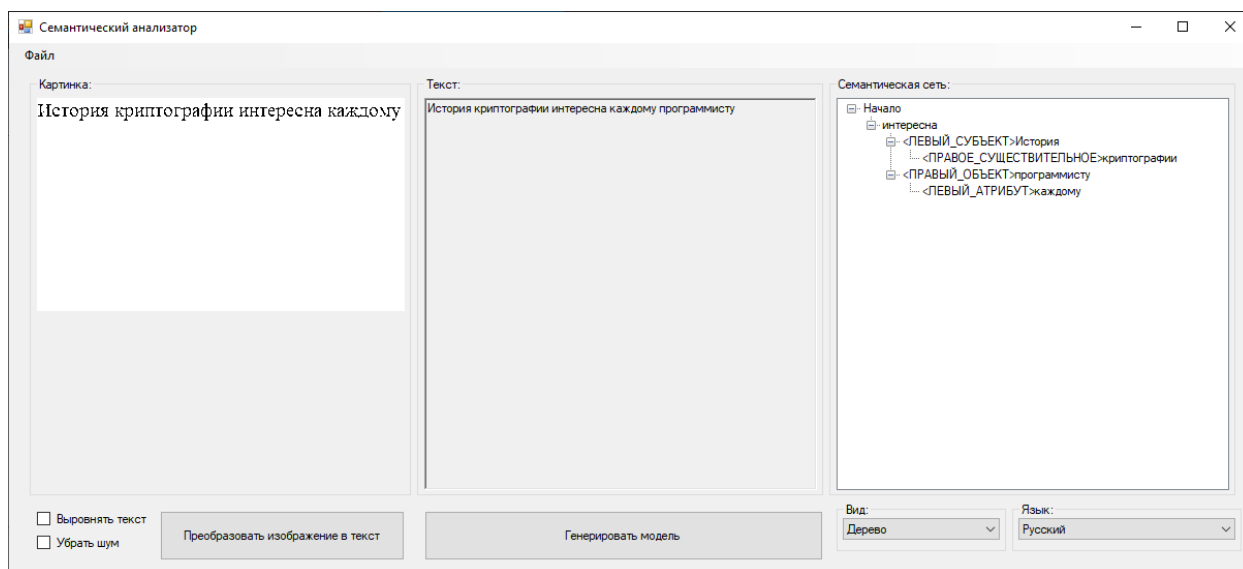


Рисунок 43 – Представление полученной модели в виде дерева

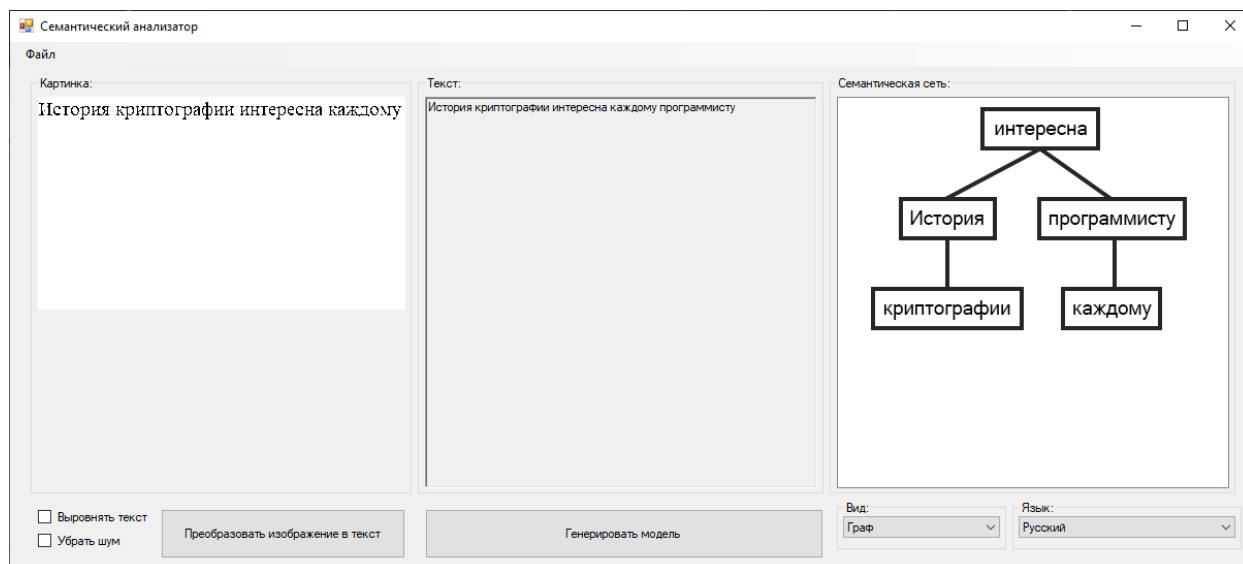


Рисунок 44 – Представление полученной модели в виде графа

После сохранения модели получен XML-документ, содержащий описание модели. Текст полученного файла представлен на рисунке 45.

```

1  <?xml version="1.0"?>
2  <model>
3    <predicate type="verb" val="интересна" case="NOM" number="SIN" kind="FEM">
4      <subject type="noun" val="История" case="NOM" number="SIN" kind="FEM" anim="0" enum="1">
5        <noun type="noun" val="криптографии" case="GEN" number="SIN" kind="FEM" anim="0" enum="0"/>
6      </subject>
7      <object type="noun" val="программисту" case="DAT" number="SIN" kind="MAS" anim="1" enum="1">
8        <attribute type="adjective" val="каждому" case="DAT" number="SIN" kind="MAS"/>
9      </object>
10   </predicate>
11 </model>

```

Рисунок 45 – XML-документ

Таким образом, при наличии осмысленного текста на русском языке, приложение работает без ошибок.

### 3.2.2 Отсутствие словаря

Как было сказано в предыдущих разделах, приложение в своей работе использует словари русского и английского языка для более точного соотнесения лексем. Очевидно, что при отсутствии словарей, работа приложения происходит некорректно. При попытке загрузить приложение вызывается функция проверки всех необходимых для его работы компонентов. В случае отсутствия одного из них, на экран будет выведено сообщение об ошибке, уведомляющее пользователя о том, какой компонент отсутствует. Результат загрузки приложения, из каталога которого был удалён словарь, представлен на рисунке 46.

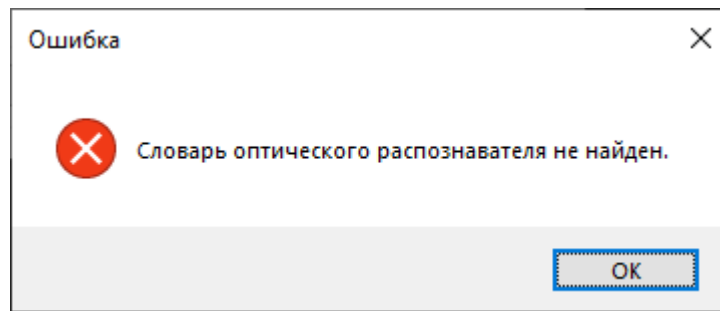


Рисунок 46 – Ошибка загрузки словаря

Таким образом, приложение не позволит пользователю осуществить семантический анализ при отсутствии необходимых для этого компонентов ПО.

### 3.2.3 Изображение не содержит текста

Поскольку формат .bmp поддерживает загрузку любых изображений, в том числе не содержащих текстовой информации, необходимо чтобы в случае отсутствия текста пользователь был предупреждён, что приложение не поддерживает работу с изображениями, не содержащими текста. Тестовый файл и результат загрузки в приложение изображения, не содержащего текст, представлен на рисунках 47, 48.

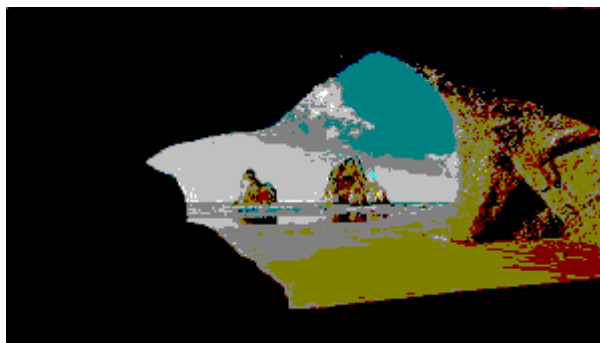


Рисунок 47 – Изображение, не содержащее текстовую информацию

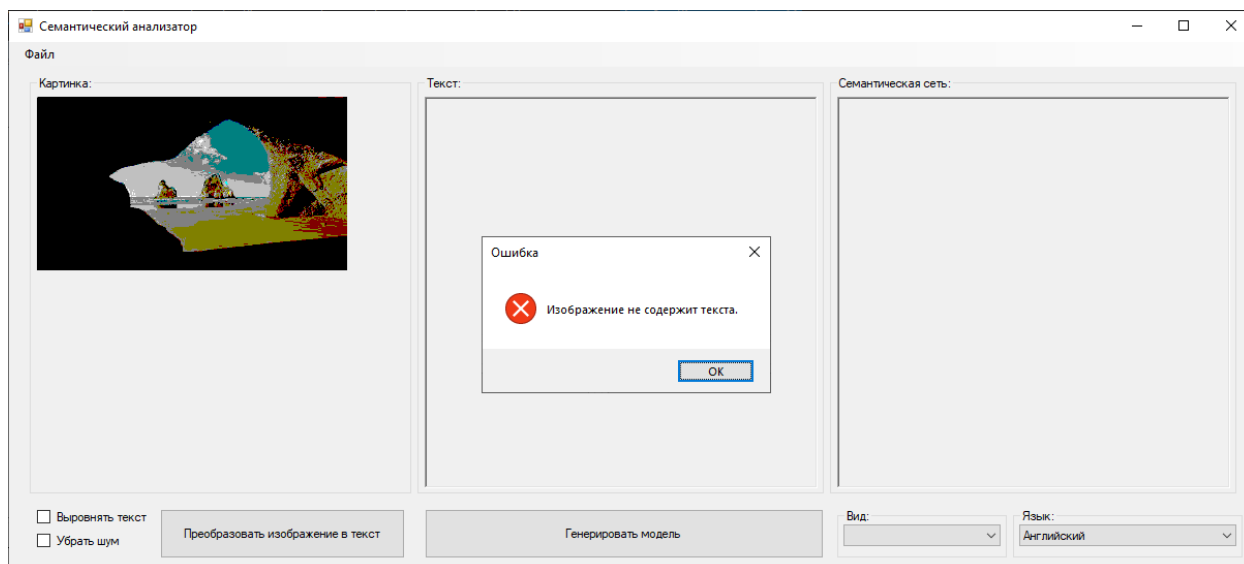


Рисунок 48 – Результат загрузки изображения

Таким образом, приложение не позволяет продолжить работу с графическим файлом, не содержащим текста.

### 3.2.4 Текст на другом языке

Поскольку набор словарей ограничен русским и английским, приложение невозможно использовать для семантического анализа текста на других языках. Изображение, содержащее текст на французском языке и результат, полученный при попытке загрузки изображения в приложение приведены на рисунках 49, 50.

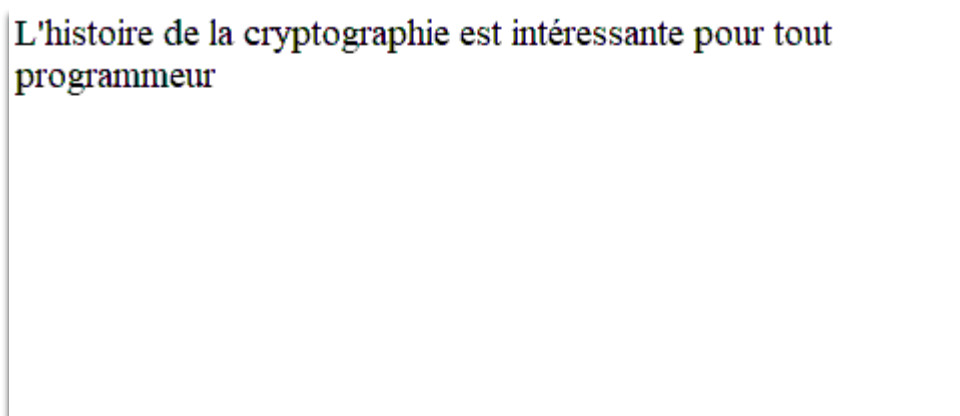


Рисунок 49 – изображение, содержащее текст на французском языке

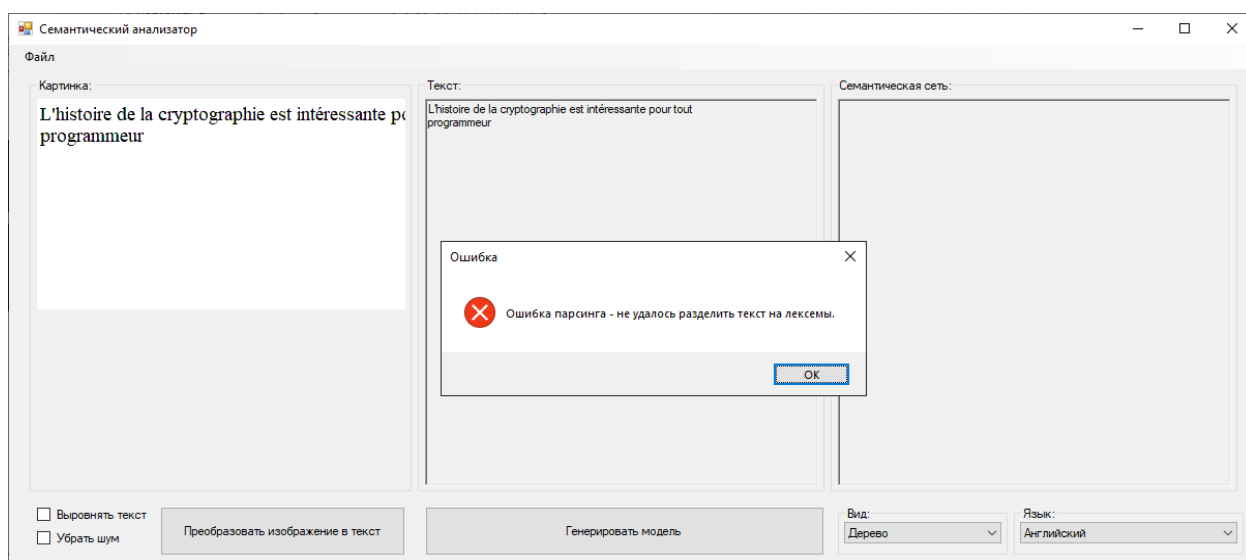


Рисунок 50 – Результат попытки создать семантическую модель

Как видно из результатов тестирования, модуль оптического распознавания символов смог получить текст, заключённый в изображение. Однако поскольку словарь английского языка не содержит французские лексемы, преобразовать текст в семантическую сеть не удалось, о чём пользователя предупреждает соответствующее сообщение.

### 3.3 Выводы

В этом разделе было приведено описание итогового приложения, которое реализует пользовательский интерфейс взаимодействия с системой, описаны тестовые данные и методы, проведено итоговое функциональное тестирование разработанной системы, а также составлено руководство пользователя, где подробно с иллюстрациями был описан алгоритм взаимодействия пользователя с приложением.

## **ЗАКЛЮЧЕНИЕ**

В рамках данной выпускной квалификационной работы было разработано приложение для семантического анализа текстовой информации, представленной в графической форме.

В основу механизма семантического анализа были положены алгоритмы токенизации и морфологического разбора. Приложение обладает простым и интуитивно понятным пользовательским интерфейсом и реализовывает функционал загрузки, проверки, представления и сохранения модели семантического графа.

Разработка осуществлялась с использованием среды Visual Studio 2019 и языка C#.

Таким образом, можно сказать, что приложение соответствует поставленным целям.

Научная ценность заключается в перспективе внедрения алгоритмов семантического анализа в средства определения неоригинального или потенциально опасного контента с целью его фильтрации или удаления. В дальнейшем приложение можно улучшить, добавив поддержку большего числа языков, автоматического исправления ошибок.

В ходе выполнения работы, были получены навыки разработки оконных приложений. Улучшены навыки разработки в среде Visual Studio 2019. Получен дополнительный опыт и расширены знания по языку программирования C#.

## СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Интерфейсы. Определение интерфейсов [Электронный ресурс]. URL: <https://metanit.com/sharp/tutorial/3.9.php>. (дата обращения 18.03.2022).
2. Команда (Command) [Электронный ресурс]. URL: <https://metanit.com/sharp/patterns/3.3.php> (дата обращения 18.03.2022).
3. Evaluation of OCR Algorithms for Images with Different Spatial Resolutions and Noises by Qing Chen – School of Information Technology and Engineering Faculty of Engineering University of Ottawa, 2003.
4. Джеф Просиз - Программирование для Microsoft .NET, – М.: Издательско-торговый дом «Русская Редакция», 2003.
5. Требования к программному обеспечению/ Материал из Википедии – свободной энциклопедии. [Электронный ресурс]. URL: [https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5\\_%D1%80%D0%B0%D1%81%D0%BF%D0%BE%D0%B7%D0%BD%D0%B0%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5\\_%D1%81%D0%B8%D0%BC%D0%B2%D0%BE%D0%BB%D0%BE%D0%B2](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D1%80%D0%B0%D1%81%D0%BF%D0%BE%D0%B7%D0%BD%D0%B0%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D1%81%D0%B8%D0%BC%D0%B2%D0%BE%D0%BB%D0%BE%D0%B2) (дата обращения 12.03.2022).
6. Макконнелл С. «Совершенный код». – М.: Издательство БХВ-Петербург, 2017. – 914 с.
7. Документация по C#: [Электронный ресурс]. URL: <https://docs.microsoft.com/ru-ru/dotnet/csharp/>. (дата обращения 4.03.2022).
8. Руководство по рабочему столу .NET для Windows Forms: [Электронный ресурс]. URL: <https://docs.microsoft.com/ru-ru/dotnet/desktop/winforms/?view=netframeworkdesktop-4.8>. (дата обращения 14.04.2022).
9. Optical Character Recognition Based Retrieval by Asnake Biniam (Author), Aweke Girma (Author) – LAP Lambert Academic Publishing; Illustrated edition, 2014.

## ПРИЛОЖЕНИЕ А

### Листинг программы

#### Файл «DrawingObject.cs»

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Windows.Forms;

namespace GraphicObjects
{
    public class DrawingObject
    {
        public enum MouseEventType
        {
            Move,
            Down,
            Up
        }

        #region Внутренние типы

        /// <summary>
        ///
        /// </summary>
        [Flags]
        internal enum BoundForChangingValues
        {
            None = 0,
            X = 1,
            Y = 1 << 1,
            Width = 1 << 2,
            Height = 1 << 3
        }
    }
}
```

```

}

/// <summary>
/// Класс описывает прямоугольник изменения размера.
/// </summary>
internal class ResizeBoxInternal
{
    /// <summary>
    /// Конструктор.
    /// </summary>
    /// <param name="size"></param>
    public ResizeBoxInternal(Size size)
    {
        var b = Bounds;
        b.Width = size.Width;
        b.Height = size.Height;
        Bounds = b;
    }

    /// <summary>
    ///
    /// </summary>
    public Rectangle Bounds { get; set; }

    /// <summary>
    ///
    /// </summary>
    public BoundForChangingValues BoundForChangingValue { get; set; }

    /// <summary>
    ///
    /// </summary>
    public Cursor Cursor { get; set; }
}

```

```
#endregion
```

```
protected readonly Control _parent;  
protected Rectangle _objectRectangle;  
protected Point _lastMouseLocation;
```

```
protected bool _resizing;  
protected bool _locationChanging;
```

```
protected readonly Pen _dashedPen;  
protected readonly Pen _resizeBoxPen;  
protected readonly Brush _resizeBoxFill;  
protected readonly Brush _objectBrush;
```

```
private ResizeBoxInternal _activeBox;
```

```
private    readonly    List<ResizeBoxInternal>    _resizeBoxes    =    new  
List<ResizeBoxInternal>();
```

```
private ResizeBoxInternal _leftTopBox;  
private ResizeBoxInternal _topBox;  
private ResizeBoxInternal _rightTopBox;  
private ResizeBoxInternal _leftBox;  
private ResizeBoxInternal _rightBox;  
private ResizeBoxInternal _leftDownBox;  
private ResizeBoxInternal _downBox;  
private ResizeBoxInternal _rightDownBox;  
private ResizeBoxInternal _borderBox;
```

```
protected const int BoxSize = 8;  
protected const int ResizeBorderSize = 5;  
protected const int MinWidth = 15;  
protected const int MinHeight = 15;
```

```

public bool Resizing { get => _resizing; }
public bool LocationChanging { get => _locationChanging; }

public DrawingObject(Rectangle rect, Control parent)
{
    _parent = parent;
    _objectRectangle = rect;
    _resizeBoxFill = new SolidBrush(Color.White);
    _objectBrush = new SolidBrush(Color.LightGreen);
    _resizeBoxPen = new Pen(Color.Gray, 1) { DashStyle = DashStyle.Solid };
    _dashedPen = new Pen(Color.Black, 1) { DashStyle = DashStyle.Dash };
    InitResizeBoxes();
}

public void Draw(Graphics g)
{
    ArrangeBoxes();
    DrawObject(g);
    DrawBoxes(g);
}

private void InitResizeBoxes()
{
    var boxSize = new Size(BoxSize, BoxSize);

    _leftTopBox = new ResizeBoxInternal(boxSize)
    {
        BoundForChangingValue = BoundForChangingValues.X |
        BoundForChangingValues.Y
    };
    _resizeBoxes.Add(_leftTopBox);

    _topBox = new ResizeBoxInternal(boxSize)
    {
        BoundForChangingValue = BoundForChangingValues.Y
    };
    _resizeBoxes.Add(_topBox);
}

```

```

};
_resizeBoxes.Add(_topBox);

_rightTopBox = new ResizeBoxInternal(boxSize)
{
    BoundForChangingValue = BoundForChangingValues.Y
BoundForChangingValues.Width
};
_resizeBoxes.Add(_rightTopBox);

_leftBox = new ResizeBoxInternal(boxSize)
{
    BoundForChangingValue = BoundForChangingValues.X
};
_resizeBoxes.Add(_leftBox);

_rightBox = new ResizeBoxInternal(boxSize)
{
    BoundForChangingValue = BoundForChangingValues.Width
};
_resizeBoxes.Add(_rightBox);

_leftDownBox = new ResizeBoxInternal(boxSize)
{
    BoundForChangingValue = BoundForChangingValues.X
BoundForChangingValues.Height
};
_resizeBoxes.Add(_leftDownBox);

_downBox = new ResizeBoxInternal(boxSize)
{
    BoundForChangingValue = BoundForChangingValues.Height
};
_resizeBoxes.Add(_downBox);

```

```

        _rightDownBox = new ResizeBoxInternal(boxSize)
        {
            BoundForChangingValue = BoundForChangingValues.Width |
BoundForChangingValues.Height
        };
        _resizeBoxes.Add(_rightDownBox);

        _borderBox = new ResizeBoxInternal(Size.Empty)
        {
            BoundForChangingValue = BoundForChangingValues.None,
            Cursor = null
        };
        _resizeBoxes.Add(_borderBox);

        //
        // Установка курсоров.
        //

        _leftTopBox.Cursor = _rightDownBox.Cursor = Cursors.SizeNWSE;
        _rightTopBox.Cursor = _leftDownBox.Cursor = Cursors.SizeNESW;
        _leftBox.Cursor = _rightBox.Cursor = Cursors.SizeWE;
        _topBox.Cursor = _downBox.Cursor = Cursors.SizeNS;

        ArrangeBoxes();
    }

    private void ArrangeBoxes()
    {
        _leftTopBox.Bounds = new Rectangle(_objectRectangle.Left - BoxSize / 2,
_objectRectangle.Top - BoxSize / 2,
        BoxSize, BoxSize);

        _topBox.Bounds = new Rectangle((_objectRectangle.Left +
_objectRectangle.Right) / 2 - BoxSize / 2,
        _objectRectangle.Top - BoxSize / 2, BoxSize, BoxSize);
    }

```

```

        _rightTopBox.Bounds = new Rectangle(_objectRectangle.Right - BoxSize / 2,
            _objectRectangle.Top - BoxSize / 2, BoxSize, BoxSize);

        _leftBox.Bounds = new Rectangle(_objectRectangle.Left - BoxSize / 2,
            (_objectRectangle.Top + _objectRectangle.Bottom) / 2 - BoxSize / 2,
            BoxSize, BoxSize);

        _rightBox.Bounds = new Rectangle(_objectRectangle.Right - BoxSize / 2,
            (_objectRectangle.Top + _objectRectangle.Bottom) / 2 - BoxSize / 2,
            BoxSize, BoxSize);

        _leftDownBox.Bounds = new Rectangle(_objectRectangle.Left - BoxSize / 2,
            _objectRectangle.Bottom - BoxSize / 2, BoxSize, BoxSize);

        _downBox.Bounds = new Rectangle((_objectRectangle.Left +
            _objectRectangle.Right) / 2 - BoxSize / 2,
            _objectRectangle.Bottom - BoxSize / 2, BoxSize, BoxSize);

        _rightDownBox.Bounds = new Rectangle(_objectRectangle.Right - BoxSize / 2,
            _objectRectangle.Bottom - BoxSize / 2, BoxSize, BoxSize);
    }

    private void DrawBoxes(Graphics g)
    {
        foreach (var box in _resizeBoxes)
        {
            g.FillRectangle(_resizeBoxFill, box.Bounds);
            g.DrawRectangle(_resizeBoxPen, box.Bounds);
        }
    }

    private void DrawObject(Graphics g)
    {
        g.FillRectangle(_objectBrush, _objectRectangle);
    }

```

```

        g.DrawRectangle(_dashedPen, _objectRectangle);
    }

    public void Update(MouseEventType eventType, MouseEventArgs e)
    {
        ResizeBoxInternal box;

        switch (eventType)
        {
            case MouseEventType.Move:
                if (_resizing)
                {
                    var sizeOffset = new Size(e.Location.X - _lastMouseLocation.X,
e.Location.Y - _lastMouseLocation.Y);
                    ResizeRectangle(_activeBox.BoundForChangingValue, sizeOffset);
                    _lastMouseLocation = e.Location;
                }
                else if (_locationChanging)
                {
                    var locationOffset = new Size(e.Location.X - _lastMouseLocation.X,
e.Location.Y - _lastMouseLocation.Y);
                    _objectRectangle.Location = new Point(_objectRectangle.Location.X +
locationOffset.Width, _objectRectangle.Location.Y + locationOffset.Height);
                    _lastMouseLocation = e.Location;
                }
                else if ((box = HitTestBox(e.Location)) != null)
                {
                    _parent.Cursor = box.Cursor;
                }
                else
                {
                    _parent.Cursor = HitTestBorder(e.Location) ? Cursors.SizeAll :
Cursors.Default;
                }
                break;

```

```

        case MouseEventType.Down:
            box = HitTestBox(e.Location);

            if (box != null)
            {
                _activeBox = box;
                _resizing = true;
                _lastMouseLocation = e.Location;
            }
            else if (HitTestBorder(e.Location))
            {
                _locationChanging = true;
                _lastMouseLocation = e.Location;
            }
            break;
        case MouseEventType.Up:
            _parent.Cursor = Cursors.Default;
            _resizing = false;
            _locationChanging = false;
            break;
        default:
            throw new ArgumentOutOfRangeException(nameof(eventType), eventType,
null);
    }
}

internal ResizeBoxInternal HitTestBox(Point location)
{
    return _resizeBoxes.FirstOrDefault(box => box.Bounds.Contains(location));
}

internal bool HitTestBorder(Point location)
{
    if (_objectRectangle.Top < location.Y && _objectRectangle.Bottom > location.Y)
    {

```

```

        if (Math.Abs(_objectRectangle.Left - location.X) < ResizeBorderSize)
        {
            return true; // курсор на левой границе прямоугольника
        }

        if (Math.Abs(_objectRectangle.Right - location.X) < ResizeBorderSize)
        {
            return true; // курсор на правой границе прямоугольника
        }
    }

    if (_objectRectangle.Left < location.X && _objectRectangle.Right > location.X)
    {
        if (Math.Abs(_objectRectangle.Top - location.Y) < ResizeBorderSize)
        {
            return true; // курсор на верхней границе прямоугольника
        }

        if (Math.Abs(_objectRectangle.Bottom - location.Y) < ResizeBorderSize)
        {
            return true; // курсор на нижней границе прямоугольника
        }
    }

    return false;
}

private void ResizeRectangle(BoundForChangingValues changingValues, Size
offset)
{
    var newSize = _objectRectangle.Size;
    var newLocation = _objectRectangle.Location;

    switch (changingValues)
    {

```

```

        case BoundForChangingValues.X | BoundForChangingValues.Y:
            newLocation = new Point(_objectRectangle.Location.X + offset.Width,
                                    _objectRectangle.Location.Y + offset.Height);
            newSize = new Size(_objectRectangle.Width - offset.Width,
                                _objectRectangle.Height - offset.Height);
            break;
        case BoundForChangingValues.Y:
            newLocation = new Point(_objectRectangle.Location.X,
                                    _objectRectangle.Location.Y + offset.Height);
            newSize = new Size(_objectRectangle.Width, _objectRectangle.Height -
                                offset.Height);
            break;
        case BoundForChangingValues.Y | BoundForChangingValues.Width:
            newLocation = new Point(_objectRectangle.Location.X,
                                    _objectRectangle.Location.Y + offset.Height);
            newSize = new Size(_objectRectangle.Width + offset.Width,
                                _objectRectangle.Height - offset.Height);
            break;
        case BoundForChangingValues.X:
            newSize = new Size(_objectRectangle.Width - offset.Width,
                                _objectRectangle.Height);
            newLocation = new Point(_objectRectangle.Location.X + offset.Width,
                                    _objectRectangle.Location.Y);
            break;
        case BoundForChangingValues.Width:
            newSize = new Size(_objectRectangle.Width + offset.Width,
                                _objectRectangle.Height);
            break;
        case BoundForChangingValues.X | BoundForChangingValues.Height:
            newLocation = new Point(_objectRectangle.Location.X + offset.Width,
                                    _objectRectangle.Location.Y);
            newSize = new Size(_objectRectangle.Width - offset.Width,
                                _objectRectangle.Height + offset.Height);
            break;
        case BoundForChangingValues.Height:

```

```

        newSize = new Size(_objectRectangle.Width, _objectRectangle.Height +
offset.Height);
        break;
        case BoundForChangingValues.Width | BoundForChangingValues.Height:
            newSize = new Size(_objectRectangle.Width + offset.Width,
_objectRectangle.Height + offset.Height);
            break;
        case BoundForChangingValues.None:
            break;
    }

    if (newSize.Width < MinWidth || newSize.Height < MinHeight)
    {
        return;
    }

    _objectRectangle.Location = newLocation;
    _objectRectangle.Size = newSize;
}
}
}

```

### **Файл «MainWindow.cs»**

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using IronOcr;

namespace OCR_Using_IronOCR

```

```

{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            cbLanguage.SelectedIndex = 0;
        }

        private void btnTransform_Click(object sender, EventArgs e)
        {
            btnTransform.Enabled = false;

            var Ocr = new IronTesseract();

            // Если выбран русский язык
            switch (cbLanguage.SelectedIndex)
            {
                case 0:
                    Ocr.Language = OcrLanguage.EnglishBest;
                    break;

                case 1:
                    Ocr.Language = OcrLanguage.RussianBest;
                    break;
            }

            using (var Input = new OcrInput(pbPic.Image))
            {
                // Если изображение неровное
                if (chbAlign.Checked)
                    Input.Deskew();

                // Очистка шума
                if (chbNoise.Checked)

```

```

        Input.DeNoise();
        var Result = Ocr.Read(Input);

        tbText.Text = Result.Text;
    }

    btnTransform.Enabled = true;
}

private void btnClose_Click(object sender, EventArgs e)
{
    Application.Exit();
}

private void btnOpen_Click(object sender, EventArgs e)
{
    if (fd.ShowDialog() == DialogResult.OK)
    {
        pbPic.Image = Image.FromFile(fd.FileName);
        btnTransform.Enabled = true;
    }
}
}
}

```

## ПРИЛОЖЕНИЕ В

### Презентация

# Семантический анализатор текстовой информации

Автор работы: магистрант группы 20ВВм1 Прокуров Д.А.

Научный руководитель: к.т.н., доцент Федюнин Р.Н.

## Цель и объект работы

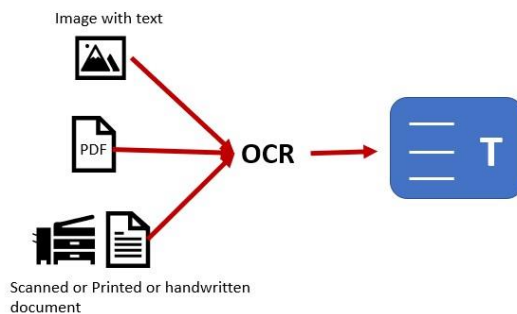
**Цель работы:** разработать приложение для семантического анализа текстовой информации, представленной в графическом виде.

**Объект работы:** десктопное приложение, реализующее следующие возможности:

- считывание графических файлов, содержащих текстовую информацию;
- проведение семантического анализа представленного текста;
- представление семантической модели в виде графа или дерева;
- сохранение семантической модели для последующей обработки.

# Технология OCR

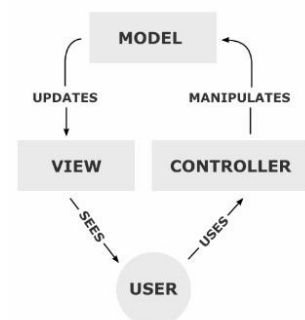
**OCR** — это механический или электронный перевод изображений рукописного, машинописного или печатного текста в текстовые данные, используемые для представления символов в компьютере.



# Паттерн MVC

**Model-View-Controller** — схема разделения данных приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер.

- **Модель** предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.
- **Представление** отвечает за отображение данных модели пользователю, реагируя на изменения модели.
- **Контроллер** интерпретирует действия пользователя, оповещая модель о необходимости изменений.



# Токенизация

На этапе **токенизации** задача программы – выделить в тексте основные структурные единицы – лексемы и распознать их, сопоставив со словарными формами или другими морфологическими образцами

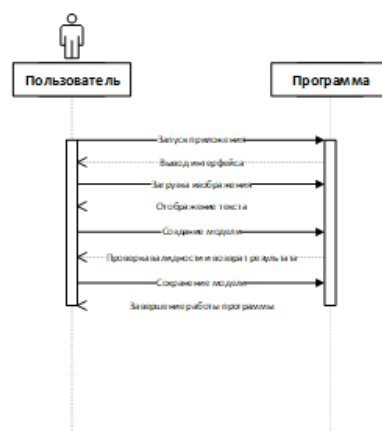
- Результат токенизации – направленный граф, не имеющий циклических путей.



## Морфологический разбор предложений

- Для осуществления морфологического разбора используется морфологический анализатор
- **Морфологический анализатор** — это набор алгоритмов, которые занимается сопоставлением отдельных слов и словоформ в словаре и выяснением грамматических характеристик слов.

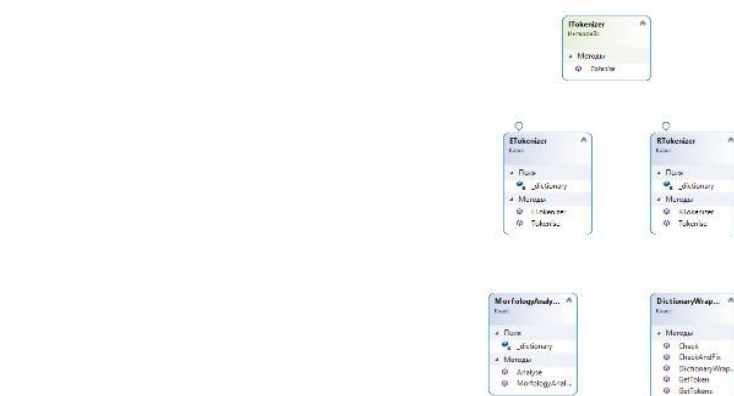
## Диаграмма последовательности



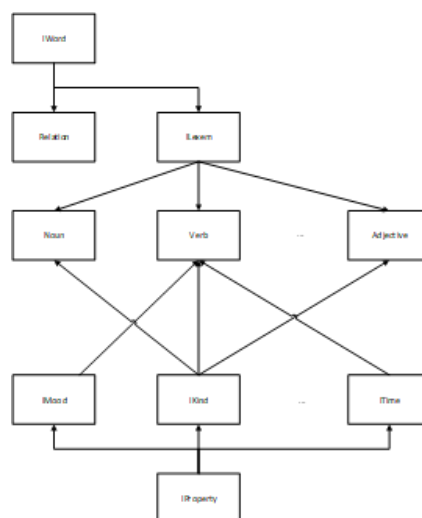
## Стек технологий



## Схема модуля семантического анализа



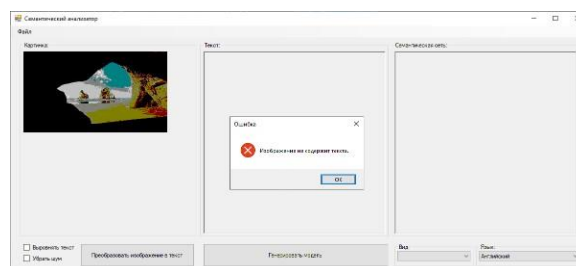
## Схема модели данных



## Схема модуля представления



## Тестирование приложения



## Заключение

В ходе проведённой работы получены следующие результаты:

- Разработано приложение для семантического анализа текстовой информации, представленной в графическом виде;
- Система протестирована на различных входных данных.

В дальнейшем приложение можно улучшить, добавив поддержку большего числа языков, автоматического исправления ошибок.

Спасибо за внимание!