Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Вычислительная техника
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой

_____ _____
подпись инициалы, фамилия
« ____» _____ 20 __ г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 Информатика и вычислительная техника
код и наименование направления

Средство объектно-ориентированного моделирования для поддержки процесса разработки ICONIX
тема

Руководитель _____ ст. преподаватель В.С. Васильев
подпись, дата должность, ученая степень инициалы, фамилия

Выпускник _____ П.Р. Краснова
подпись, дата инициалы, фамилия

Консультант _____ доцент, канд. техн. наук Л.И. Покидышева
подпись, дата должность, ученая степень инициалы, фамилия

Нормоконтролер _____ доцент, канд. техн. наук В.И. Иванов
подпись, дата должность, ученая степень инициалы, фамилия

Красноярск 2019

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Проектирование – это процесс декомпозиции системы. В процессе объектно-ориентированного проектирования разрабатывается ряд диаграмм. Так, например, в рамках Rational Unified Process используется 12 видов диаграмм [1], а в процессе ICONIX 4 вида. Большинство инструментов объектно-ориентированного проектирования поддерживают концепцию визуального проектирования, то есть диаграммы составляются путём манипулирования графическими объектами. Однако, в некоторых случаях, построение диаграммы с помощью текстового ее представления является более удобным, например, не возникает проблем при сохранении в репозитории, упрощается командная разработка диаграмм, текст проще редактировать, существует множество инструментов, специализирующихся на обработке текстов.

Целью работы является создание среды объектно-ориентированного моделирования на базе процесса ICONIX, которая обеспечивает разработку и хранение диаграмм в текстовом виде.

В настоящее время существует ряд инструментов, позволяющих по текстовому описанию UML диаграмм получать их графическое представление. Например, PlantUML [2]. PlantUML – проект с открытым кодом. Это быстрый, удобный в использовании инструмент с простым и понятным синтаксисом. Одним из требований при выполнении работы было использование формата диаграмм, принятого в PlantUML. Для достижения цели в работе решаются **следующие задачи**:

- составить техническое задание на разработку десктопного оконно-браузерного приложения, основываясь на прецедентах;
- выполнить проектирование среды объектно-ориентированного моделирования на базе процесса ICONIX основываясь на техническом задании;
- реализовать систему моделирования придерживаясь разработанных ранее технических решений;

- провести тестирование приложения, опираясь на разработанные в ходе составления технического задания функциональные требования;
- разработать инструкции пользователя и программиста.

1 Анализ задания на выпускную квалификационную работу

Система проектирования должна быть создана на основе процесса ICONIX [3]. Проектирование в соответствии с процессом ICONIX заключается в разработке диаграмм 4-х видов:

- диаграммы использования (use-case);
- диаграммы пригодности (robustness);
- диаграммы последовательности (sequence);
- диаграммы классов (class) [3].

Разработка диаграмм ведется итеративно. Соответствующий процесс проиллюстрирован на рисунке 1.

Рисунок 1 - Этапы процесса ICONIX

Подавляющее большинство средств проектного моделирования ориентированы на процесс RUP [1]. Такие как IBM Rational [4], IBM Rational XDE [5], IBM Rational SoDA [6] и т.д. Инструментов на основе процесса ICONIX, описанного на рисунке 1, не существует.

Use-case диаграммы, которые еще называют диаграммами вариантов использования или диаграммами прецедентов, присутствуют как в процессе RUP, так и в ICONIX. У такой диаграммы есть как графическая часть, наглядно показывающая варианты использования, так и текстовая часть, которая более подробно описывает прецеденты.

Пример графической составляющей диаграммы показан на рисунке 2. Код для диаграммы, изображенной на рисунке 2, взят из справочного руководство по языку PlantUML [2]. При построении use-case диаграммы необходимо также описание ее прецедентов по определенному шаблону, который предусматривает описание реакции программы на те или иные действия со стороны пользователя. В описании прецедентов учитываются как оптимистические сценарии работы программы, так и прагматические. Пример описания прецедентов для use-case диаграммы есть в книге «Объектно-ориентированный анализ и проектирование с примерами приложений» [7]. В системе проектирования описание можно добавить к любой из диаграмм, но только к use-case диаграммам оно обязательно и только для use-case диаграмм оно имеет определенную структуру.

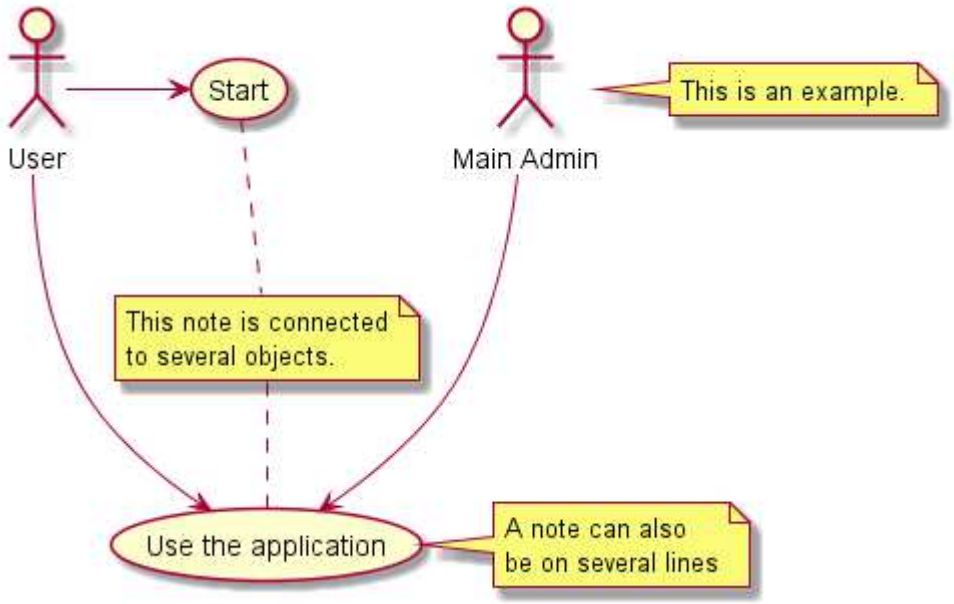

Рисунок 2 – Пример диаграммы вариантов использования

Разрабатываемая в рамках ВКР система проектирования также предусматривает создание отчетности по проекту. То есть пользователь, после составления всех нужных ему UML-диаграмм сможет создать отчет, включающий в себя все диаграммы созданного проекта и описания к ним.

PlantUML простой, быстрый и понятный инструмент для построения диаграмм. В среде разработки (IDE) используется интегрированный транслятор с языка PlantUML, на выходе из которого получается готовая UML диаграмма, которую можно сохранить в графическом формате. Пример PlantUML диаграммы также показан на рисунке 2.

1.1 Разработка технического задания

1.1.1 Интерфейс программы

При запуске система проектирования пуста. Это видно со снимка экрана, показанного на рисунке 3. Для того чтобы начать работу следует либо создать проект, либо открыть существующий.

Рисунок 3 – Начальный экран системы проектирования

При открытом проекте интерфейс программы выглядит так, как показано на рисунке 4. В дереве проекта создаются 4 папки для каждого из видов диаграмм, предусмотренных в процессе ICONIX.

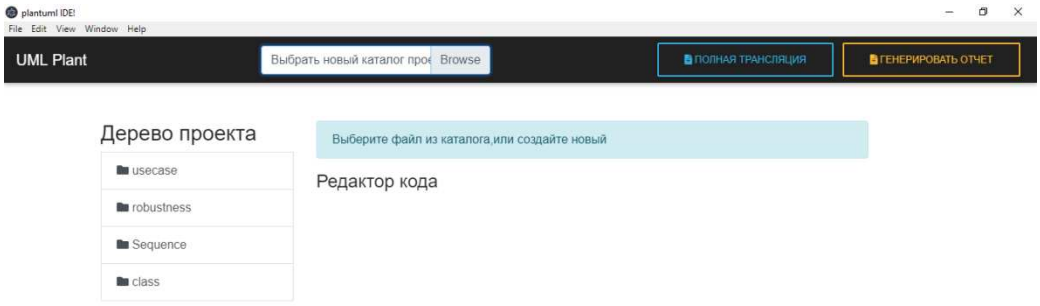

Рисунок 4 – Экран системы проектирования, при пустом проекте

Для того чтобы создать диаграмму следует создать файл в одной из папок проекта. Для этого необходимо выбрать папку проекта, в которой будет создан новый файл, затем ввести имя файла и нажать на кнопку «+». Таким образом файл будет добавлен как показано на рисунке 5.

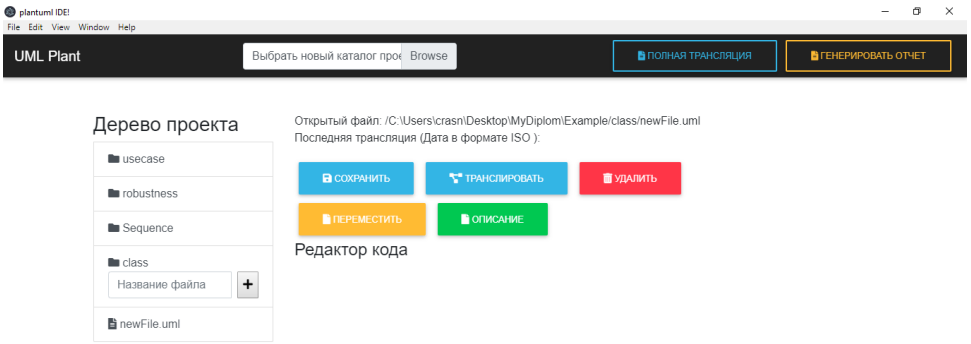

Рисунок 5 – Экран системы проектирования после создания файла проекта

Для того что бы начать работу с файлом пользователю необходимо открыть файл. Если файл открыт в IDE, поле «Редактор кода» становится доступным для заполнения. Также становится доступным весь остальной функционал, связанный с работой с файлом. Далее пользователь пишет код в поле «Редактор кода» и добавляет описание к диаграмме, если этого требует система. После написания кода его необходимо транслировать, нажав на кнопку «Транслировать» или «Полная трансляция», а также сохранить, нажав на кнопку «Сохранить». После проделанных действий, в папке появятся файл построенной диаграммы с расширением .jpg. Результаты показаны на рисунке 6. После того, как пользователь построит все нужные ему диаграммы, ему следует нажать на кнопку «Генерировать отчет» для генерации отчета в форматах .doc и .pdf.

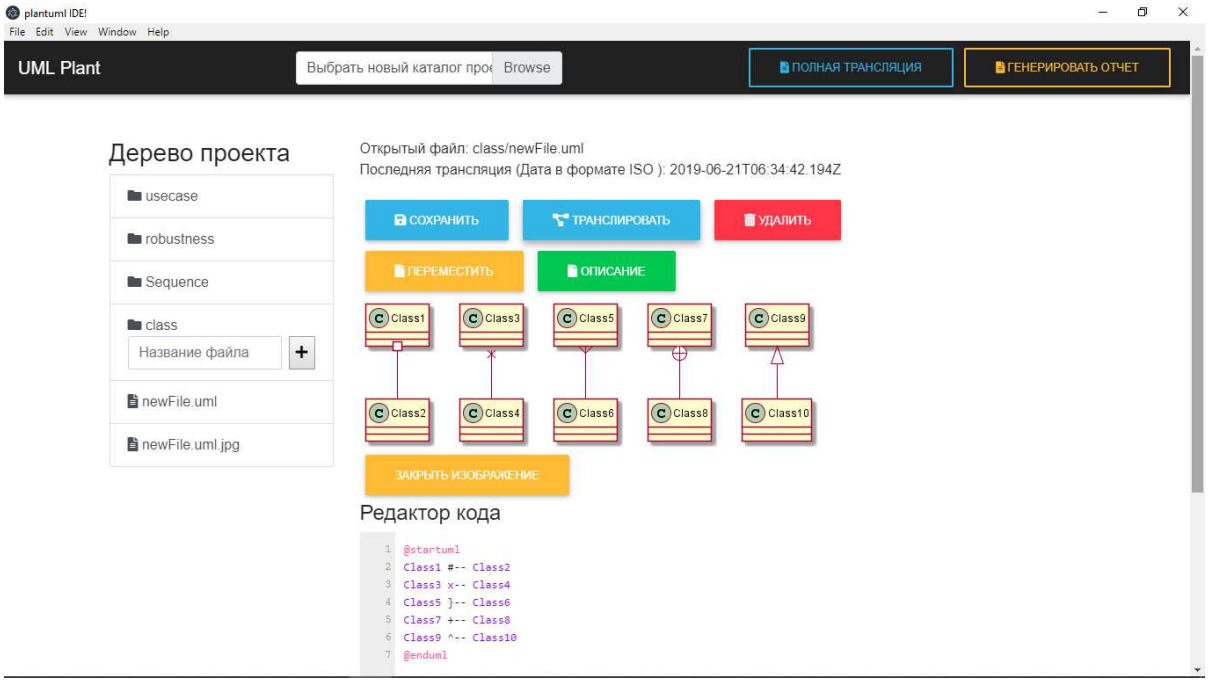

Рисунок 6 – Пример проекта в системе проектирования

Код для построения диаграммы классов, показанной на рисунке 6, взят из справочного руководства по языку PlantUML [2].

Сам проект ВКР будет создан на основе процесса проектирования ICONIX, так как это удобное решение для разработки данного проекта.

1.1.2 Функциональные требования к программе

Функциональные требования выражены в виде диаграмм вариантов использования, как показано на рисунке 7.

Рисунок 7 – Диаграмма вариантов использования системы моделирования

Текстовое описание прецедентов:

Название прецедента: Создать/открыть проект.

Предусловие: Пользователь находится в главном окне программы.

Цель сценария: Создать папку проекта на компьютере и открыть ее в IDE для дальнейшей работы. Если папка проекта уже существует, открыть ее в приложении.

Основной сценарий:

1) пользователь нажимает на поле «Выбрать новый каталог проекта»;

2) пользователь попадает в корневой каталог компьютера;

3) в этом каталоге, или же в одном из подкаталогов пользователь создает папку проекта;

4) после того, как папка проекта создана, пользователь открывает проект в IDE.

Постусловия: В IDE открыт проект.

Название прецедента: Создать файл в проекте.

Предусловие: Пользователь находится в главном окне программы, проект уже существует.

Цель сценария: Создать/открыть файл в проекте.

Основной сценарий:

1) пользователь нажимает на поле «Название файла» и вводит название файла. Расширение файла должно быть .uml;

2) пользователь нажимает на кнопку «+»;

3) файл открывается в IDE, поле «Редактор кода» становятся доступны для ввода данных, а также становится доступным остальной функционал приложения, связанный с работой с файлами;

4) если файл уже существует в системе и его необходимо редактировать, то нужно нажать на название файла с расширением .uml в дереве проекта и он откроется в IDE для редактирования;

5) в графу «Открытый файл» записывается путь до созданного файла и его имя.

Постусловия: Создан и открыт в IDE новый файл.

Условие ввода и действие альтернативных сценариев:

Условие 1. Имя файла не введено, пользователь нажимает на кнопку «Создать файл», после чего система показывает предупреждение: «Пустое имя файла».

Условие 2. Если в названии файла не указано расширение .uml:

1) поле «Редактор кода» не доступно для ввода данных;

2) файл появляется в файлах проекта, но открыть его в IDE невозможно;

3) при попытке открыть такой файл система сообщает: «Поддерживается только открытие файлов txt и uml».

Название прецедента: Написать код и описание диаграммы.

Предусловие: Пользователь находится в главном окне программы, проект уже существует, файл проекта существует, открыт и имеет расширение .uml.

Цель сценария: Написать код на основе которого будет построена диаграмма и добавить описание диаграммы, если это необходимо.

Основной сценарий:

1) пользователь вводит код для построения диаграммы в поле «Редактор кода»;

2) пользователь нажимает на кнопку «Описание» и вводит текстовое описание диаграммы (если это требуется), если это use-case диаграмма, то в качестве описание пользователь заполняет форму «Прецеденты».

Постусловия: Написан код для построения диаграммы и описание диаграммы (описание – необязательно для всех видов диаграмм, кроме use-case).

Условие ввода и действие альтернативных сценариев:

Условие 1. При попытке ввода чего-либо кроме текста, текстовое поле никак не изменится, так как поля «Редактор кода», «Описание» и форма «Прецеденты» текстовые, соответственно, вводить туда можно только текст.

Условие 2. Если заполнены не все поля формы «Прецеденты», система выдаст сообщение «Не все поля заполнены».

Название прецедента: Транслировать.

Предусловие: Пользователь выполнил прецедент «Написать код и описание диаграммы», файл с кодом открыт в главном окне приложения.

Цель сценария: Обработка кода, создание на основе предложенного кода диаграммы в графическом виде.

Основной сценарий:

1) пользователь нажимает на кнопку «Транслировать»;

2) файл .uml сохраняется;

3) код передается на обработку транслятору;

4) обновляется время последней трансляции;

5) создается новый файл проекта с расширением .jpg;

6) система проектирования выдает сообщение: «Транслировано»;

7) файл проекта с расширением .jpg открывается в IDE.

Постусловия: Обновляется время последней трансляции, создаётся и открывается в IDE новый файл проекта с расширением .jpg. В новом файле проекта хранится графическое представление диаграммы.

Условие ввода и действие альтернативных сценариев:

Условие 1. Если код некорректный:

1) код программы не может быть обработан транслятором;

2) транслятор присылает ошибку обработки;

3) файл с картинкой создается, но вместо диаграммы в нем сообщение об ошибке.

Название прецедента: Создать отчет.

Предусловие: Пользователь выполнил прецеденты «Создать файл», «Транслировать». Все необходимые диаграммы и описания к ним построены.

Цель сценария: Составление отчета с диаграммами и их описаниями в формате .docx и .pdf.

Основной сценарий:

1) пользователь нажимает на кнопку «Отчет»;

2) система выдает сообщение «Отчет базируется на последней скомпилированной диаграмме»;

3) считываются данные из файла с графическим представлением диаграммы и поля «ОПИСАНИЕ» или же прецеденты для use-case диаграммы;

4) создается новый файл в формате .docx;

5) создается новый файл в формате .pdf;

6) данные из файла с графическим представлением диаграммы и поля «ОПИСАНИЕ» или прецеденты записываются в новые файлы отчета в формате .docx и .pdf, при том, сначала записывается диаграмма, а потом ее описание.

Постусловия: Созданы новые файлы с информацией из файла с графическим представлением диаграммы, поля «ОПИСАНИЕ» и формы «Прецеденты».

Условие ввода и действие альтернативных сценариев:

Условие 1. Если произошла ошибка при составлении диаграммы в графическом виде, то отчеты будут построены с картинкой, на которой отображена ошибка с описанием к диаграмме.

Условие 2. Если файл не найден, система выдаст сообщение: «Файл не найден. Возможно вы не делали трансляцию диаграммы».

Название прецедента: Удаление файла проекта.

Предусловие: Пользователь выполнил прецедент «Создать файл».

Цель сценария: Удалить файл.

Основной сценарий:

1) пользователь выбирает файл, который необходимо удалить из дерева проекта в IDE;

2) нужный файл открывается в системе;

3) пользователь нажимает на кнопку «Удалить»;

4) IDE выдает сообщение «Вы действительно хотите удалить файл имяФайла.uml вместе с диаграммой?»;

5) пользователь нажимает «ok» и IDE удаляет файл из проекта. Система выдает сообщение «Файл удален».

Постусловия: Ненужный файл удален.

Условие ввода и действие альтернативных сценариев:

Условие 1. В папке проекта существует файл формата, который система не может открыть.

1) пользователь попытается открыть файл для дальнейшего его удаления, но система выдаст сообщение «Поддерживается только открытие файлов .jpg и .uml»;

2) пользователь должен будет удалить файл неподдерживаемого формата вне IDE.

Условие 2. IDE выдает сообщение «Вы действительно хотите удалить файл имяФайла.uml вместе с диаграммой?» и пользователь нажимает «Отмена».

1) сообщение исчезает;

2) файл остается открытым для редактирования.

Название прецедента: Полная трансляция.

Предусловие: Пользователь выполнил прецедент «Написать код и описание диаграммы» для всех диаграмм.

Цель сценария: Обработка кода для всех uml-файлов проекта, создание на основе предложенного кода диаграмм в графическом виде.

Основной сценарий:

1) пользователь нажимает на кнопку «Полная трансляция»;

2) код передается на обработку модулю NODEPLANTUML;

3) обновляется время последней трансляции;

4) создаются новые файлы проекта с расширением .jpg для каждой диаграммы;

5) система проектирования выдает сообщение: «Трнаслировано»;

6) в главном окне программы отображаются все картинки с диаграммами проекта.

Постусловия: Обновляется время последней трансляции, создаются и отображаются на экране новые файлы проекта с расширением .jpg, транслированные файлы сохраняются в файловой системе.

Условие ввода и действие альтернативных сценариев:

Условие 1. Некорректный код.

1) код программы некорректен и не может быть обработан транслятором;

2) транслятор присылает ошибку обработки;

3) файл с картинкой создается, картинка содержит информацию об ошибке, но она существует только в файловой системе и не отображается в IDE.

1.1.3 Нефункциональные требования

Оконно-браузерное десктопное приложение, разработанное для операционных систем Linux, Windows, Mac.

Система должна обладать удобным, интуитивно понятным, не напрягающим глаз интерфейсом.

Также требуется предусмотреть наличие **java runtime environment** [10] для работы с транслятором.

1.2 Выводы по главе

После выполнения анализа задания на ВКР, были сделаны выводы:

- среды проектирования основанной на процессе ICONIX не существует. Это значит, что IDE разработанная в рамках ВКР будет единственной средой проектирования на основе процесса ICONIX;
- сформировано техническое задание, в ходе создания которого были сформулированы функциональные требования, в виде диаграммы использования и прецедентов к ней, и нефункциональные требования.

2 Проектирование

Для того, чтобы показать взаимодействия компонентов среды моделирования между собой необходимо определить архитектуру системы.

На рисунке 8 приведена архитектура системы. На нем указаны все главные компоненты системы:

- транслятор – программа, которая транслирует код с PlantUML языка в изображение диаграмм;
- файловая система – часть операционной системы, которая обслуживает работу чтение/записи на накопительные устройства;
- графический интерфейс – часть системы, которая занимается отрисовкой пользовательского интерфейса;
- главный процесс – процесс, занимающийся общением с транслятором и файловой системой через программу-браузер. Он содержит основную логику приложения.

Главный процесс приложения отправляет текущий открытий исходный код в транслятор. Транслятор обрабатывает код и отдает результат файловой системе. При открытии изображения, главный процесс electron делает запрос отобразить картинку графическому интерфейсу, а далее графический интерфейс отрисовывает нужное изображение.

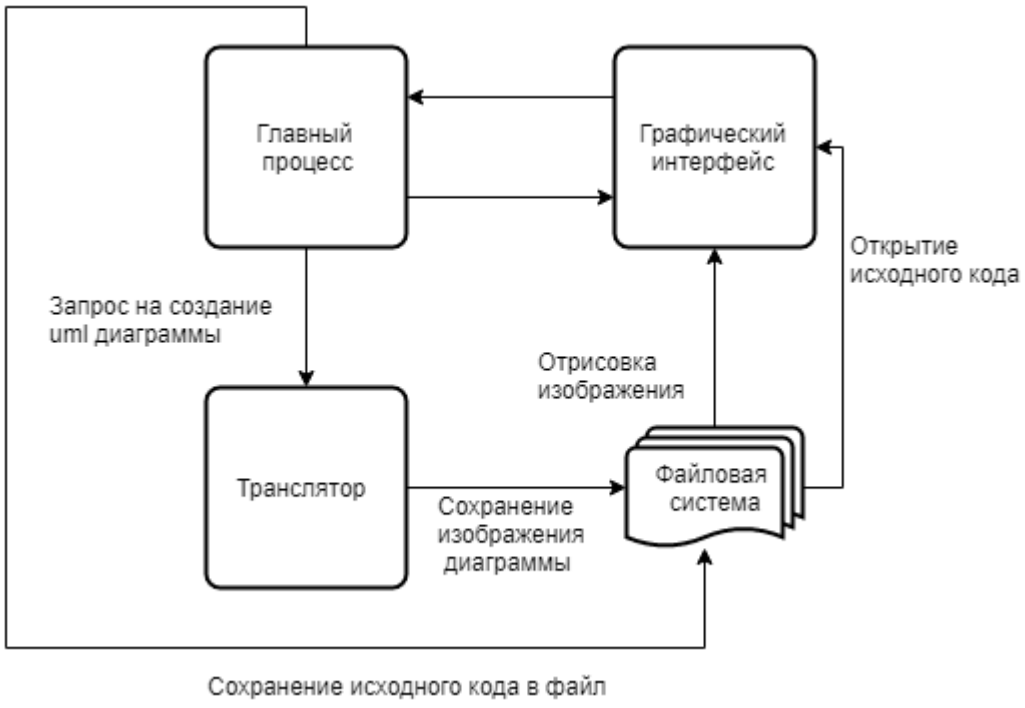

Рисунок 8 – Архитектура системы моделирования

2.1 Динамическая модель системы

Большая часть логики системы проектирования связана с трансляцией и составлением отчета.

В разделе приведены диаграммы робастности и последовательности для наиболее значимых прецедентов, на примере которых будет более подробно рассмотрено взаимодействие компонентов системы.

2.1.1 Диаграммы робастности

На рисунке 9 изображена диаграмма робастности для варианта использования «Транслировать». Она точнее показывает поведение системы, взаимодействие ее компонентов при трансляции кода, чем диаграмма вариантов использования. В ходе построения диаграммы была проверена логика системы и отредактирован прецедент «Транслировать». Был выявлен метод для

построения последующих диаграмм и написания кода. Это метод: translation. Также было решено добавить форму для отображения картинок.

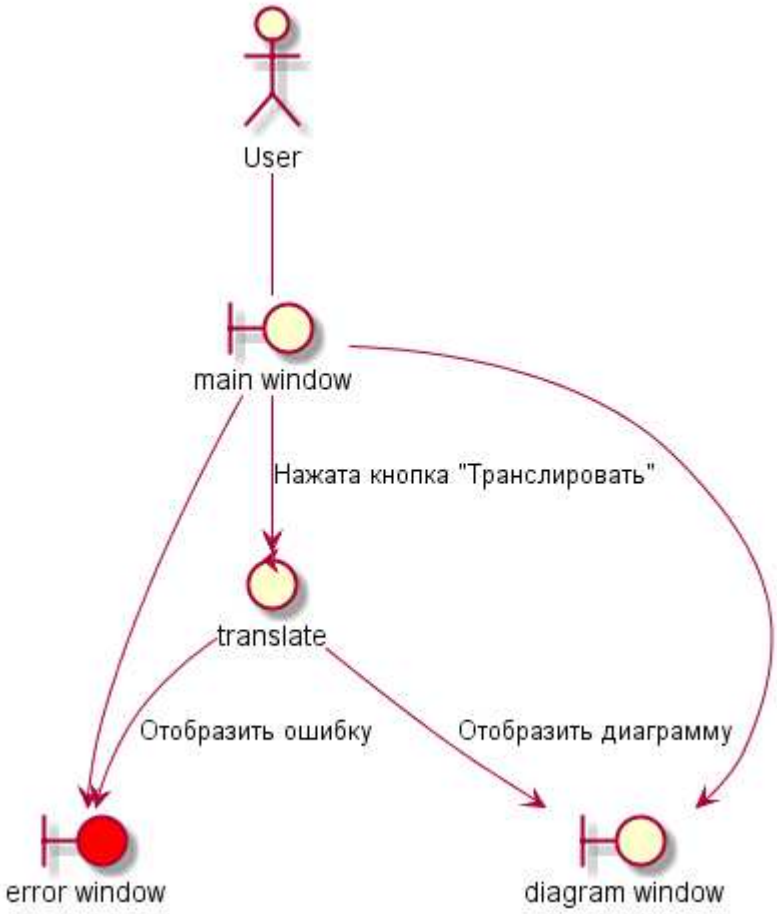

Рисунок 9 – Диаграмма робастности «Транслировать»

На рисунке 10 изображена диаграмма робастности на основе прецедента «Создать отчет». В ходе ее разработки был выделен метод: generateReport. А также необходимость создавать окна-предупреждения для оповещения пользователя о работе системы.

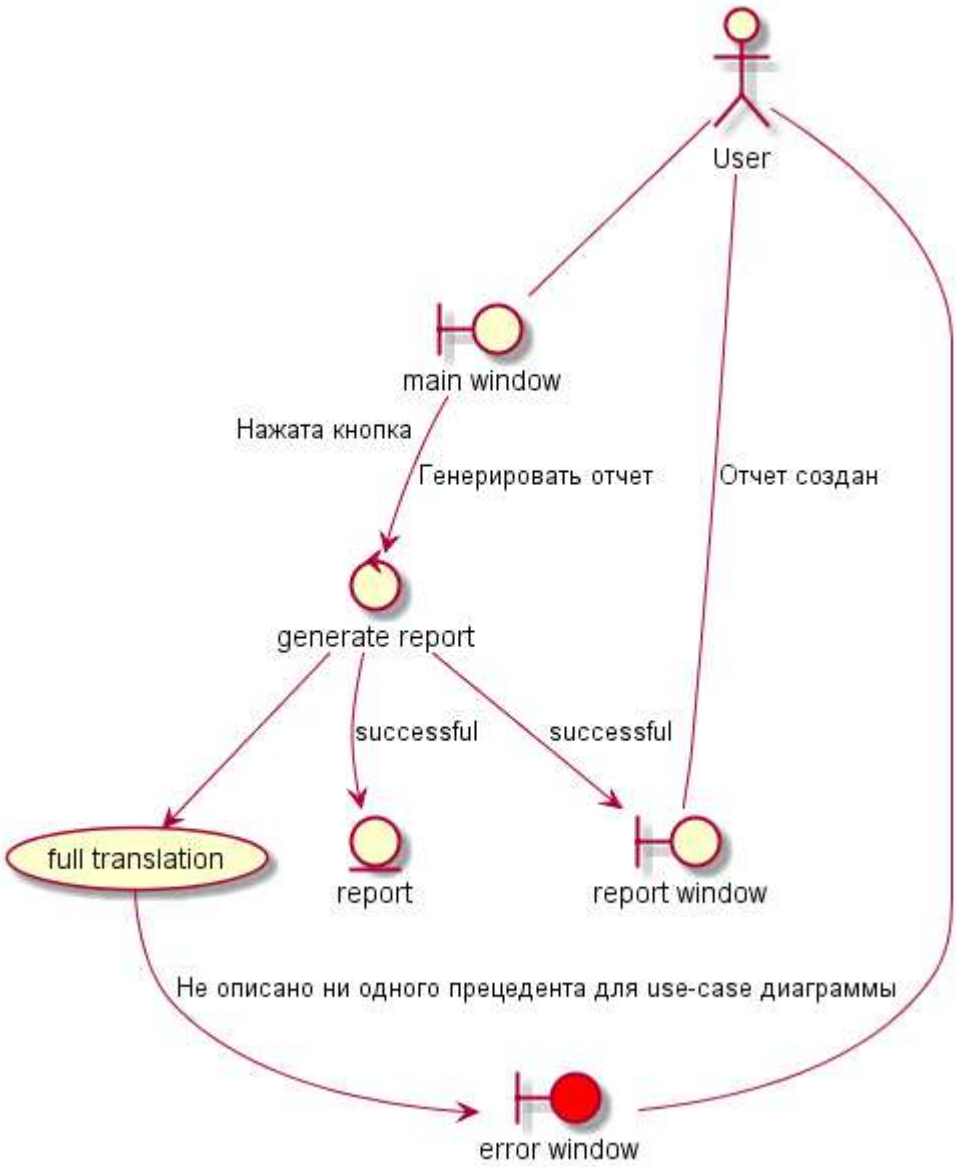

Рисунок 10 – Диаграмма робастности «Создать отчет»

Из рассмотренных диаграмм можно сделать вывод, что диаграммы робастности более подробно показывают поведение системы и взаимодействие ее компонентов, чем диаграмма вариантов использования. В ходе построения диаграммы была проверена логика приложения и отредактирована диаграмма вариантов использования. Так, например, сделав выводы по диаграмме робастности «Транслировать», было решено добавить прецедент «Полная трансляция». Также был выявлен будущий модуль mainWindow. Он присутствует во всех диаграммах выше, так как из-за специфики проекта

экраны не меняются. Диаграммы робастности приведены только для наиболее значимых прецедентов.

2.1.2 Диаграмма последовательности

Следующий этап проектирования – это создание диаграмм последовательности. Диаграммы последовательности разработаны для 2 прецедентов: «Создать отчет» и «Транслировать».

На рисунке 11 изображена диаграмма последовательности для прецедента «Создать отчет».

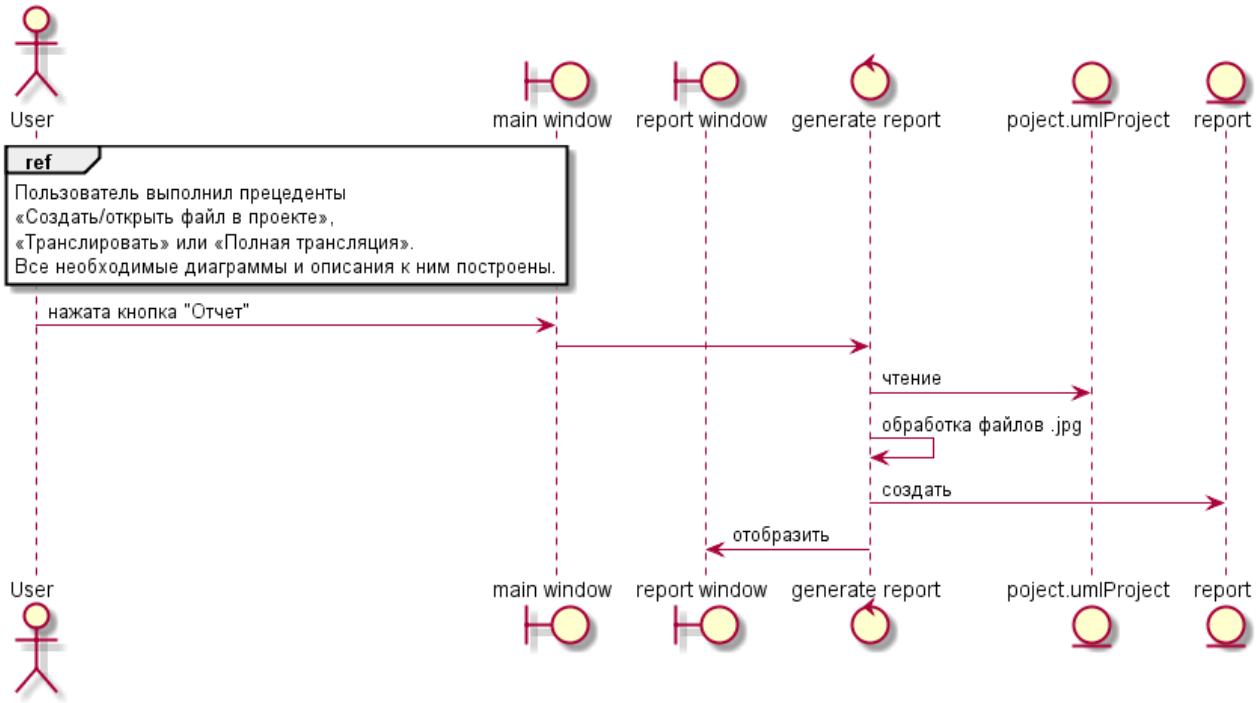

Рисунок 11 – Диаграмма последовательности «Создать отчет»

Исходя из данной диаграммы, было принято решение создать файл проекта с расширением. umlProject. Файл проекта – это файл, который в JSON формате хранит пары ключ-значение, где ключ – строка содержащая путь до файла, а значение – описание к диаграмме.

На рисунке 12 изображена диаграмма последовательности для прецедента

«Транслировать».

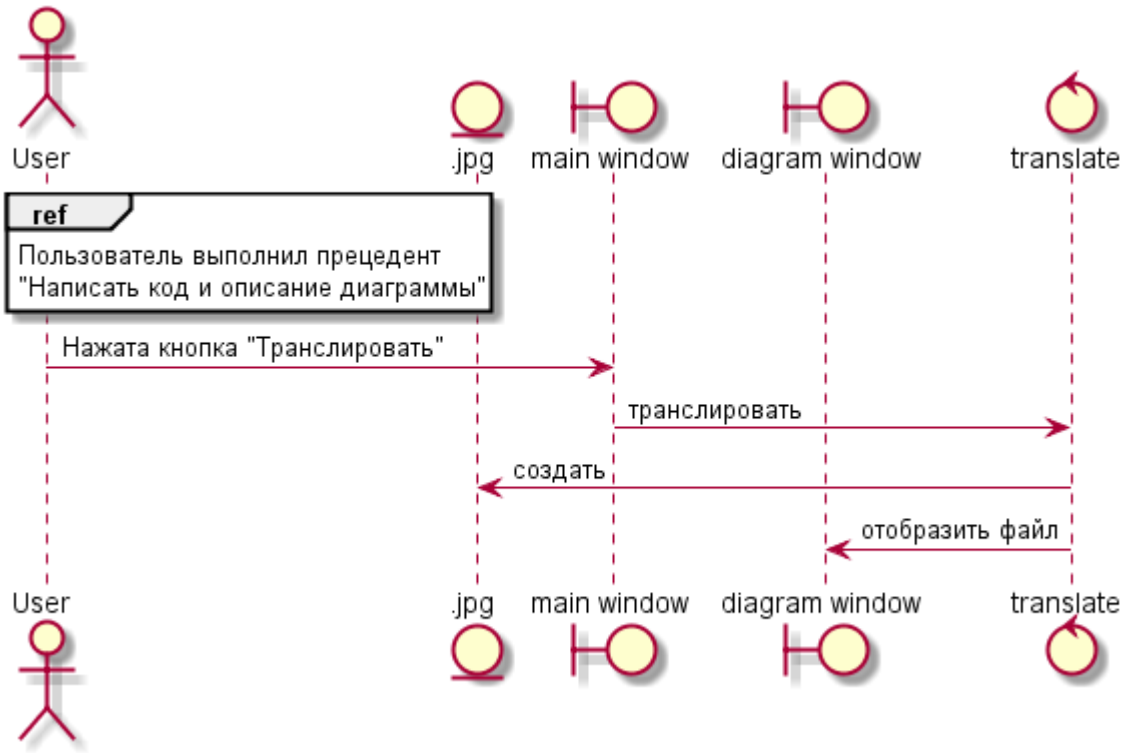

Рисунок 12 – Диаграмма последовательности «Транслировать»

Диаграмма последовательности «Транслировать» показывает процесс сохранения и трансляции кода.

Данный тип диаграмм подробнее, чем диаграммы предыдущих этапов разработки. И отлично иллюстрирует обмен данными между компонентами программы. После разработки диаграмм пригодности была отредактирована диаграмма робастности «Трансляция», в которой была нарушена последовательность действий.

2.2 Статическая модель системы

2.2.1 Модули системы

Система состоит из нескольких модулей, каждый из которых отвечает за определенное функциональное требование. На диаграмме развертывания

системы моделирования, изображенной на рисунке 13 приведены эти модули и показаны связи между ними.

Модули системы:

- главное окно – модуль, который позволяет работать с пользовательским интерфейсом;
- генерация отчета – модуль, отвечающий за генерацию отчета;
- управление файлами в проекте – модуль для работы с файлами;
- транслятор – модуль, отвечающий за трансляцию кода;
- управление проектами – модуль, работающий с проектами;
- работа со стилями кода – модуль, созданный для подсветки кода синтаксиса;
- работа с .doc и .pdf файлами – работает с документами формата docx и pdf.

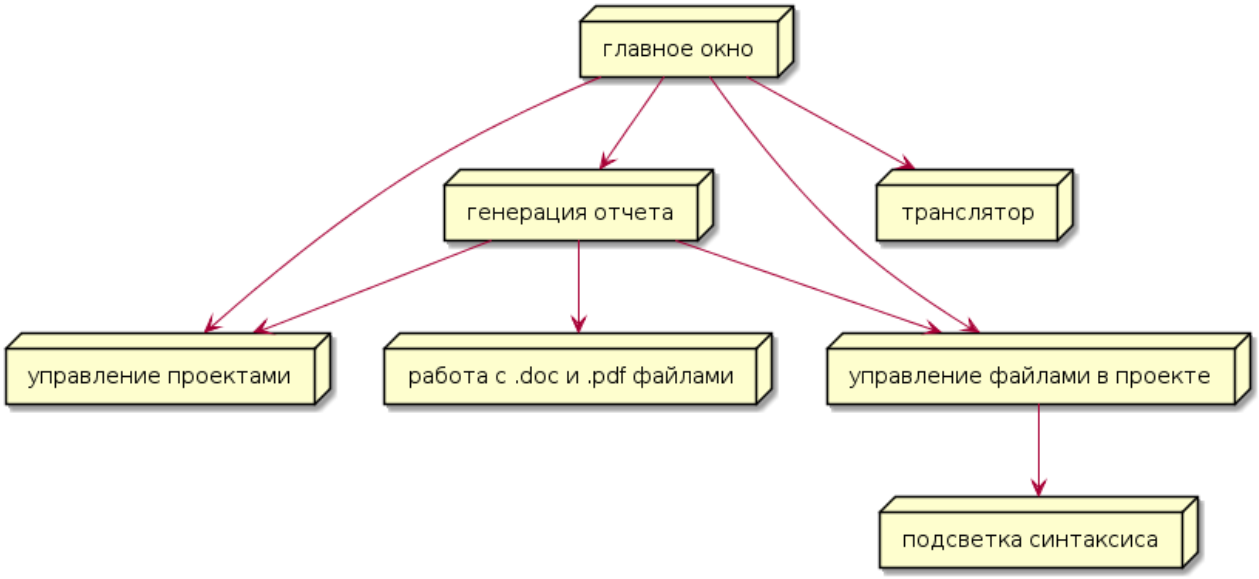

Рисунок 13 – Диаграмма развертывания системы моделирования

2.2.2 Диаграммы взаимодействия модулей системы

В процессе проектирования было решено взять несколько готовых модулей и модифицировать их по мере необходимости. Остальные же модули

решено было писать вручную и на рисунке 14 изображена диаграмма развертывания для наглядного отображения взаимосвязей модулей, написанных вручную между собой. А также были определены компоненты и методы модулей.

«Главное окно» – это модуль, который работает с пользовательским интерфейсом, обеспечивая функционал IDE. Часть обязанностей модуля «главное окно» делегировано модулям «транслятор», «управление файлами проекта» и «генерация отчета». Это показано на рисунке 14 и этот вид отношений называется агрегацией [20].

Для того, чтобы сработал метод модуля «генерация отчета», ему необходимо будет создать и сохранить файлы, а затем сохранить изменения в проекте. Поэтому модуль «генерация отчета» ассоциативно [20] связан с «управление файлами проекта» и «управление проектами». Модуль главное окно тоже ассоциативно связан с «управление проектами», что и показано на рисунке 14.

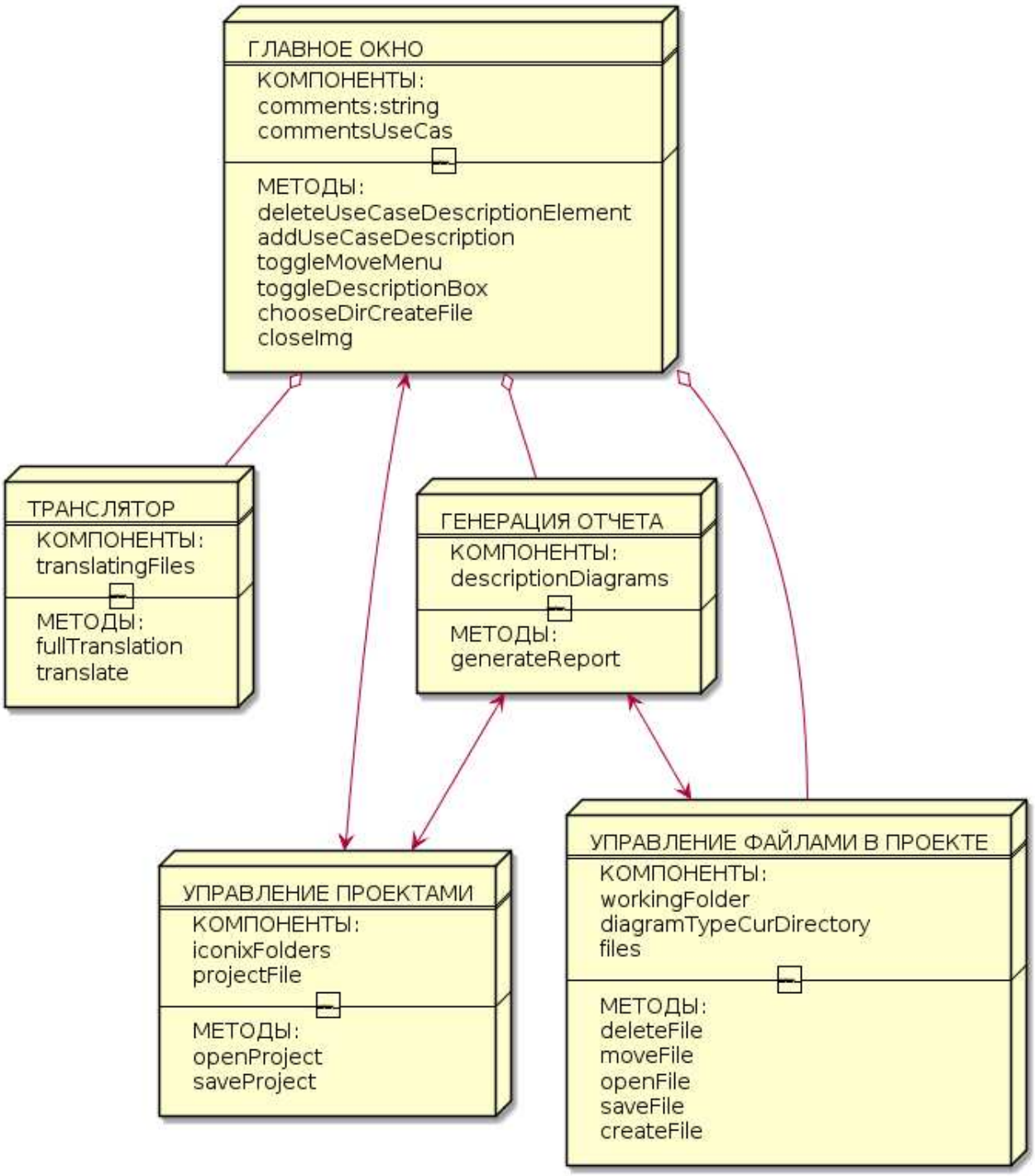

Рисунок 14 – Диаграмма развертывания для модулей системы

Каждый из рукописных модулей для реализации своего функционала связан с интегрированными модулями, как показано на рисунке 13.

Модуль «управление файлами проекта» связан с модулем «Подсветка синтаксиса». Для того что бы подсвечивать код, который вводит пользователь в IDE.

Модуль «генерация отчета» был включен в систему для создания отчета и связан с модулем «работа с .doc и .pdf файлами» для работы с .docx и .pdf файлами. Что также проиллюстрировано на рисунке 13.

2.3 Выводы по главе

По завершению второй главы были сделаны выводы:

- разработана динамическая модель системы;
- разработана статическая модель системы;
- произведена декомпозиция логики IDE;
- определены обязанности и установлены связи для каждого функционального модуля.

3 Реализация и документация

3.1 Реализация

3.1.1 Используемые инструменты

Система проектирования сделана в виде оконно-браузерного десктопного приложения. Для реализации этой идеи было решено использовать фреймворк Electron.

Electron – это фреймворк для разработки настольных приложений с использованием HTML, CSS и JavaScript. В основе Electron лежат проекты Chromium [21] и Node.js [9], объединённые в единую среду, обеспечивающую работу приложений. Это даёт возможность применять веб-технологии при разработке настольных программ.

Chromium используется для отрисовки графического интерфейса. А полный набор модулей Node.js позволяет получить доступ к операционной системе.

Electron – это проект, который использован при создании множества популярных приложений. Таких как мессенджеры Skype и Discord, редакторы для кода Visual Studio Code и Atom и множество других приложений, сведения о которых можно найти на официальном сайте Electron [22].

Весь функционал для элементов интерфейса реализован с помощью фреймворка Vue.js [23].

3.1.2 Интерфейс

Графический интерфейс приложения создан с помощью браузера Chromium.

В браузере Chromium с помощью языка разметки HTML отрисовываются формы, кнопки, таблицы и другие элементы разметки страницы. Также построена иерархическая структура элементов.

На формальном языке описания внешнего вида документа CSS прописаны стили элементов интерфейса, т.е. осуществляется графическое оформление приложения.

Весь функционал для элементов интерфейса реализован с помощью фреймворка Vue.js [23]. Так как Cromium так же использует движок V8, как и Node.js [24], у приложения появляется доступ к операционной системе. За счет этого реализован функционал для таких элементов интерфейса, как: кнопка «Выбрать новый каталог проекта», кнопка «Сохранить», «Транслировать» и так далее.

Также с помощью модуля CodeFlask реализована подсветка интерфейса. Как показано на рисунке 15 на примере диаграммы классов созданной в разработанной в рамках ВКР IDE.

В модуле CodeFlask прописываются зарезервированные слова PlantUML. При открытии файла появляется тестовое поле для ввода кода. Это html элемент <textarea>. Элемент <textarea> преобразуется в <code> элемент.

Функция модуля CodeFlask «просматривает» элемент <code> на предмет наличия зарезервированных слов, обозначенных как регулярные выражения. Если такие слова существуют в тексте, то обработчики модуля CodeFlask придают им определенный стиль. На рисунке 15 показан пример диаграммы классов с подсветкой синтаксиса.

Редактор кода

```
1  class generateReport {
2  + descriptionDiagrams: {}
3  + generateReport ()
4  }
5  class "image size" as isi {
6  }
7   generateReport --> fs
8   generateReport --> path
9   generateReport --> docx
10  generateReport --> word2pdf
11  generateReport --> isi
```

Рисунок 15 – Пример подсветки кода диаграммы

3.1.3 Функционал

Система состоит из нескольких модулей. Часть модулей написана вручную, а остальные модули позаимствованы из разных источников, указанных ниже, и интегрированы в систему.

К написанным вручную модулям относятся:

- mainWindow – модуль, который позволяет работать с пользовательским интерфейсом;
- generateReport – модуль, отвечающий за генерацию отчета;
- fileSystemWrapper – модуль для работы с файлами;
- transSender – модуль, отвечающий за трансляцию кода;
- projectManager – модуль, работающий с проектом.

К интегрированным модулям относятся:

- CodeFlask – модуль, созданный для подсветки кода диаграммы [11];
- fs – модуль Node.js предназначенный для работы с файлами [12];
- path – модуль Node.js предназначенный для работы с путями к файлам и каталогам [13];
- fsExtra – модуль-дополнение к Node.js. Этот модуль используется при реализации перемещения файла из одной папки проекта в другую [14];

- docx – работает с документами формата docx [15];
- image-sizer – модуль Node.js, используемый для получения информации о размерах картинки [16];
- word2pdf – модуль преобразующий документ формата docx в pdf [17];
- child_process – модуль Node.js. Этот модуль нужен чтобы выполнять в командной строке команды для передачи кода диаграммы модулю NODEPLANTUML [18];
- NODEPLANTUML – встроенный транслятор кода. Используется для транслирования кода UML диаграммы в картинку [19].

Все функциональные требования, заявленные в диаграмме прецедентов выполнены. Функционал для системы проектирования полностью написан на языке JavaScript. Главный файл функционала системы это mainWindow. В нем прописан функционал для всех элементов интерфейса, находящихся в главном окне программы. Если для какого-либо элемента функции в mainWindow не предусмотрено, в mainWindow осуществляется вызов данной функции из другого модуля.

Вся работа с проектами и файлами в проектах описывается в модуле FileSystemWrapper и projectManajer. Для работы с файловой системы используются модули Node.js: fs, fsExtra, path.

Рассмотрим подробнее модули, разработанные в рамках ВКР.

Модуль mainWindow работает с пользовательским интерфейсом, в нем реализуется основной функционал системы.

Описание наиболее важных методов mainWindow:

- deleteUseCaseDescriptionElement – метод удаляющий из списка прецедентов один прецедент;
- addUseCaseDescription – добавляет в список прецедент.

Модуль mainWindow связан с модулем generateReport, который собирает все скомпилированные файлы воедино и генерирует отчет.

Метод модуля generateReport – generateReport – метод который вставляет в отчет описания из descriptionDiagrams объекта по каждой из диаграмм, а также .jpg файлы.

Модуль ProjectManager работает с проектами.

Включает в себя методы:

- openProject, который ищет файл в выбранном каталоге с названием project.umlProject, метод загружает файл, если файл существует, или создает новый и загружает его пустым объектом, затем идет загрузка файлов из каталогов, указанных в массиве iconixFolders;
- saveProject, который сохраняет в файл project.umlProject описания всех диаграмм в формате JSON.

Модуль transSender создан для отправки кода транслятору, и получения от него ответа. Его методы – это:

- translate – метод который транслирует текущий открытый файл и автоматически открывает изображение полученной диаграммы;
- fullTranslation – трансляция всех файлов в проекте.

Модуль FileSystemWrapper – модуль для управления файлами через средства ОС.

Содержит поля:

- workingFolder – путь до рабочей директории;
- openedFile – путь до открытого файла;
- diagramTypeCurDirectory – название каталога открытого файла;
- files – список файлов директории.
- Содержит методы:
- deleteFile – метод удаляющий текущий открытый файл проекта вместе с его диаграммой .jpg;
- moveFile – метод переносящий текущий открытий файл в выбранную директорию;
- openFile – метод который, при нажатии на файл в дереве проекта, открывает его в IDE;

- saveFile – сохранение файла;
- createFile – метод создания файла, который вызывается при нажатии на «+» в дереве проекта.

3.2 Документация

3.2.1 Инструкции пользователя

Инструкции пользователя описываются с помощью use-case диаграммы, изображенной на рисунке 7 и прецедентов к ней.

Для начала работы с IDE необходимо:

1) выбрать готовую сборку по ссылке [26];
2) открыть каталог с готовой сборкой;
3) запустить основной файл приложения.

3.2.2 Инструкции программиста

Для запуска версии разработки необходимо:

1) установить Node.js;
2) скачать компонент JDK по ссылке [25];
3) в оболочке Node.js выполнить команду: *npm install*;
4) для запуска версии отладки использовать команду: *npx electron-forge start*.

3.2.3 Тестирование

Для проверки работоспособности среды проектирования было проведено ручное тестирование приложения под операционными системами Windows 10 и Mac OS Mojave. Весь функционал разработанного приложения корректно работает под обеими операционными системами.

На рисунке 16 представлено открытое главное окно IDE с изображением use-case диаграммы в ОС Windows 10.

На рисунке 17 представлено открытое главное окно IDE с простым примером кода диаграммы классов в Mac OS Mojave.

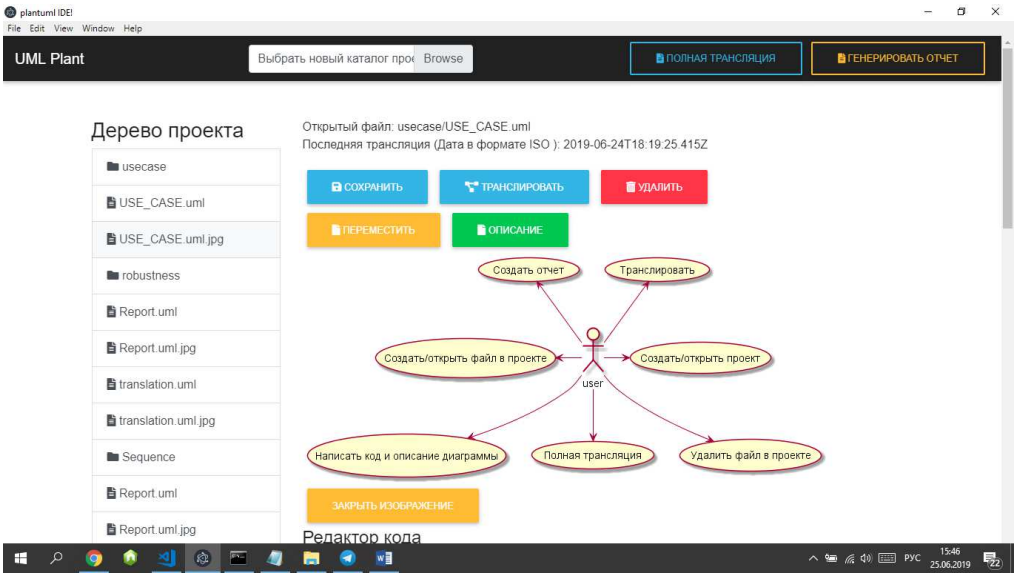

Рисунок 16 – Пример работы IDE в ОС Windows 10

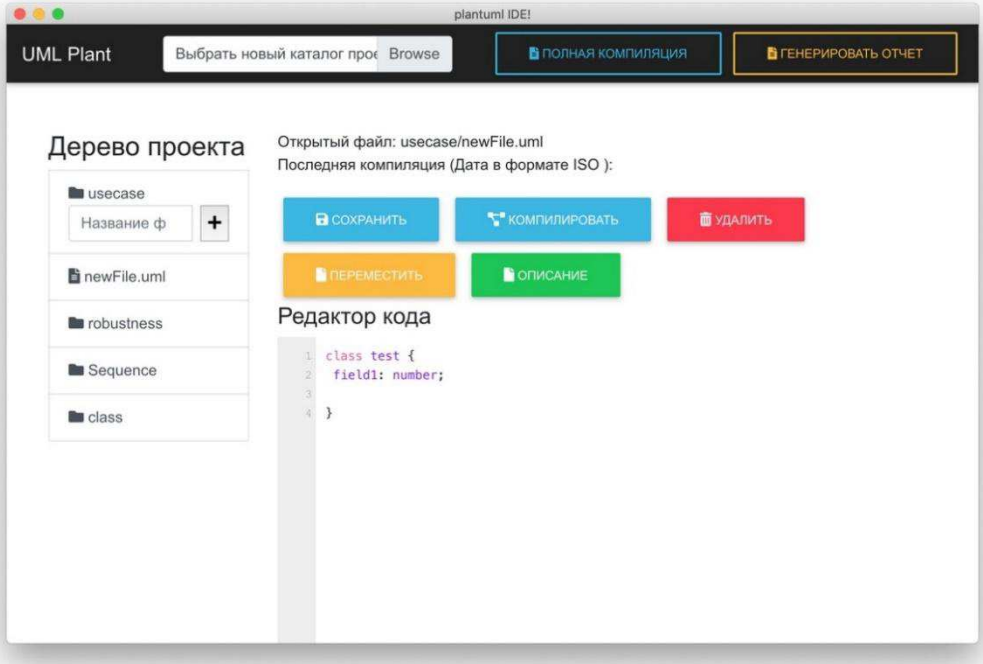

Рисунок 17 – Пример работы IDE в Mac OS Mojave

3.3 Выводы по главе

По завершению реализации проекта были сделаны выводы:

- выбраны инструменты для реализации среды моделирования;
- реализована в виде десктопного оконно-браузерного приложения среда объектно-ориентированного моделирования на базе процесса ICONIX;
- составлены инструкции пользователя и инструкции программиста;
- проведено тестирование системы моделирования.

ЗАКЛЮЧЕНИЕ

В результате проделанной работы спроектирована, реализована и протестирована среда объектно-ориентированного моделирования на базе процесса ICONIX, обеспечивающую разработку и хранение диаграмм в тестовом и графическом виде, а также создание отчетности по проекту.

Также были разработаны инструкции пользователя и программиста. Приложение было упаковано в архив с минимальными изменениями в исходном коде для упрощения его использования. Существуют сборки для трех операционных систем: Windows, Mac, Linux. Сборки доступны для скачивания с git-репозитория [26].

Исходный код приложения доступен для скачивания с git-репозитория [26].

СПИСОК СОКРАЩЕНИЙ

ОС – операционная система

CSS – Cascading Style Sheets

HTML – HyperText Markup Language

IDE – Integrated development environment

RUP – Rational Unified Process

UML – Unified Modeling Language

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Kroll, P. The Rational Unified Process Made Easy: tutorial / P. Kroll, Ph. Kruchten; under the gen. edit. G. Booch. – Booch, Jacobson, Rambaugh, 2006. – 253с.

2. Построение диаграмм UML с использованием PlantUML [Электронный ресурс]: Справочное руководство по языку PlantUML (Version 1.2019.6). – Режим доступа: http://pdf.plantuml.net/PlantUML_Language_Reference_Guide_ru.pdf

3. Процесс разработки программного обеспечения ICONIX [Электронный ресурс]: – Режим доступа: https://pro-prof.com/archives/4126

4. Rational для новичков // IBM Developer [Электронный ресурс]: – Режим доступа: https://www.ibm.com/developerworks/ru/rational/newto/index.html

5. Средства разработки IBM Rational XDE [Электронный ресурс]: – Режим доступа: https://www.bytemag.ru/articles/detail.php?ID=6686

6. IBM Rational SoDA [Электронный ресурс]: – Режим доступа: http://www.interface.ru/home.asp?artId=483

7. Объектно-ориентированный анализ и проектирование с примерами приложений / Г. Буч [G. Booch] [и др.]. – Москва: Вильямс. 2008. – 718с.

8. Node-plantuml [Electronic resource]: – Access mode: https://www.npmjs.com/package/node-plantuml.

9. About Node.js // Nodejs [Electronic resource]: – Access mode: https://nodejs.org/en/about/

10. Java Runtime Environment [Электронный ресурс]: – Режим доступа: http://java-runtime.ru/

11. CodeFlask [Электронный ресурс]: – Режим доступа: https://github.com/kazzkiq/CodeFlask

12. File System [Electronic resource]: Node.js v12.4.0 Documentation // Node.js. – Access mode: https://nodejs.org/api/fs.html#fs_file_system

13. File paths [Electronic resource]: Node.js v12.4.0 Documentation // Node.js. – Access mode: https://nodejs.org/api/fs.html#fs_file_paths

14. Node.js: fs-extra [Electronic resource]: – Access mode: http://adilapapaya.com/docs/fs-extra/

15. docx [Electronic resource]: – Access mode: https://www.npmjs.com/package/docx

16. image-sizer [Electronic resource]: – Access mode: https://github.com/image-size/image-size

17. word2pdf [Electronic resource]: – Access mode: https://www.npmjs.com/package/word2pdf

18. Child Process [Electronic resource]: Node.js v12.4.0 Documentation // Node.js. – Access mode: https://nodejs.org/api/child_process.html#child_process_child_process

19. node-plantuml [Electronic resource]: – Access mode: https://www.npmjs.com/package/node-plantuml

20. Диаграммы классов UML [Электронный ресурс]: – Режим доступа: https://pro-prof.com/archives/3212

21. The Chromium Projects [Electronic resource]: – Access mode: https://www.chromium.org/

22. Electron [Электронный ресурс]: – Режим доступа: https://electronjs.org/

23. Vue.js [Electronic resource]: – Access mode: https://vuejs.org/

24. Node.js + Chromium = AppJS: один из перспективных вариантов второго шага эволюции вебразработчика [Электронный ресурс]: – Режим доступа: https://habr.com/ru/post/153013/

25. Java SE Development Kit 8 Downloads [Электронный ресурс]: – Режим доступа: https://www.oracle.com/technetwork/java/javase/downloads/2133151

26. My-Diploma - репозиторий проекта [Электронный ресурс]: PlantUML IDE – Режим доступа: https://github.com/PolinaKrasnova/My-Diploma.git

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Вычислительная техника
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ Мельниченко О.В.
подпись инициалы, фамилия
«02» 07 20 19 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 Информатика и вычислительная техника
код и наименование направления

Средство объектно-ориентированного моделирования для поддержки процесса разработки ICONIX
тема

Руководитель	Васильев 29.06.19	ст. преподаватель	В.С. Васильев
	подпись, дата	должность, ученая степень	инициалы, фамилия
Выпускник	28.06.2019		П.Р. Краснова
	подпись, дата		инициалы, фамилия
Консультант	28.06.2019	доцент, канд. техн. наук	Л.И. Покидышева
	подпись, дата	должность, ученая степень	инициалы, фамилия
Нормоконтролер	01.07.19	доцент, канд. техн. наук	В.И. Иванов
	подпись, дата	должность, ученая степень	инициалы, фамилия

Красноярск 2019