Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
_____О.В. Непомнящий
"___"_____2023 г.

БАКАЛАВРСКАЯ РАБОТА

090301 Информатика и вычислительная техника

Редактор кода для экспериментального языка программирования O2M

Руководитель _____ _____ доцент, канд. техн. наук Д.А. Швец
подпись *дата* *должность, ученая степень*

Выпускник _____ _____ А.Д. Белоусов
подпись *дата*

Нормоконтролёр _____ _____ Д.А. Швец
подпись *дата*

Красноярск 2023

РЕФЕРАТ

Выпускная квалификационная работа (ВКР) на тему «Редактор кода для экспериментального языка программирования O2M» содержит 106 страниц текстового документа, 59 иллюстраций, 6 использованных источников, презентацию из 12 слайдов.

РЕДАКТОР КОДА, ПРОЦЕСС ICONIX, БИБЛИОТЕКА WXWIDGETS, РАБОТА С ФАЙЛАМИ, ПОДСВЕТКА СИНТАКСИСА, СБОРКА ПРОЕКТА.

Цель выпускной квалификационной работы заключается в разработке редактора кода для экспериментального языка программирования O2M, поддерживающего уникальные возможности данного языка.

Для достижения поставленной цели необходимо решить следующие задачи:

– исследовать особенности существующих редакторов кода;

– разработать программную архитектуру редактора кода;

– выбрать инструменты разработки и реализовать редактор кода на основе разработанной архитектуры;

– произвести тестирование редактора кода на готовых примерах программ на языке O2M.

В результате работы был разработан редактор кода для экспериментального языка программирования O2M, поддерживающий уникальные возможности данного языка.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современной IT-индустрии широко применяются экспериментальные языки программирования, предназначенные для исследования новых языковых механизмов, поддерживающих различные концепции и парадигмы программирования. В отличие от обычных языков программирования, предназначенных для разработки программного обеспечения, экспериментальные языки характеризуются постоянным изменением синтаксиса, что препятствует интеграции подобных языков в универсальные интегрированные среды разработки программ (IDE). Таким образом, часто возникает необходимость в разработке специализированной IDE, предназначенной для конкретного экспериментального языка программирования.

Экспериментальный язык программирования O2M является расширением языка Оберон-2, включающим уникальные механизмы инструментальной поддержки процедурно-параметрического программирования – обобщения и обобщающие процедуры. Данные абстракции позволяют непосредственно использовать множественный полиморфизм, частным случаем которого является объектно-ориентированный полиморфизм, широко применяющийся в современном программировании.

Цель выпускной квалификационной работы заключается в разработке редактора кода для экспериментального языка программирования O2M, поддерживающего уникальные возможности данного языка.

Для достижения поставленной цели необходимо решить следующие задачи:

– исследовать особенности существующих редакторов кода;

– разработать программную архитектуру редактора кода;

– выбрать инструменты разработки и реализовать редактор кода на основе разработанной архитектуры;

– произвести тестирование редактора кода на готовых примерах программ на языке O2M.

По результатам выполненной работы был сделан доклад на 1 международной конференции.

1 Обзор аналогов и описание требований к редактору кода

1.1 Обзор существующих редакторов кода

1.1.1 Code::Blocks

Code::Blocks — это интегрированная среда разработки с открытым исходным кодом, предназначенная для разработки программного обеспечения на языке *C++*, *C* и *Fortran* [1].

Особенности *Code::Blocks* включают в себя возможность создания собственных плагинов, интеграцию с отладчиком *GDB*, возможность быстрой разработки приложений с помощью плагина *wxSmith*. Пример рабочего окна приложения представлен на рисунке 1.1.

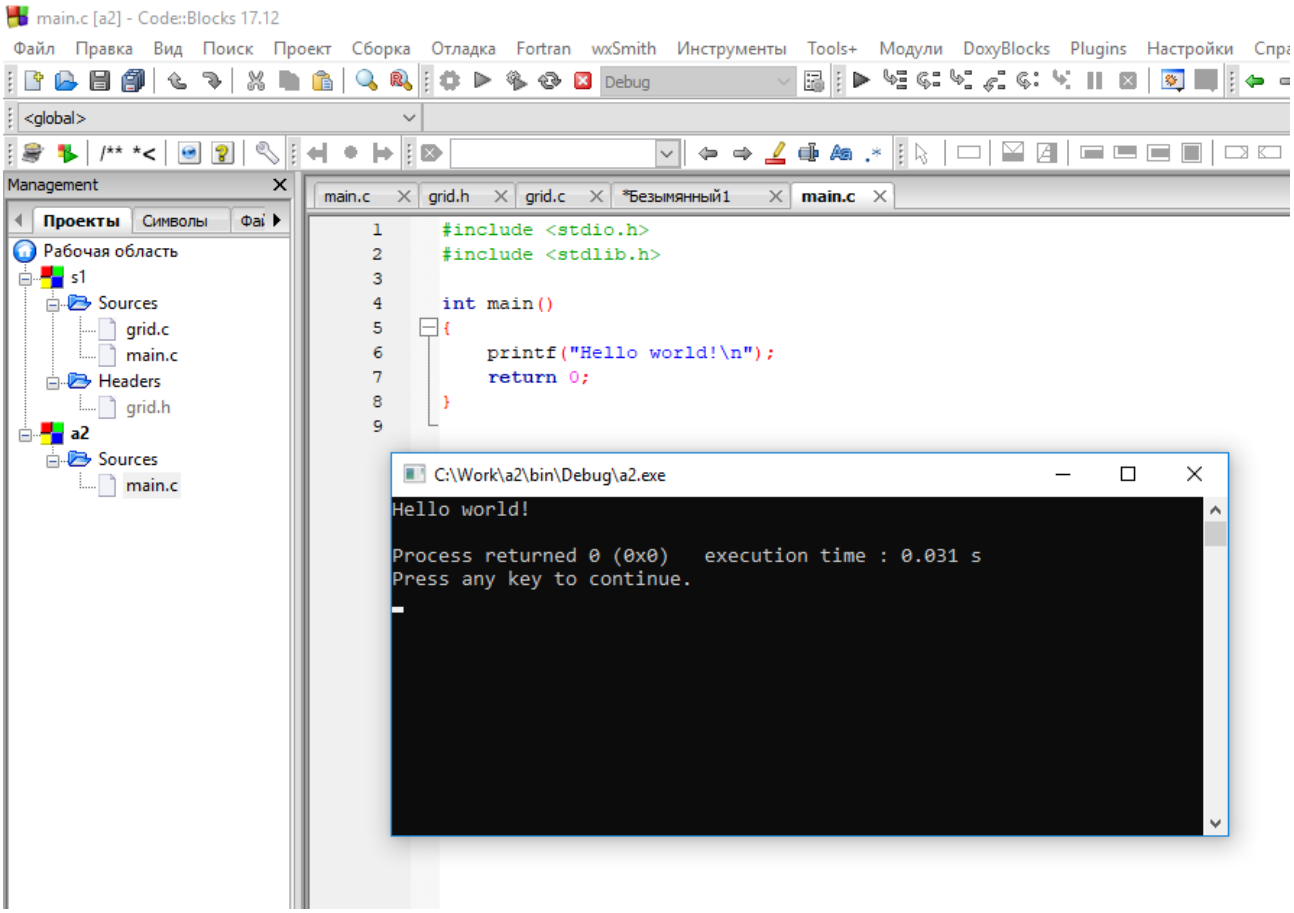

Рисунок 1.1 — Пример рабочего окна *Code::Blocks*

1.1.2 CodeLite

CodeLite — это кроссплатформенная интегрированная среда разработки с открытым исходным кодом, специализирующаяся на языках программирования *C*, *C++*, *Rust*, *Python*, *PHP* и *JavaScript* (в основном для бэкэнд-разработчиков, использующих *Node.js*) [2].

Основные возможности, которые может предоставить *CodeLite*:

– интеграция с системами контроля версий *Subversion* и *Git*;

– интеграция с *UnitTest++*;

– интеграция с отладчиком *GDB*.

Пример рабочего окна приложения представлен на рисунке 1.2.

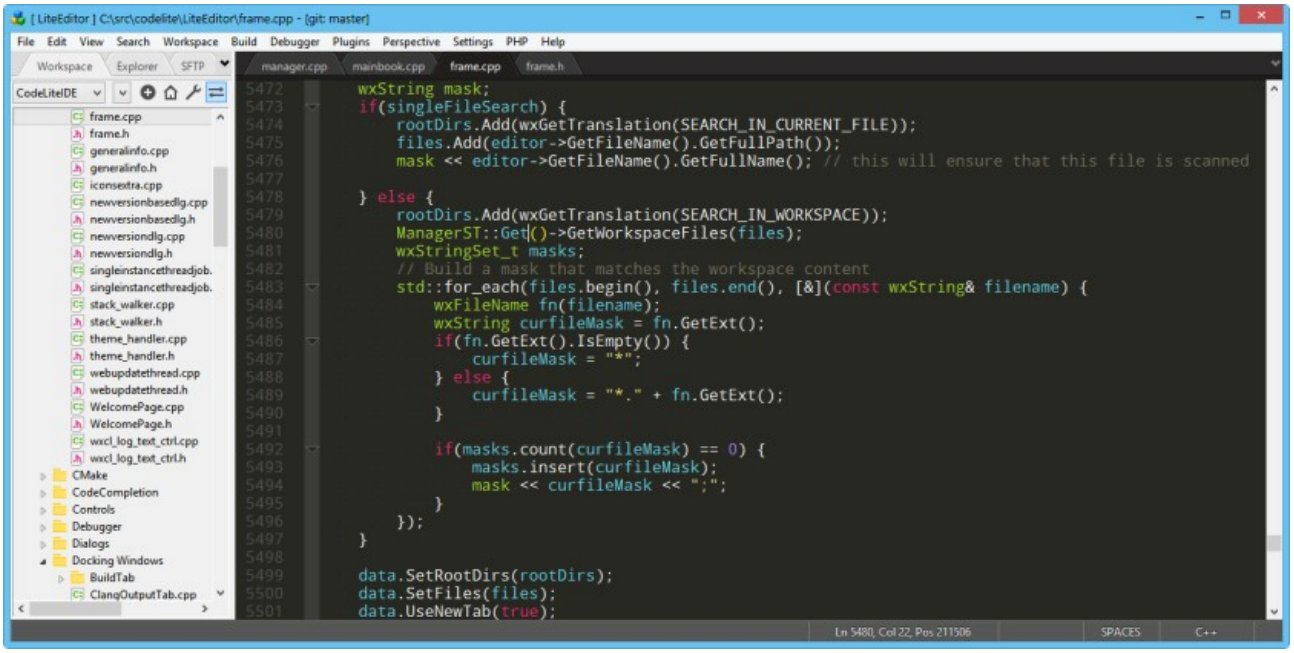

Рисунок 1.2 — Пример рабочего окна *CodeLite*

1.1.3 KDevelop

KDevelop — это интегрированная среда разработки программного обеспечения, разработанная под *KDE Umbrella*. *KDevelop* обеспечивает поддержку широкого спектра языков (таких как *C/C++*, *Python*, *PHP*, *Ruby* и т. д.) с помощью расширяемой среды подключаемых модулей [3].

Основные возможности:

– отсутствует встроенный текстовый редактор;

– графическая оболочка для *GNU Compiler Collection* и *GNU Debugger;*

– встроенная поддержка *Doxygen*

– открытие документации к языку прямо в редакторе.

Пример рабочего окна приложения представлен на рисунке 1.3.

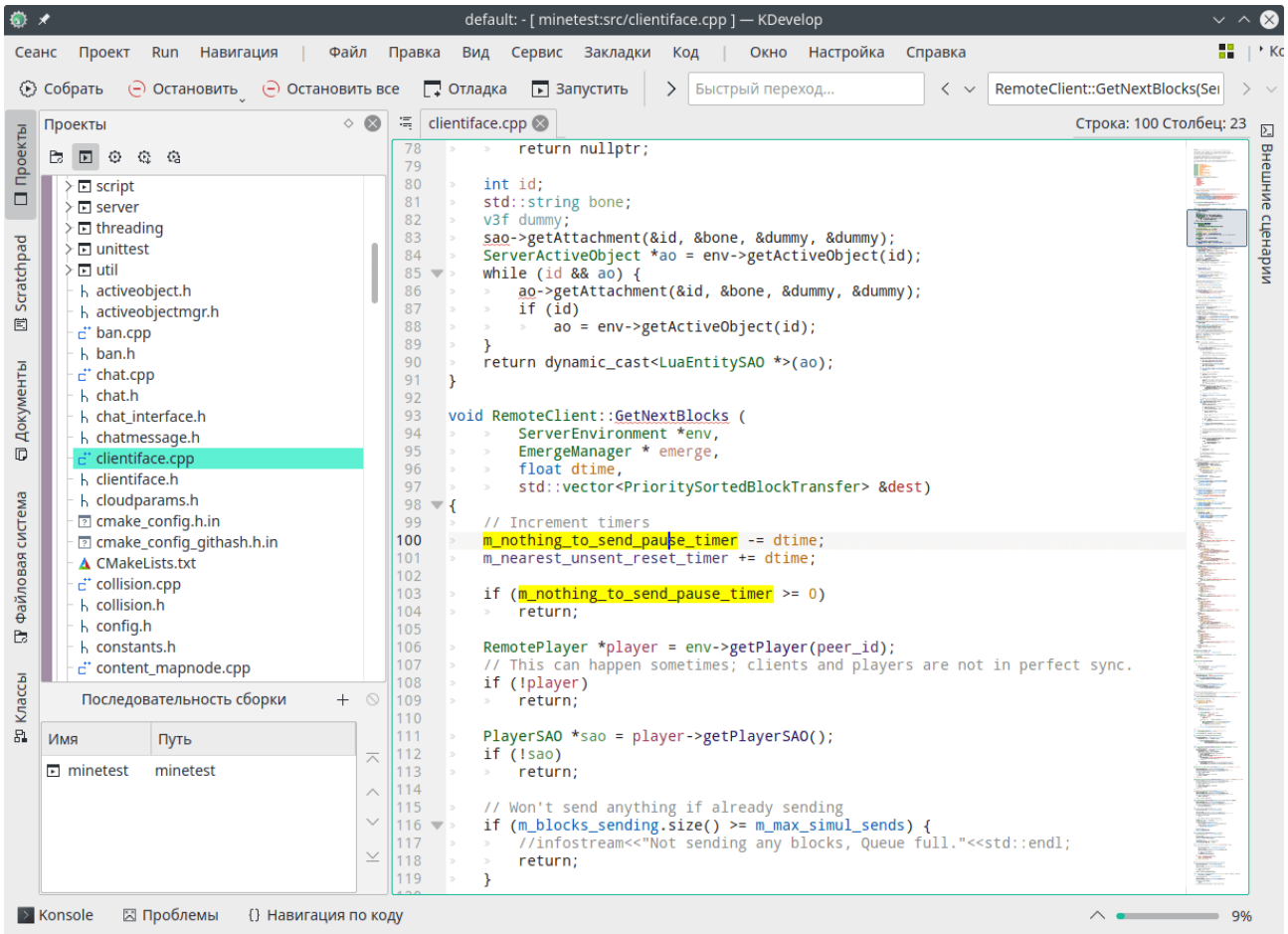

Рисунок 1.3 — Пример рабочего окна *KDevelop*

1.1.4 Выводы по обзору существующих редакторов кода

В результате обзора были выявлены следующие недостатки существующих популярных решений:

– имеют большой объём исходного кода, который разрабатывается многими людьми в течение нескольких лет;

– являются крупными проектами, которые в минимальном виде имеют лишние функции;

– требуют высокий уровень квалификации разработчиков, которые поддерживают исходный код в рабочем состоянии.

У небольших, малоизвестных существующих решений также присутствуют недостатки:

– реализованы так, что исходный код сложно поддерживать;

– имеют лицензию, которая ограничивает распространение и свободное использование исходного кода.

1.2 Функциональные требования к редактору кода

Исходя из задания на ВКР были выделены необходимые функции разрабатываемой интегрированной среды разработки:

– создать, открыть, сохранить, закрыть, редактировать, подключить, исключить, удалить файл проекта;

– создать, открыть, сохранить, закрыть проект;

– выбрать импортируемые проектом модули;

– выбрать язык программирования для подсветки его синтаксиса;

– запустить транслятор «*O2M*», компоновщик «*Link2M*» и генератор проекта для *C++* «*Make2M*»;

– выбрать и запустить компилятор для преобразования кода C++ в исполняемый файл;

– отобразить вывод транслятора «*O2M*», компоновщика «*Link2M*», генератора проекта для *C++* «*Make2M*», выбранного компилятора;

– запустить исполняемый файл.

1.3 Выводы по разделу 1

Рассмотрены широко известные редакторы программного кода, которые с каждым годом продолжают набирать популярность и расширять функционал. Сформированы функциональные требования к разрабатываемому редактору кода.

2 Проектирование редактора кода

В процессе разработки *RUP* используется более десяти диаграмм *UML*, поэтому при его использовании разработчикам сложно выбрать нужный набор диаграмм, которые могут быть взаимозаменяемые [4]. Процесс *ICONIX* является менее сложным, потому что в нём используется всего четыре вида диаграмм:

– диаграмма вариантов использования;

– диаграммы пригодности;

– диаграммы последовательности;

– диаграмма классов.

Используя процесс *XP* трудно предугадать затраты времени на проект, потому что в начале никто не знает полного списка требований. Также, успех *XP* сильно зависит от уровня программистов [5]. Процесс разработки *ICONIX* лишён этих недостатков, потому что использует более структурированный подход к написанию кода, минимизирует количество технических проблем при разработке и позволяет быстро изменять требования и архитектуру приложения.

Основываясь на вышеизложенных сравнениях, в качестве процесса проектирования был выбран *ICONIX*.

2.1 Макет интерфейса приложения

Макет интерфейса главного окна приложения представлен на рисунке 2.1.

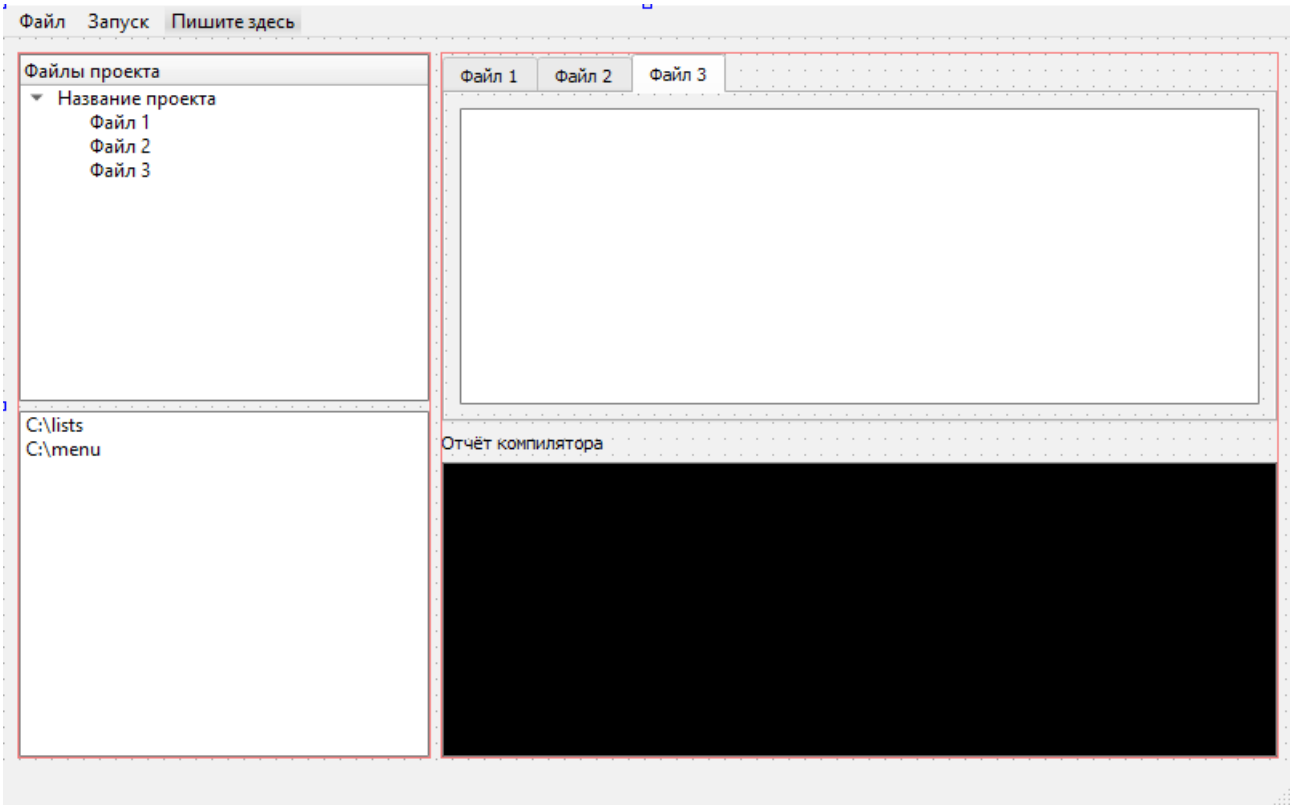

Рисунок 2.1 — Окно главное

2.2 Диаграмма прецедентов

Диаграмма вариантов использования приложения представлена на рисунке 2.2.

Рисунок 2.2 — Диаграмма вариантов использования, лист 1

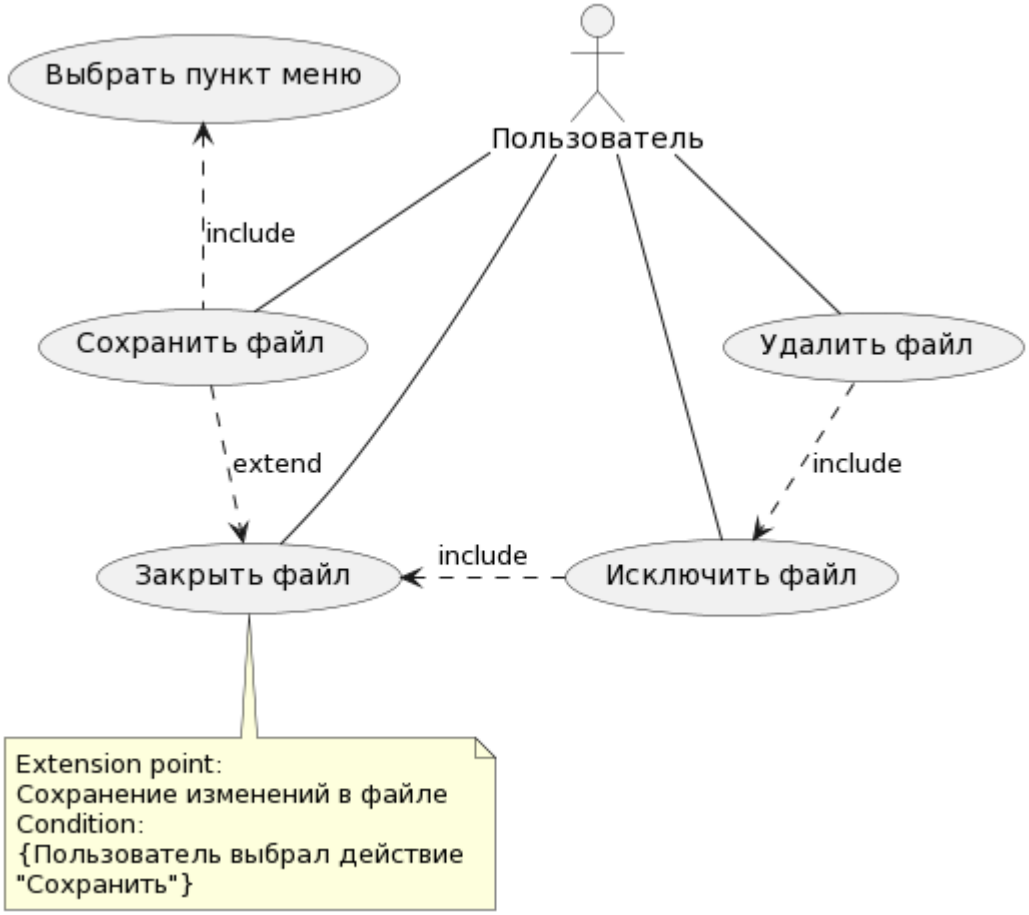

Рисунок 2.2, лист 2

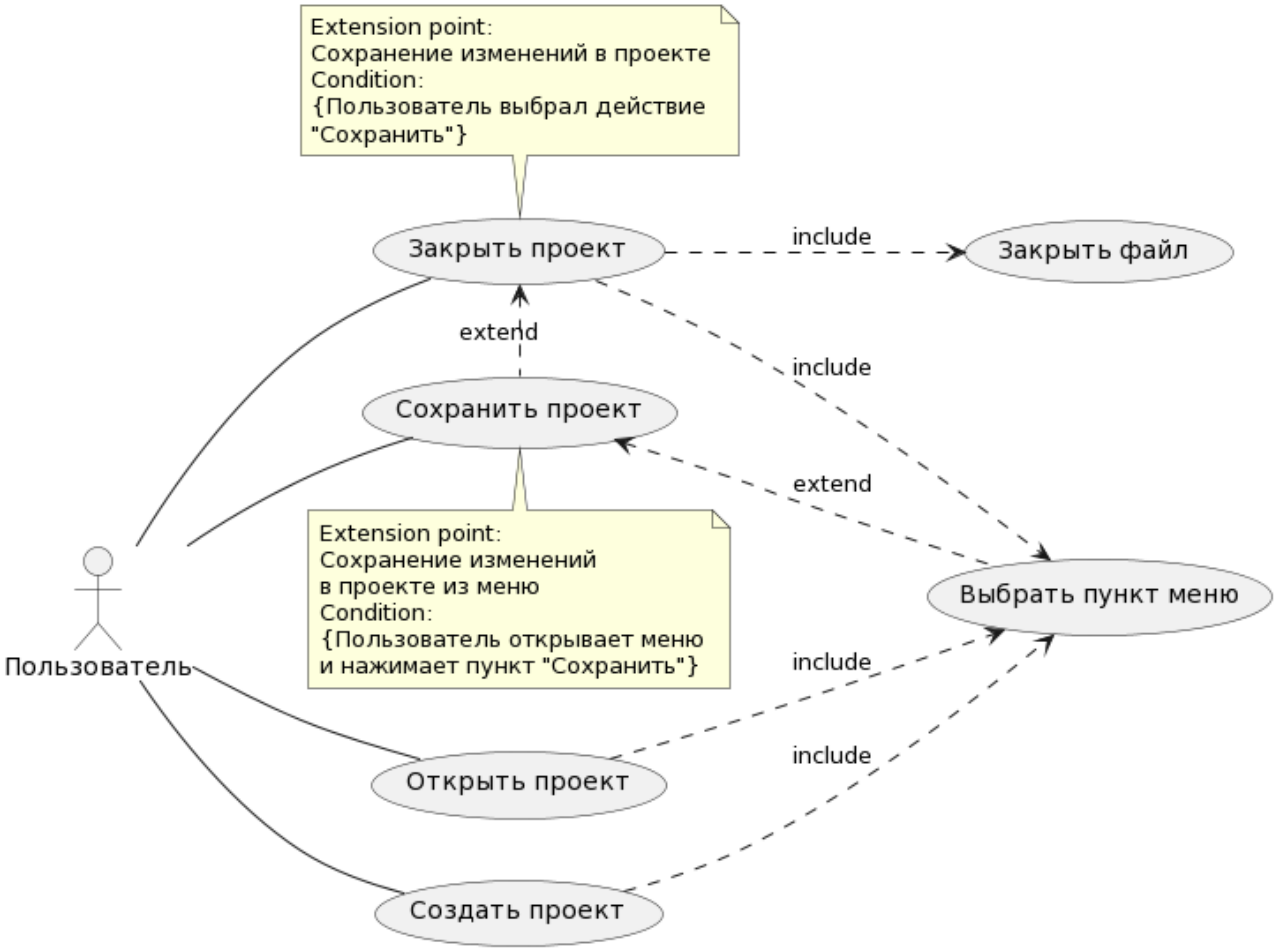

Рисунок 2.2, лист 3

Рисунок 2.2, лист 4

Приведённая выше диаграмма иллюстрирует:

– набор функций, которые обеспечат выполнение функциональных требований;

– повторы сценариев использования приложения, выделенные в отдельные прецеденты;

– отношения между прецедентами (включение, расширение);

– комментарии с описанием точек расширения;

– отношения ассоциации между пользователем и прецедентами.

2.3 Текстовое описание прецедентов

Название прецедента: выбрать пункт меню.

Цель: выбрать в меню пункт, который обозначает действие над файлом или проектом.

Предусловия: открыто главное окно.

Главная последовательность:

– нажать на одно из меню;

– раскроется выбранное меню;

– нажать на один из пунктов меню.

Название прецедента: подключить файл.

Цель: включить существующий файл в проект.

Предусловия: выполнен прецедент «Выбрать пункт меню», открыт проект.

Главная последовательность:

– приложение откроет диалоговое окно для выбора файла;

– выбрать необходимый файл и нажать на кнопку «Открыть»;

– диалоговое окно для выбора файла закроется;

– приложение включит файл в проект и отобразит его в обозревателе проекта;

– приложение отметит название проекта в наименовании главного окна символом «звёздочка» — это показывает пользователю, что присутствуют несохранённые изменения в проекте.

Альтернативная последовательность (пользователь выбирает файл, который уже включен в проект):

– приложение откроет диалоговое окно для выбора файла;

– выбрать необходимый файл и нажать на кнопку «Открыть»;

– диалоговое окно для выбора файла закроется;

– приложение отобразит сообщение о том, что файл уже включен в проект.

Название прецедента: создать файл.

Цель: открыть новую вкладку для написания программного кода.

Предусловия: выполнен прецедент «Выбрать пункт меню», открыт проект.

Главная последовательность:

– приложение откроет диалоговое окно для выбора файла;

– написать название нового файла и нажать на кнопку «Сохранить»;

– диалоговое окно для выбора файла закроется;

– приложение создаст новый файл;

– переход к прецеденту «Подключить файл».

Название прецедента: создать вкладку.

Цель: создать вкладку для редактирования файла.

Предусловия: выполнен вход из другого прецедента.

Главная последовательность:

– приложение выведет в главное окно новую пустую вкладку для дальнейшего написания кода и сделает её активной;

– приложение подсветит в тексте вкладки синтаксис языка по умолчанию.

Название прецедента: открыть файл.

Цель: открыть существующий файл для редактирования.

Предусловия: открыто главное окно, открыт проект.

Главная последовательность:

– нажать правой кнопкой мыши на файл в обозревателе проекта;

– приложение откроет контекстное меню;

– нажать на пункт «Открыть»;

– приложение выведет в главное окно новую вкладку и сделает её активной;

– приложение загрузит во вкладку текст файла;

– приложение подсветит в тексте вкладки синтаксис языка по умолчанию.

Альтернативная последовательность (пользователь выбирает уже открытый файл):

– нажать правой кнопкой мыши на файл в обозревателе проекта;

– приложение откроет контекстное меню;

– нажать на пункт «Открыть»;

– приложение сделает активной вкладку выбранного файла;

– приложение подсветит в тексте вкладки синтаксис выбранного языка.

Название прецедента: редактировать файл.

Цель: редактировать программный код во вкладке.

Предусловия: в главном окне открыта вкладка.

Главная последовательность:

– нажать на область редактирования и изменить текст вкладки;

– приложение отметит вкладку символом «звёздочка» — это показывает пользователю, что присутствуют несохранённые изменения в файле;

– приложение подсветит в тексте вкладки синтаксис выбранного языка.

Название прецедента: назначить файл главным в проекте.

Цель: назначить выбранный файл главным файлом в проекте.

Предусловия: открыто главное окно, открыт проект.

Главная последовательность:

– нажать правой кнопкой мыши на файл в обозревателе проекта;

– приложение откроет контекстное меню;

– нажать на пункт «Сделать главным»;

– приложение подсветит название файла в обозревателе проекта красным цветом на сером фоне;

– приложение отметит название проекта в наименовании главного окна символом «звёздочка».

Название прецедента: сохранить файл.

Цель: записать в файл изменения, внесённые в текст вкладки.

Предусловия: выполнен прецедент «Выбрать пункт меню», открыт файл.

Главная последовательность:

– приложение сохранит текст активной вкладки в файл;

– приложение снимет отметку символом «звёздочка» с вкладки.

Название прецедента: закрыть файл.

Цель: закрыть вкладку.

Предусловия: открыт файл.

Главная последовательность:

– приложение закроет активную вкладку.

Альтернативная последовательность (вкладка отмечена символом «звёздочка», пользователь не желает сохранить изменения):

– приложение отобразит диалоговое окно с вопросом о том, желает ли пользователь сохранить изменения;

– нажать на кнопку «Не сохранять»;

– приложение закроет диалоговое окно;

– переход к главной последовательности прецедента «Закрыть файл».

Альтернативная последовательность (вкладка отмечена символом «звёздочка», пользователь желает сохранить изменения):

– приложение отобразит диалоговое окно с вопросом о том, желает ли пользователь сохранить изменения;

– нажать на кнопку «Сохранить»;

– приложение закроет диалоговое окно;

– переход к прецеденту «Сохранить файл»;

– переход к главной последовательности прецедента «Закрыть файл».

Название прецедента: исключить файл.

Цель: исключить файл из проекта.

Предусловия: открыто главное окно, открыт проект.

Главная последовательность:

– нажать правой кнопкой мыши на файл в обозревателе проекта;

– приложение откроет контекстное меню;

– нажать на пункт «Исключить»;

– приложение отобразит диалоговое окно с вопросом о том, желает ли пользователь исключить файл из проекта;

– нажать на кнопку «Да»;

– приложение закроет диалоговое окно;

– переход к прецеденту «Закрыть файл»;

– приложение удалит файл из обозревателя проекта;

– приложение отметит название проекта в наименовании главного окна символом «звёздочка».

Альтернативная последовательность (пользователь передумал исключать файл):

– нажать правой кнопкой мыши на файл в обозревателе проекта;

– приложение откроет контекстное меню;

– нажать на пункт «Исключить»;

– приложение отобразит диалоговое окно с вопросом о том, желает ли пользователь исключить файл из проекта;

– нажать на кнопку «Нет»;

– приложение закроет диалоговое окно.

Название прецедента: удалить файл.

Цель: удалить файл из файловой системы.

Предусловия: открыто главное окно, открыт проект.

Главная последовательность:

– нажать правой кнопкой мыши на файл в обозревателе проекта;

– приложение откроет контекстное меню;

– нажать на пункт «Удалить»;

– приложение отобразит диалоговое окно с вопросом о том, желает ли пользователь удалить файл навсегда;

– нажать на кнопку «Да»;

– приложение закроет диалоговое окно;

– переход к прецеденту «Исключить файл»;

– приложение удалит файл из файловой системы.

Альтернативная последовательность (пользователь передумал удалять файл):

– нажать правой кнопкой мыши на файл в обозревателе проекта;

– приложение откроет контекстное меню;

– нажать на пункт «Удалить»;

– приложение отобразит диалоговое окно с вопросом о том, желает ли пользователь удалить файл навсегда;

– нажать на кнопку «Нет»;

– приложение закроет диалоговое окно.

Название прецедента: выбрать язык.

Цель: выбрать язык программирования для подсветки его синтаксиса.

Предусловия: выполнен прецедент «Выбрать пункт меню».

Главная последовательность:

– приложение подсветит в тексте вкладки синтаксис выбранного языка.

Название прецедента: добавить путь к модулям.

Цель: добавить путь к каталогам, содержащим файлы определений *O2M* (**.dfn*).

Предусловия: выполнен прецедент «Выбрать пункт меню».

Главная последовательность:

– приложение откроет диалоговое окно для выбора директории;

– выбрать необходимую директорию и нажать на кнопку «Открыть»;

– диалоговое окно для выбора директории закроется;

– приложение отобразит директорию в списке подключаемых директорий.

Альтернативная последовательность (пользователь выбирает директорию, который уже присутствует в списке подключаемых директорий):

– приложение откроет диалоговое окно для выбора директории;

– выбрать необходимую директорию и нажать на кнопку «Открыть»;

– диалоговое окно для выбора директории закроется;

– приложение отобразит сообщение о том, что директория уже находится в списке подключаемых директорий.

Название прецедента: удалить путь к модулям.

Цель: удалить путь к каталогам, содержащим файлы определений *O2M* (**.dfn*).

Предусловия: открыто главное окно.

Главная последовательность:

– нажать правой кнопкой мыши на директорию в списке подключаемых директорий;

– приложение откроет контекстное меню;

– нажать на пункт «Удалить»;

– приложение удалит директорию из списка подключаемых директорий.

Название прецедента: скомпилировать код проекта.

Цель: запустить компиляцию программного кода и посмотреть отчёт транслятора, компоновщика и генератора проекта для *C++* в виджете «Консоль».

Предусловия: выполнен прецедент «Выбрать пункт меню», открыт проект.

Главная последовательность:

– приложение создаст временные каталоги (*CPP, DFN, Make*) в директории проекта или очистит их;

– приложение очистит виджет «Консоль»;

– приложение запустит транслятор «*O2M*» и выведет его отчёт в виджет «Консоль»;

– приложение запустит компоновщик «*Link2M*» и выведет его отчёт в виджет «Консоль»;

– приложение запустит генератор проекта для *C++* «*Make2M*» и выведет его отчёт в виджет «Консоль».

Название прецедента: собрать проект.

Цель: запустить сборку проекта в исполняемый файл и посмотреть отчёт в виджете «Консоль».

Предусловия: выполнен прецедент «Выбрать пункт меню», во временном каталоге *CPP* присутствует файл «*Makefile.mak*», открыт проект.

Главная последовательность:

– приложение очистит виджет «Консоль»;

– приложение очистит временный каталог *Make* в директории проекта;

– приложение запустит утилиту «*nmake*» и выведет её отчёт в виджет «Консоль».

Название прецедента: запустить проект.

Цель: запустить исполняемый файл проекта.

Предусловия: выполнен прецедент «Выбрать пункт меню», во временном каталоге *Make* присутствует исполняемый файл, открыт проект.

Главная последовательность:

– приложение создаст в каталоге *Make* бинарный файл для запуска исполняемого файла;

– приложение запустит бинарный файл.

Название прецедента: сохранить проект.

Цель: сохранить изменения в проекте.

Предусловия: открыт проект.

Главная последовательность:

– приложение сохранит информацию о проекте в файл проекта;

– приложение снимет отметку символом «звёздочка» с названия проекта в наименовании главного окна.

Альтернативная последовательность (пользователь желает сохранить проект из меню):

– переход к прецеденту «Выбрать пункт меню»;

– переход к главной последовательности прецедента «Сохранить проект».

Название прецедента: закрыть проект.

Цель: закрыть текущий проект.

Предусловия: выполнен прецедент «Выбрать пункт меню», открыт проект.

Главная последовательность:

– пока не закрыты все вкладки происходит переход к прецеденту «Закрыть файл»;

– приложение очистит виджет «Обозреватель проекта»;

– приложение удалит название проекта из наименования главного окна.

Альтернативная последовательность (название проекта отмечено символом «звёздочка», пользователь не желает сохранить изменения):

– пока не закрыты все вкладки происходит переход к прецеденту «Закрыть файл»;

– приложение отобразит диалоговое окно с вопросом о том, желает ли пользователь сохранить изменения в проекте;

– нажать на кнопку «Не сохранять»;

– приложение закроет диалоговое окно;

– приложение очистит виджет «Обозреватель проекта»;

– приложение удалит название проекта из наименования главного окна.

Альтернативная последовательность (название проекта отмечено символом «звёздочка», пользователь желает сохранить изменения):

– пока не закрыты все вкладки происходит переход к прецеденту «Закрыть файл»;

– приложение отобразит диалоговое окно с вопросом о том, желает ли пользователь сохранить изменения в проекте;

– нажать на кнопку «Сохранить»;

– приложение закроет диалоговое окно;

– переход к прецеденту «Сохранить проект»;

– приложение очистит виджет «Обозреватель проекта»;

– приложение удалит название проекта из наименования главного окна.

Название прецедента: создать проект.

Цель: создать новый проект.

Предусловия: выполнен прецедент «Выбрать пункт меню», нет открытого проекта.

Главная последовательность:

– приложение откроет диалоговое окно для выбора директории;

– выбрать необходимую директорию и нажать на кнопку «Открыть»;

– диалоговое окно для выбора директории закроется;

– приложение создаст начальный файл в проекте;

– приложение создаст файл проекта с информацией о проекте;

– приложение отобразит файлы проекта в обозревателе проекта;

– приложение добавит к наименованию главного окна название проекта.

Название прецедента: открыть проект.

Цель: открыть существующий проект.

Предусловия: выполнен прецедент «Выбрать пункт меню», нет открытого проекта.

Главная последовательность:

– приложение откроет диалоговое окно для выбора директории;

– выбрать необходимую директорию и нажать на кнопку «Открыть»;

– диалоговое окно для выбора директории закроется;

– приложение отобразит файлы проекта в обозревателе проекта;

– приложение добавит к наименованию главного окна название проекта.

2.4 Диаграммы пригодности

Диаграммы пригодности строятся для каждого прецедента. На них выделяются объекты, которые участвуют в реализации прецедента, и их обязанности в системе.

Диаграмма пригодности для прецедента «Выбрать пункт меню» представлена на рисунке 2.3.

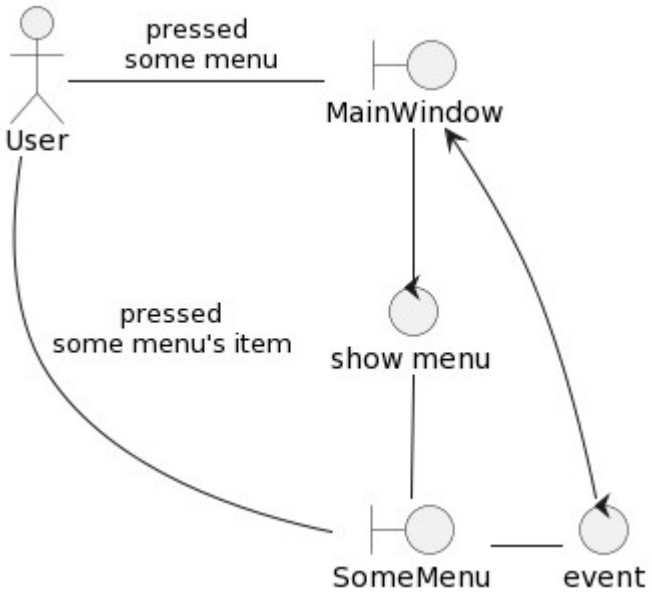

Рисунок 2.3 — Диаграмма пригодности для прецедента «Выбрать пункт меню»

В главном окне пользователь нажимает на одно из меню. Выбранное меню раскроется. Пользователь выбирает пункт меню, который его интересует, и нажимает на него. В главное окно отправится событие нажатия пункта меню, которое будет обработано в других прецедентах.

Диаграмма пригодности для прецедента «Подключить файл» представлена на рисунке 2.4

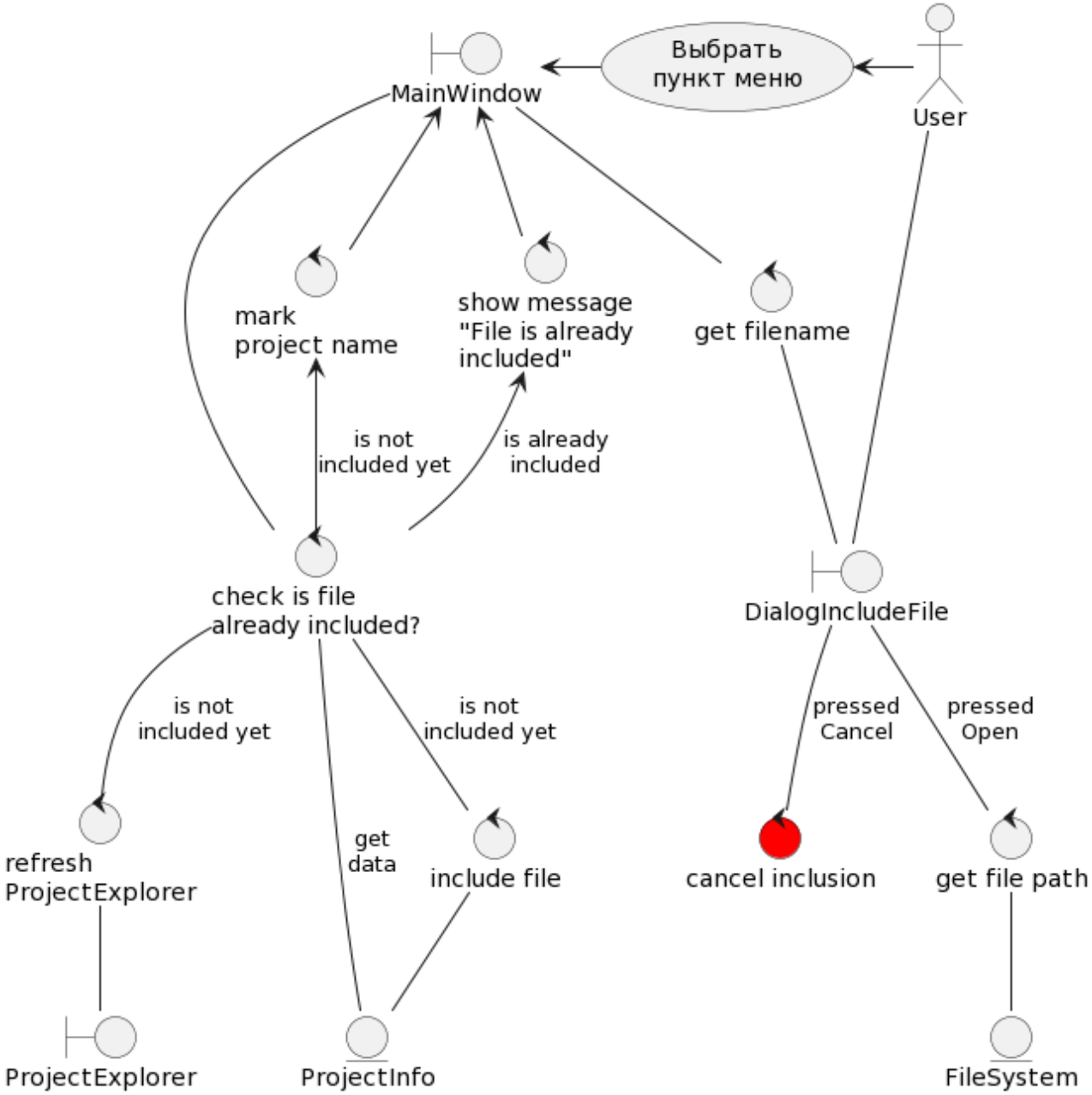

Рисунок 2.4 — Диаграмма пригодности для прецедента «Подключить файл»

После выполнения пользователем прецедента «Выбрать пункт меню» главное окно создаёт экземпляр диалогового окна и запрашивает у него имя подключаемого файла. Пользователь выбирает файл диалоговом окне или отменяет подключение файла. Главное окно, получая имя файла, проверяет, включен ли уже этот файл в проект. Если файл уже включен в проект, то главное окно отображает сообщение о том, что файл уже включен в проект. Если файл не включен в проект, то главное окно включит файл в проект

(добавит имя файла в структуру данных «Информация о проекте», которая хранится в файле проекта), обновит список файлов проекта в обозревателе проекта, отметит название проекта символом «звёздочка».

Диаграмма пригодности для прецедента «Создать файл» представлена на рисунке 2.5.

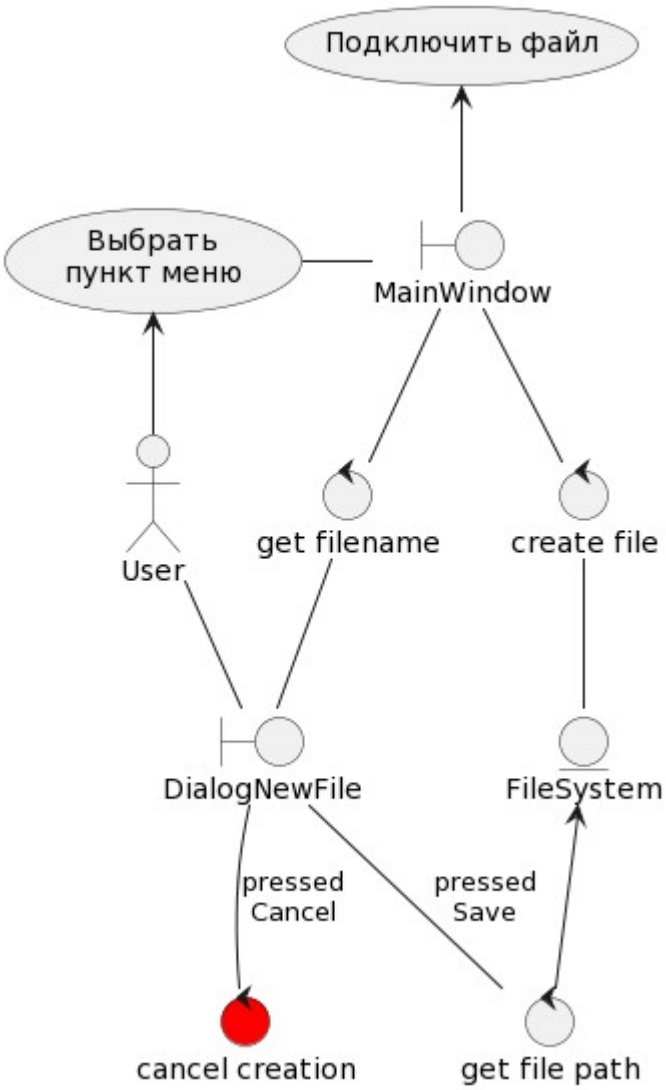

Рисунок 2.5 — Диаграмма пригодности для прецедента «Создать файл»

После выполнения пользователем прецедента «Выбрать пункт меню» главное окно создаёт экземпляр диалогового окна и запрашивает у него имя нового файла. Пользователь вводит имя файла в диалоговом окне или отменяет

создание файла. Главное окно, получая имя файла, создаёт его. Дальше происходит переход к прецеденту «Подключить файл».

Диаграмма пригодности для прецедента «Создать вкладку» представлена на рисунке 2.6.

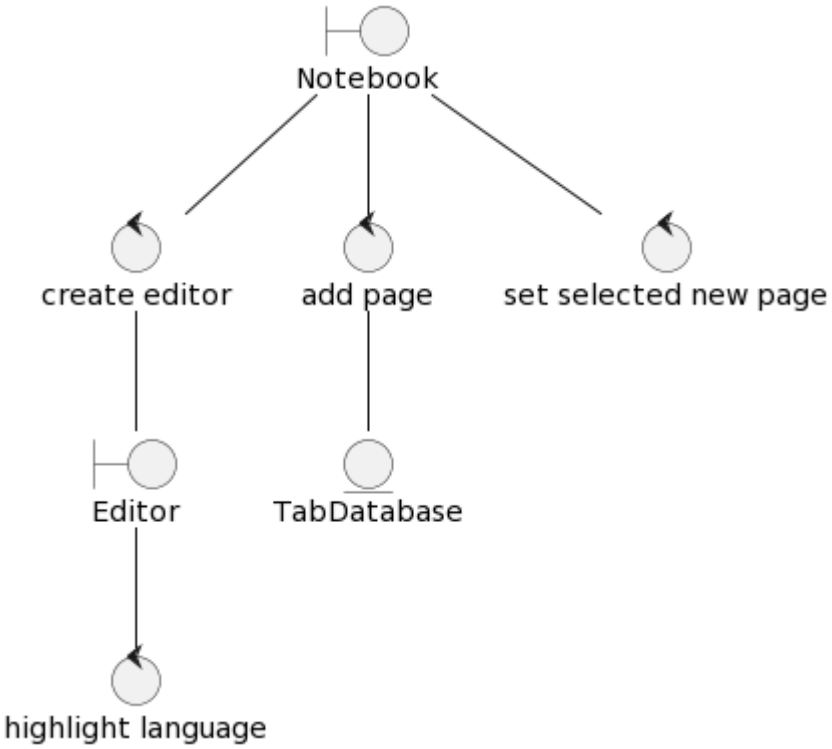

Рисунок 2.6 — Диаграмма пригодности для прецедента «Создать вкладку»

После перехода из другого прецедента в данный прецедент виджет «Блокнот» создаёт экземпляр виджета «Редактор». Виджет «Редактор» при создании подсвечивает синтаксис языка программирования, установленного по умолчанию. Виджет «Блокнот» добавляет созданный виджет в своё хранилище вкладок и делает эту новую вкладку активной.

Диаграмма пригодности для прецедента «Открыть файл» представлена на рисунке 2.7.

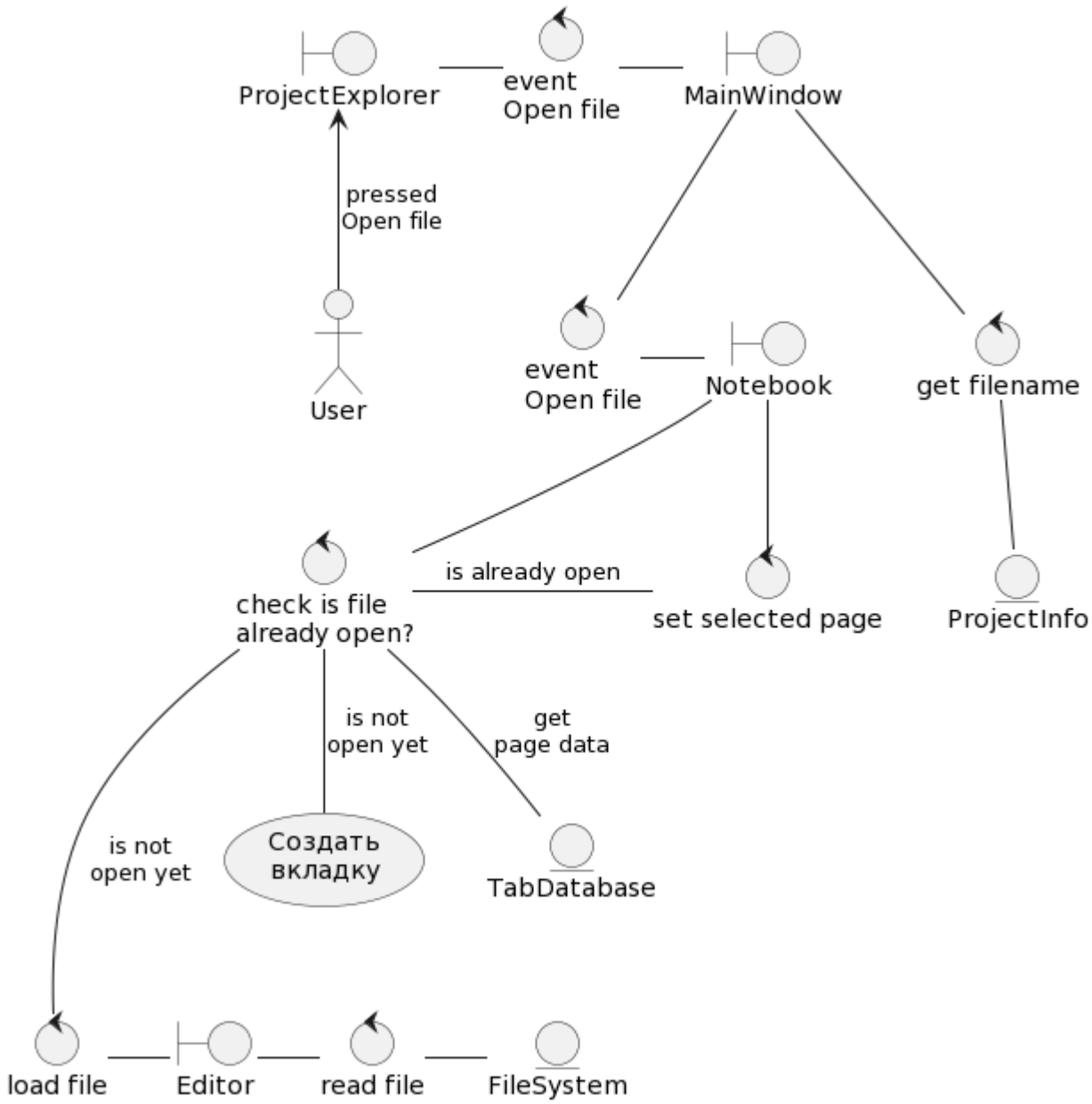

Рисунок 2.7 — Диаграмма пригодности для прецедента «Открыть файл»

При нажатии пользователем пункта «Открыть» в контекстном меню виджета «Обозреватель проекта» отправляется событие «Открыть файл» в главное окно. Главное окно запрашивает имя файла из структуры «Информация о проекте», прикрепляет его к информации события и перенаправляет событие «Открыть файл» для дальнейшей его обработки в виджет «Блокнот». Виджет «Блокнот» проверяет, открыт ли уже этот файл в одной из вкладок. Если файл уже открыт, то виджет «Блокнот» делает активной вкладку с файлом. Если файл не открыт, то виджет «Блокнот» выполняет прецедент «Создать вкладку»

и после этого просит новую страницу загрузить в себя содержимое открываемого файла.

Диаграмма пригодности для прецедента «Редактировать файл» представлена на рисунке 2.8.

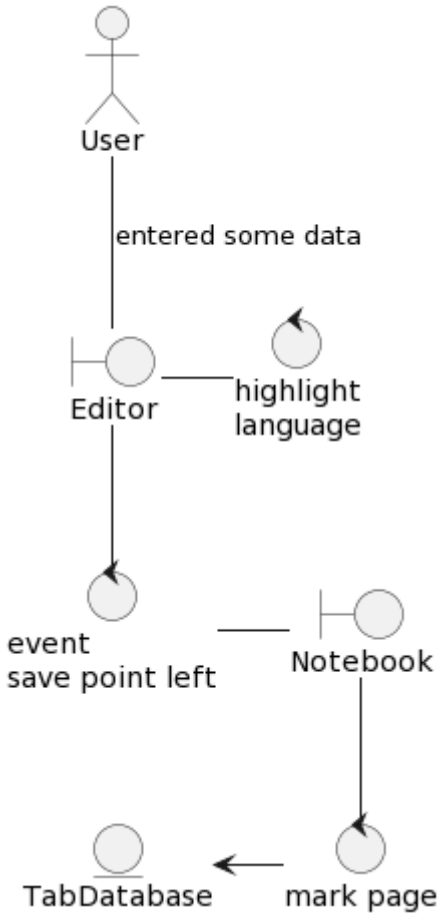

Рисунок 2.8 — Диаграмма пригодности для прецедента «Редактировать файл»

Пользователь изменяет содержимое страницы. Страница заново подсвечивает синтаксис языка программирования и создаёт событие «Точка сохранения покинута», которое отправляет на обработку виджету «Блокнот». Виджет «Блокнот» помечает вкладку в хранилище символом «звёздочка».

Диаграмма пригодности для прецедента «Назначить файл главным в проекте» представлена на рисунке 2.9.

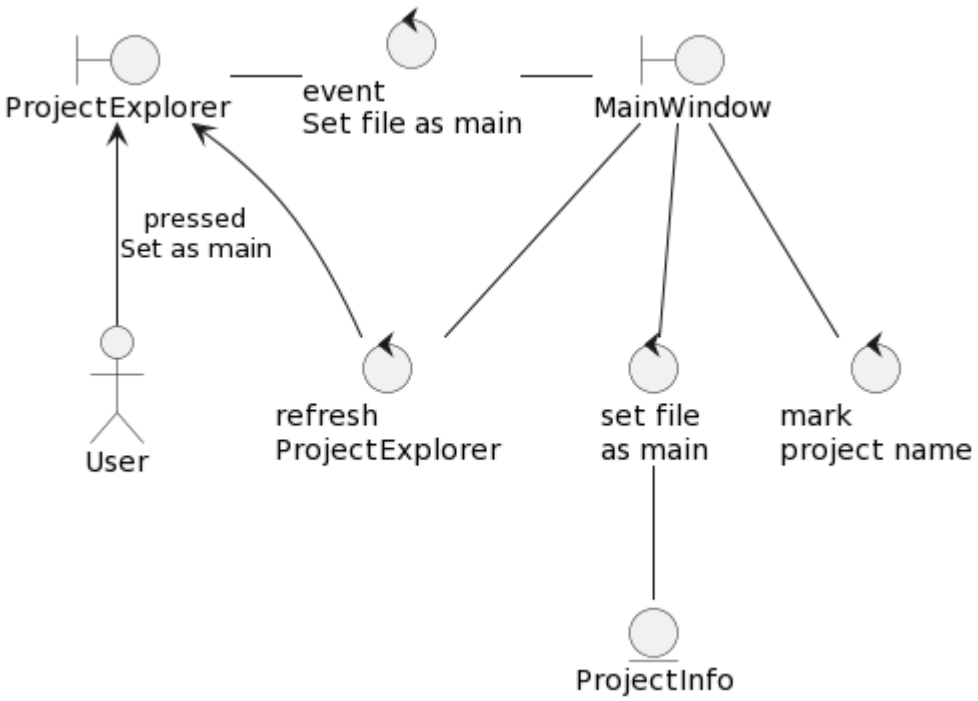

Рисунок 2.9 — Диаграмма пригодности для прецедента «Назначить файл главным в проекте»

При нажатии пользователем пункта «Назначить главным» в контекстном меню виджета «Обозреватель проекта» отправляется событие «Назначить файл главным в проекте» в главное окно. Главное окно изменяет в структуре «Информация о проекте» информацию об имени главного файла в проекте, отмечает название проекта в наименовании окна символом «звёздочка» и обновляет информацию в виджете «Обозреватель проекта».

Диаграмма пригодности для прецедента «Сохранить файл» представлена на рисунке 2.10.

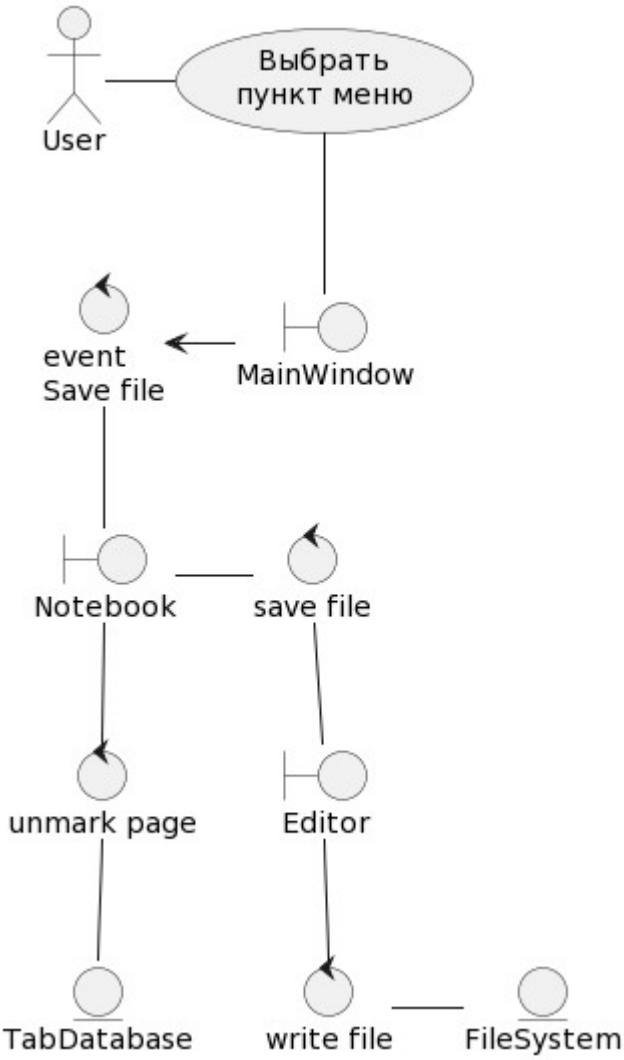

Рисунок 2.10 — Диаграмма пригодности для прецедента «Сохранить файл»

После выполнения пользователем прецедента «Выбрать пункт меню» главное окно перенаправляет событие «Сохранить файл» для дальнейшей его обработки в виджет «Блокнот». Виджет «Блокнот» обращается к активной вкладке с просьбой сохранить файл. Вкладка записывает в файл своё содержимое. После сохранения файла вкладкой виджет «Блокнот» снимает отметку символом «звёздочка» с вкладки в хранилище.

Диаграмма пригодности для прецедента «Закрыть файл» представлена на рисунке 2.11.

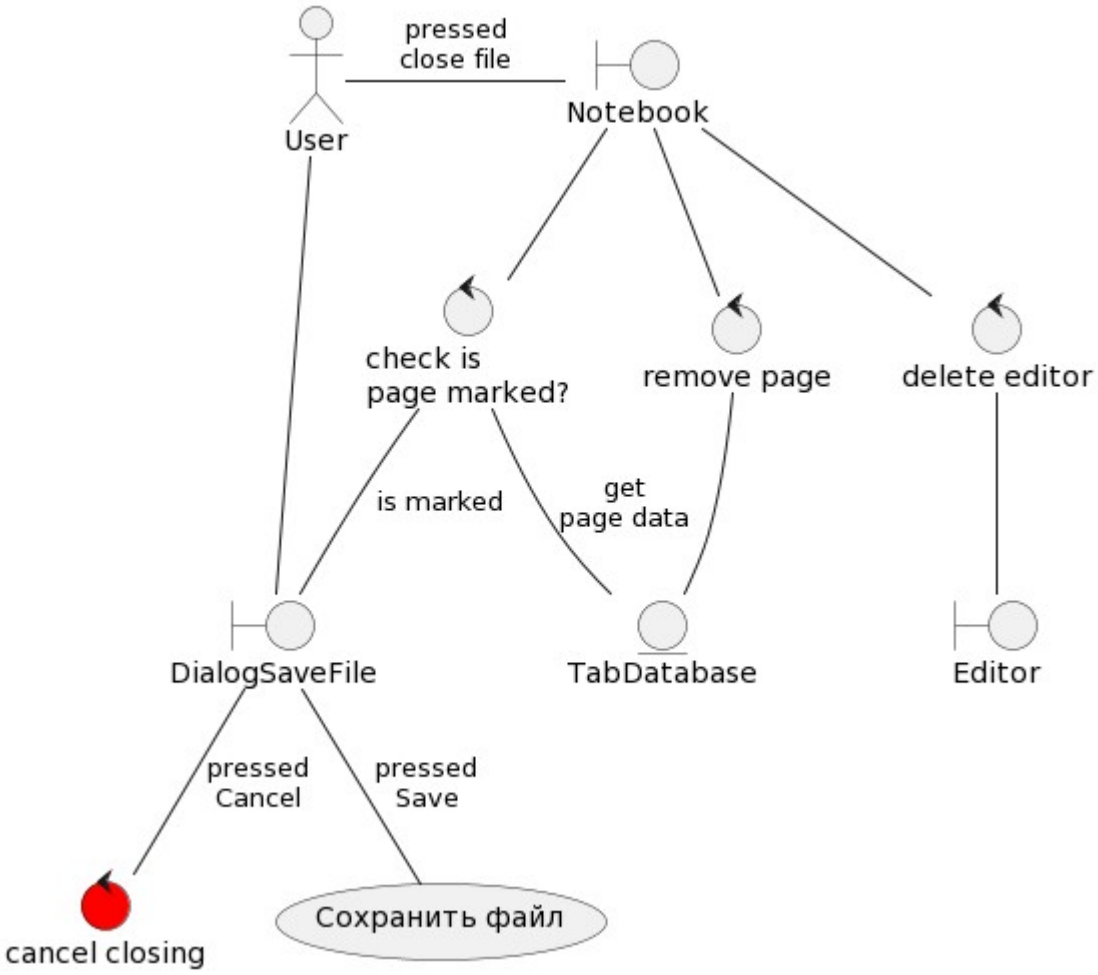

Рисунок 2.11 — Диаграмма пригодности для прецедента «Закрыть файл»

После нажатия пользователем символа «крестик» на активной вкладке виджет «Блокнот» проверяет, отмечена ли активная страница символом «звёздочка». Если страница отмечена, то виджет «Блокнот» создаёт экземпляр диалогового окна и запрашивает у него решение пользователя о дальнейших действиях с файлом. Пользователь в диалоговом окне выбирает: сохранить файл, закрыть файл или отменить закрытие файла. Если выбрано «Сохранить», то выполняется переход к прецеденту «Сохранить файл». После вышеописанных действий виджет «Блокнот» удаляет вкладку из хранилища и удаляет экземпляр вкладки.

Диаграмма пригодности для прецедента «Исключить файл» представлена на рисунке 2.12.

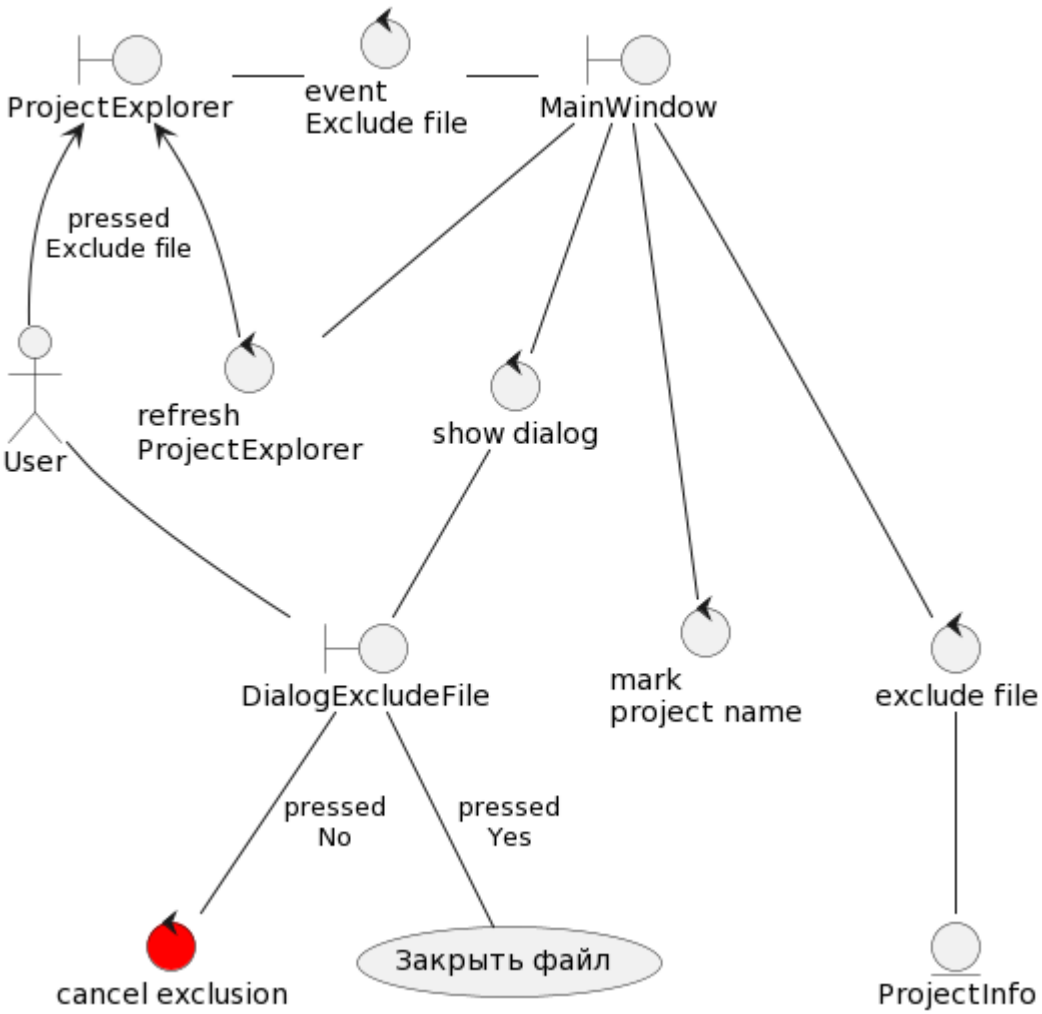

Рисунок 2.12 — Диаграмма пригодности для прецедента «Исключить файл»

При нажатии пользователем пункта «Исключить» в контекстном меню виджета «Обозреватель проекта» отправляется событие «Исключить файл» в главное окно. Главное окно создаёт экземпляр диалогового окна и запрашивает у него подтверждение действий пользователя. Пользователь в диалоговом окне подтверждает или отменяет исключение файла. Если выбрано «Да», то происходит переход к прецеденту «Закрыть файл». Дальше главное окно удаляет название файла из структуры «Информация о проекте», отмечает название проекта в наименовании окна символом «звёздочка» и обновляет информацию в виджете «Обозреватель проекта».

Диаграмма пригодности для прецедента «Удалить файл» представлена на рисунке 2.13.

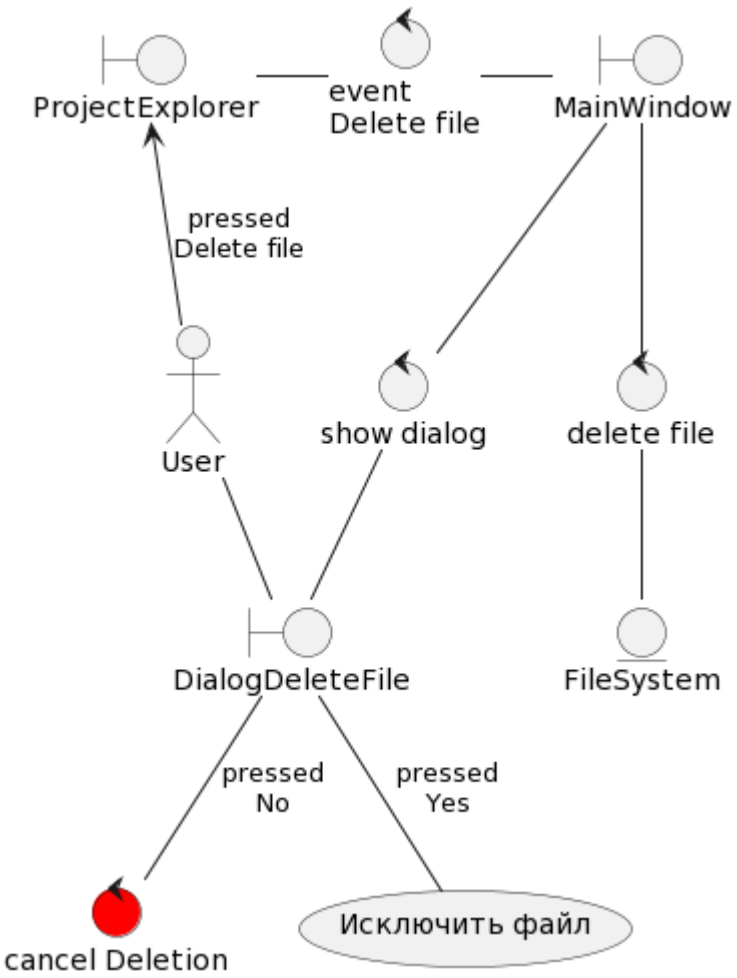

Рисунок 2.13 — Диаграмма пригодности для прецедента «Удалить файл»

При нажатии пользователем пункта «Удалить» в контекстном меню виджета «Обозреватель проекта» отправляется событие «Удалить файл» в главное окно. Главное окно создаёт экземпляр диалогового окна и запрашивает у него подтверждение действий пользователя. Пользователь в диалоговом окне подтверждает или отменяет удаление файла. Если выбрано «Да», то происходит переход к прецеденту «Исключить файл». Дальше главное окно удаляет файл.

Диаграмма пригодности для прецедента «Выбрать язык» представлена на рисунке 2.14.

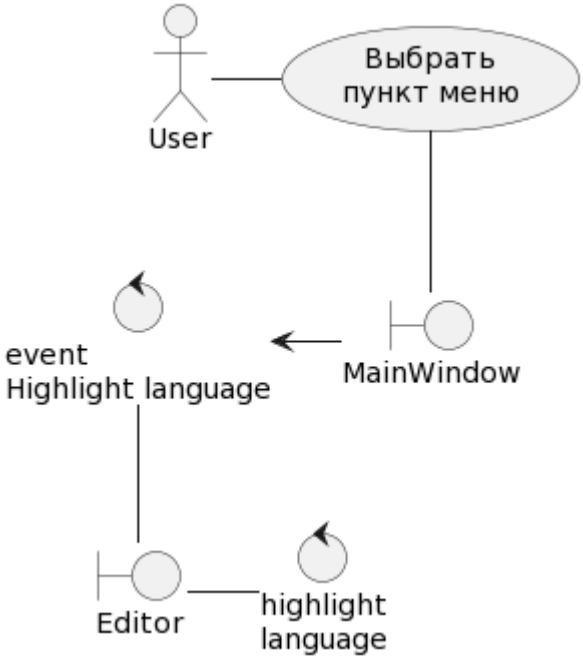

Рисунок 2.14 — Диаграмма пригодности для прецедента «Выбрать язык»

После выполнения пользователем прецедента «Выбрать пункт меню» главное окно перенаправляет событие «Подсветить язык» для дальнейшей его обработки в активную вкладку. Вкладка меняет подсвечиваемый язык программирования на требуемый.

Диаграмма пригодности для прецедента «Добавить путь к модулям» представлена на рисунке 2.15.

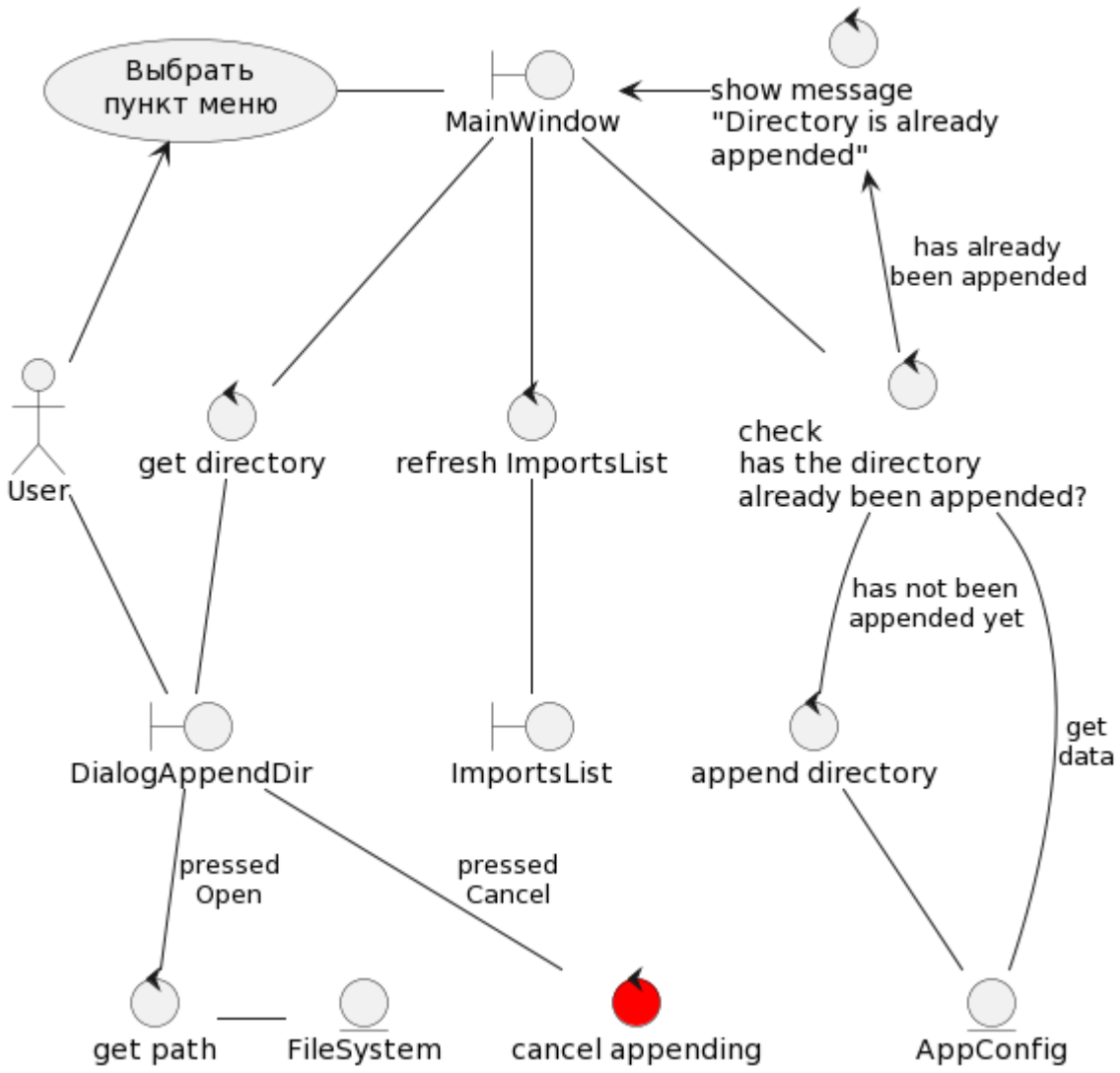

Рисунок 2.15 — Диаграмма пригодности для прецедента «Добавить путь к модулям»

После выполнения пользователем прецедента «Выбрать пункт меню» главное окно создаёт экземпляр диалогового окна и запрашивает у него путь к директории. Пользователь выбирает директорию в диалоговом окне или отменяет добавление директории. Главное окно, получая путь к директории, проверяет, добавлена ли уже эта директория в структуру данных «Настройки приложения». Если директория уже добавлена, то главное окно отображает сообщение о том, что директория уже добавлена. Если директория не добавлена, то главное окно добавит путь к директории в структуру «Настройки приложения» и обновит список подключаемых директорий.

Диаграмма пригодности для прецедента «Удалить путь к модулям» представлена на рисунке 2.16.

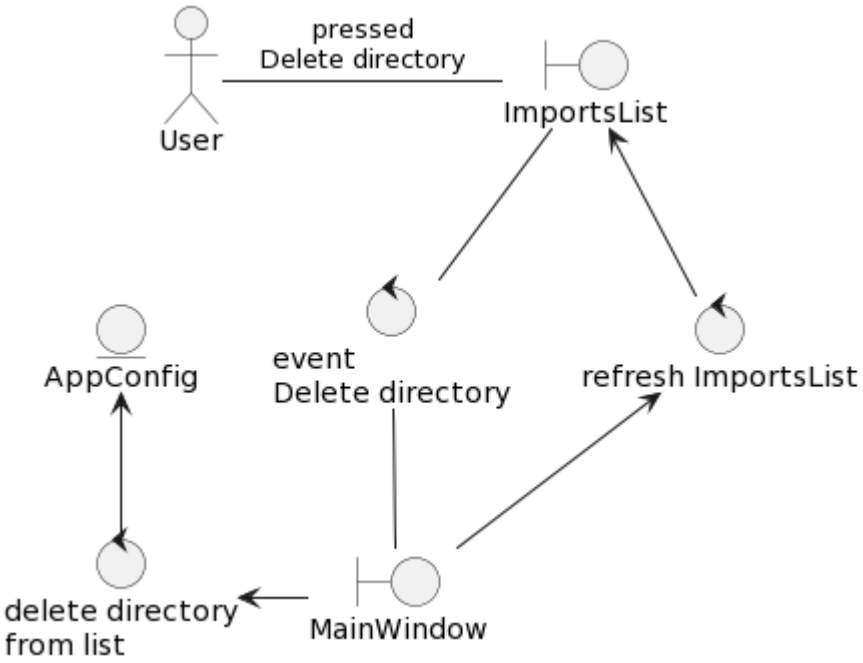

Рисунок 2.16 — Диаграмма пригодности для прецедента «Удалить путь к модулям»

При нажатии пользователем пункта «Удалить» в контекстном меню виджета «Список подключаемых директорий» отправляется событие «Удалить директорию» в главное окно. Главное окно удаляет директорию из структуры «Настройки приложения» и обновляет информацию в виджете «Список подключаемых директорий».

Диаграмма пригодности для прецедента «Скомпилировать код проекта» представлена на рисунке 2.17.

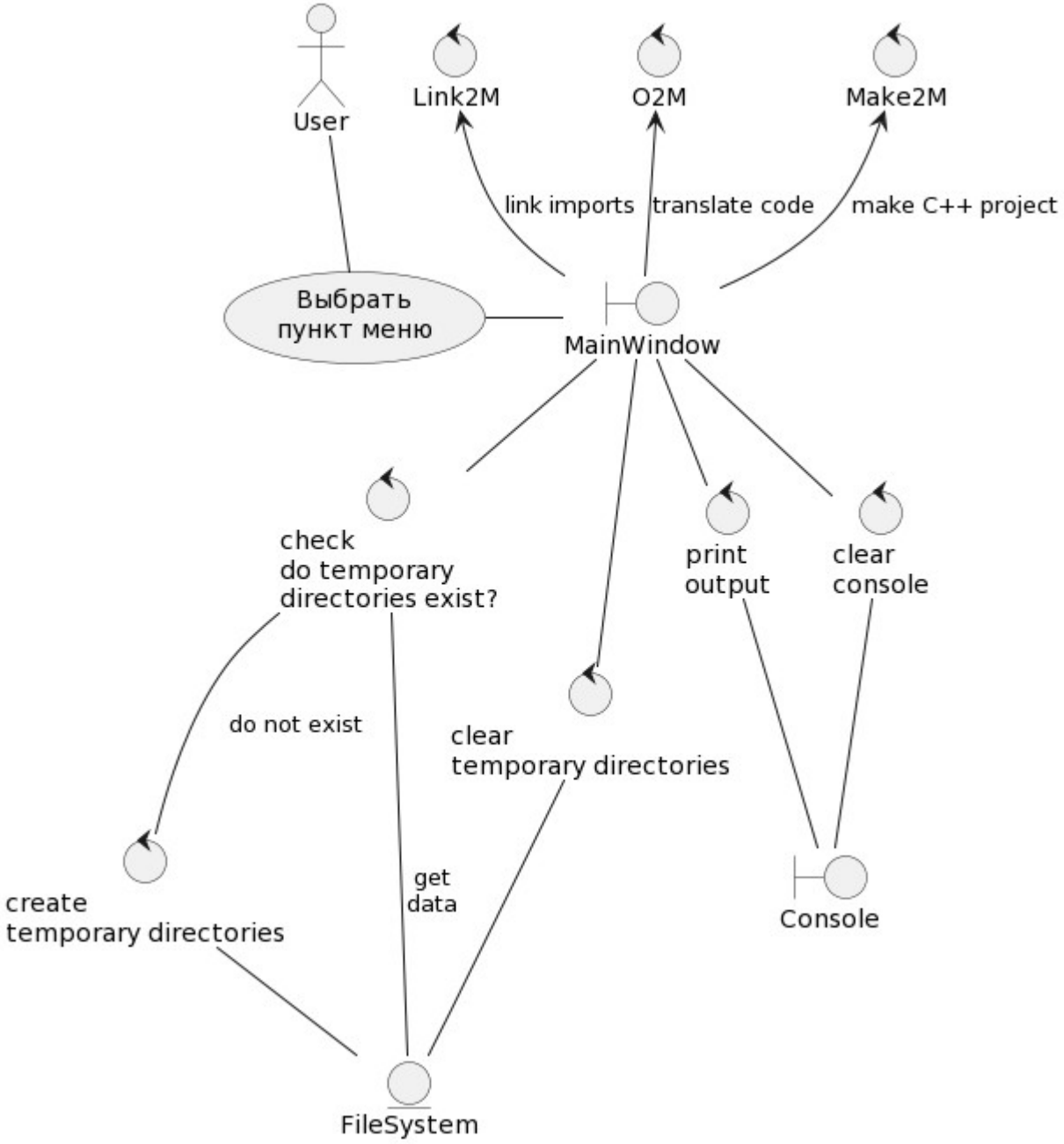

Рисунок 2.17 — Диаграмма пригодности для прецедента «Скомпилировать код проекта»

После выполнения пользователем прецедента «Выбрать пункт меню» главное окно проверяет, существуют ли временные директории. Если не существуют, то главное окно их создаёт. Дальше главное окно очищает временные директории и виджет «Консоль». После этого главное окно по очереди запускает транслятор «*O2M*», компоновщик «*Link2M*», генератор пректа для *C++* «*Make2M*» и выводит отчёты их работы в виджет «Консоль».

Диаграмма пригодности для прецедента «Собрать проект» представлена на рисунке 2.18.

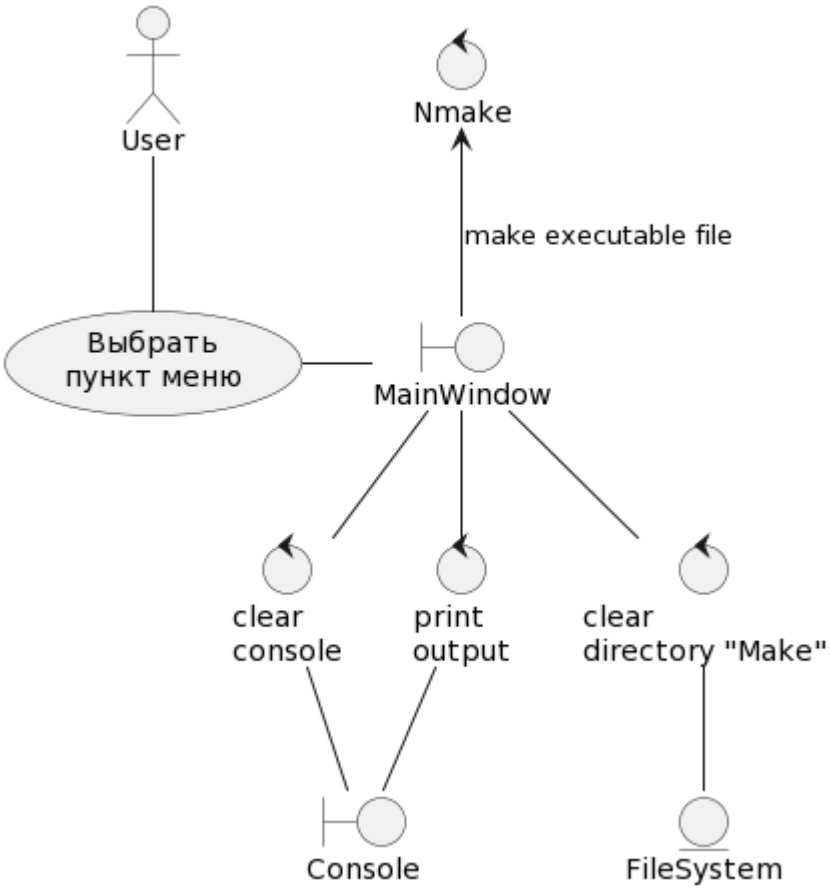

Рисунок 2.18 — Диаграмма пригодности для прецедента «Собрать проект»

После выполнения пользователем прецедента «Выбрать пункт меню» главное окно очищает временную директорию *Make* и виджет «Консоль». Дальше главное окно запускает утилиту «*nmake*» и выводит её отчёт в виджет «Консоль».

Диаграмма пригодности для прецедента «Запустить проект» представлена на рисунке 2.19.

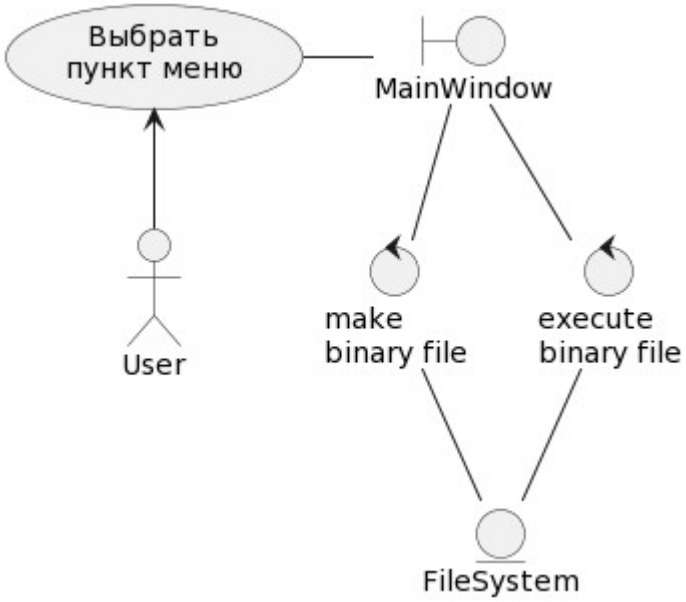

Рисунок 2.19 — Диаграмма пригодности для прецедента «Запустить проект»

После выполнения пользователем прецедента «Выбрать пункт меню» главное окно создаёт во временной директории *Make* бинарный файл для запуска исполняемого файла и запускает этот файл.

Диаграмма пригодности для прецедента «Сохранить проект» представлена на рисунке 2.20.

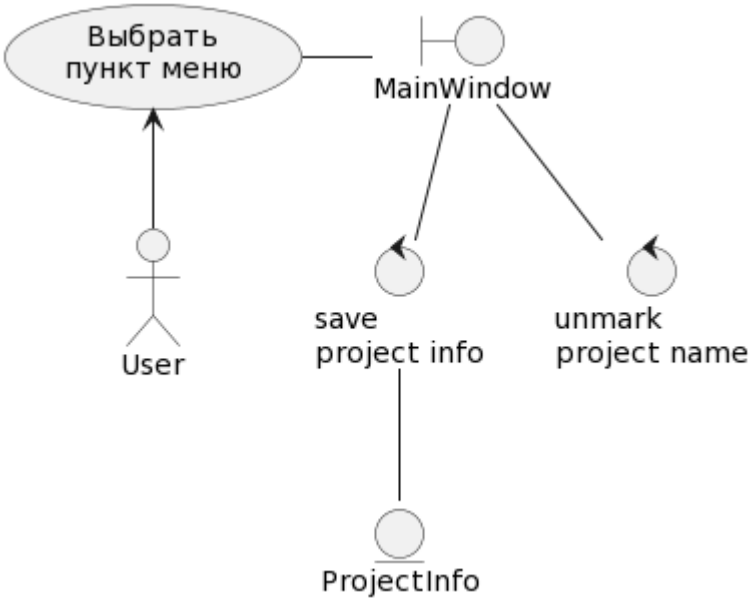

Рисунок 2.20 — Диаграмма пригодности для прецедента «Сохранить проект»

После выполнения пользователем прецедента «Выбрать пункт меню» главное окно сохраняет данные из структуры «Информация о проекте» в файл проекта и снимает отметку символом «звёздочка» с названия проекта в наименовании окна.

Диаграмма пригодности для прецедента «Закрыть проект» представлена на рисунке 2.21.

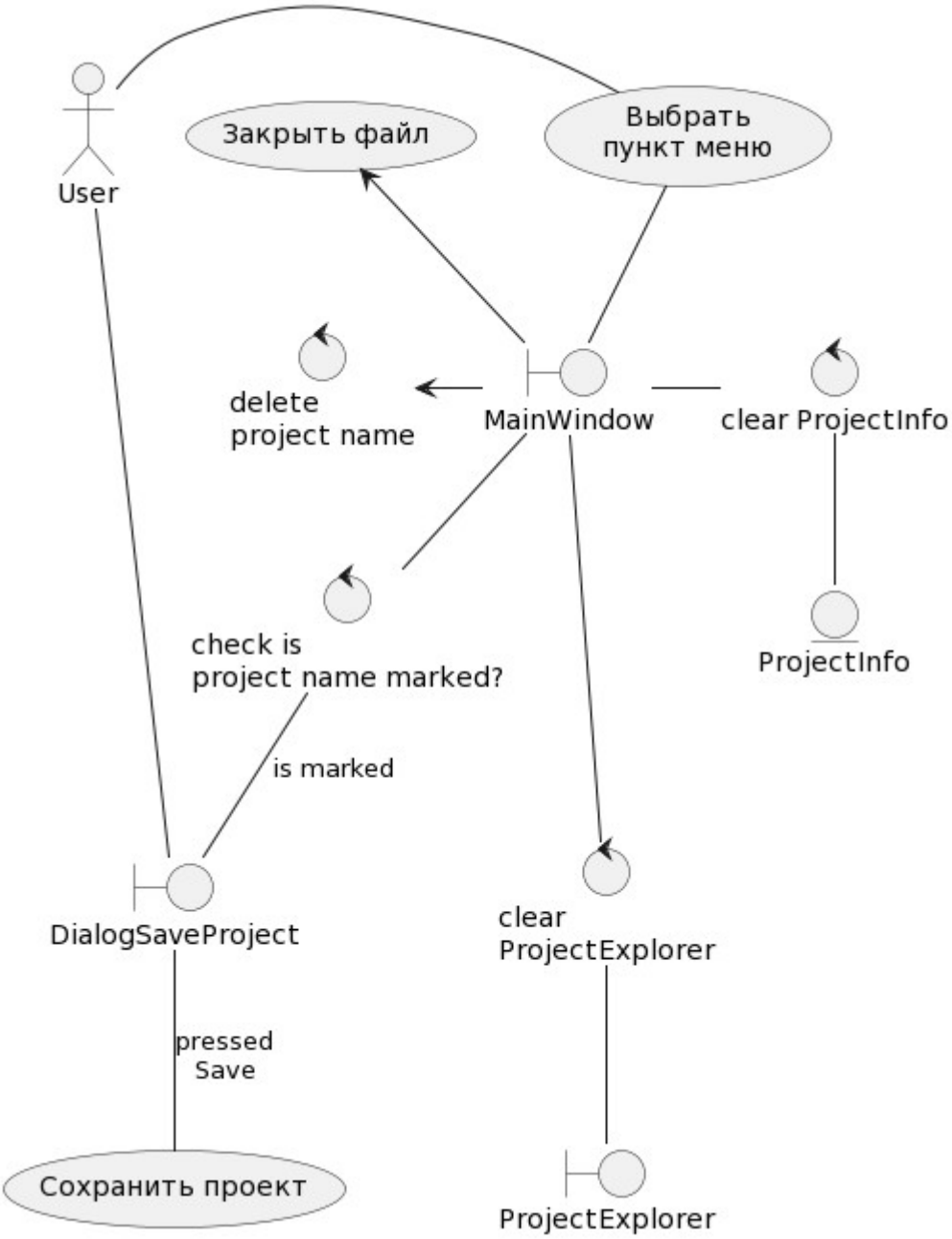

Рисунок 2.21 — Диаграмма пригодности для прецедента «Закрыть проект»

После выполнения пользователем прецедента «Выбрать пункт меню» главное окно осуществляет переход к выполнению прецедента «Закрыть файл», пока не закроются все вкладки или не будет отменено закрытие одной из вкладок. Затем, главное окно проверяет, отмечено ли название проекта символом «звёздочка». Если отмечено, то главное окно создаёт экземпляр диалогового окна и запрашивает у него решение пользователя о дальнейших действиях с проектом. Пользователь в диалоговом окне выбирает: сохранить изменения в проекте или не сохранять. Если выбрано «Сохранить», то выполняется переход к прецеденту «Сохранить проект». Дальше главное окно удаляет название проекта из наименования окна, очищает структуру «Информация о проекте» и виджет «Обозреватель проекта».

Диаграмма пригодности для прецедента «Создать проект» представлена на рисунке 2.22.

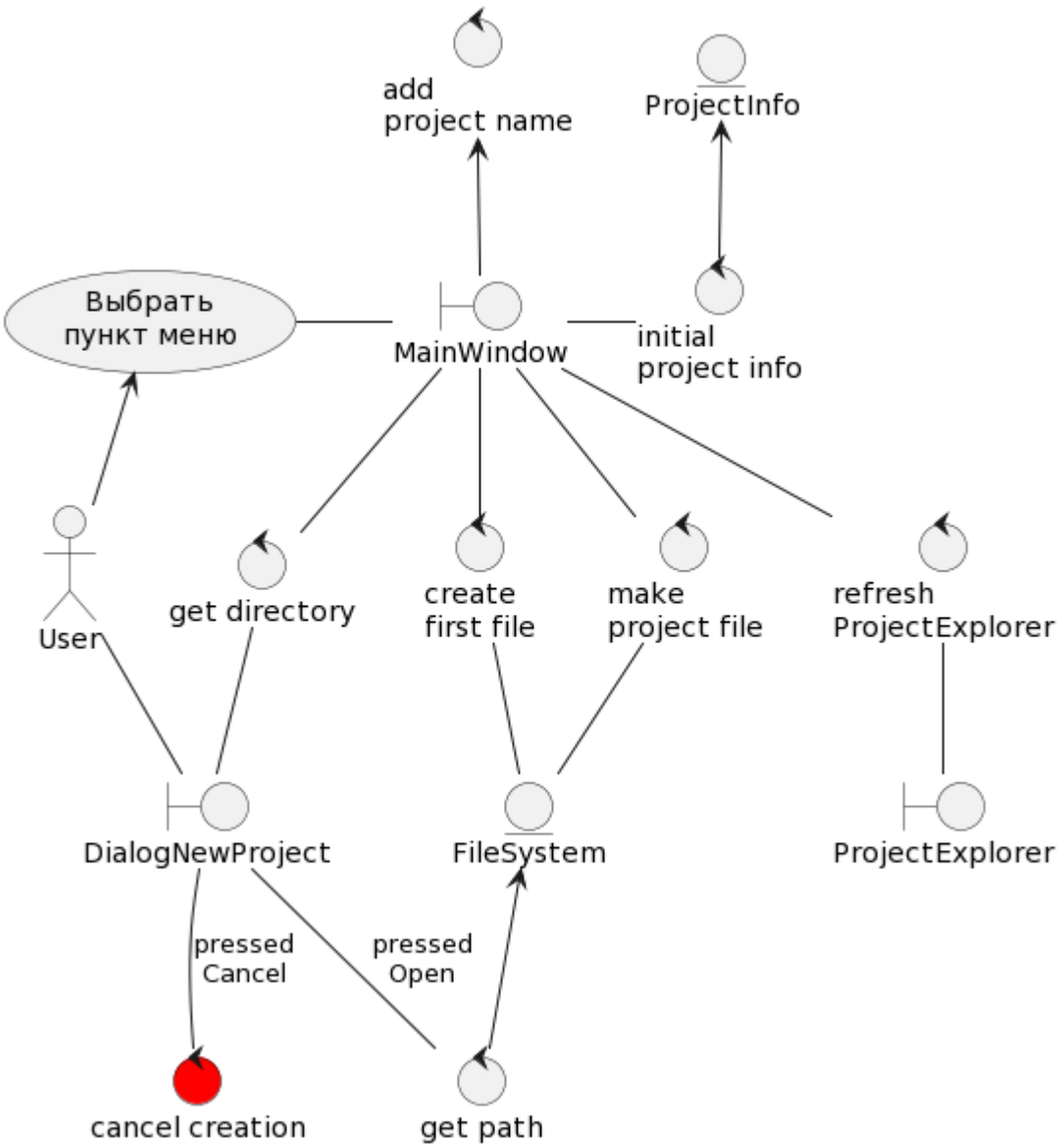

Рисунок 2.22 — Диаграмма пригодности для прецедента «Создать проект»

После выполнения пользователем прецедента «Выбрать пункт меню» главное окно создаёт экземпляр диалогового окна и запрашивает у него путь к директории. Пользователь выбирает директорию в диалоговом окне или отменяет создание проекта. Главное окно, получая путь к директории, создаёт в ней начальный файл в проекте и файл проекта (файл проекта заполняет начальной информацией о проекте). Дальше главное окно записывает начальные данные в структуру «Информация о проекте», добавляет название проекта к наименованию окна и обновляет список файлов проекта в обозревателе проекта.

Диаграмма пригодности для прецедента «Открыть проект» представлена на рисунке 2.23.

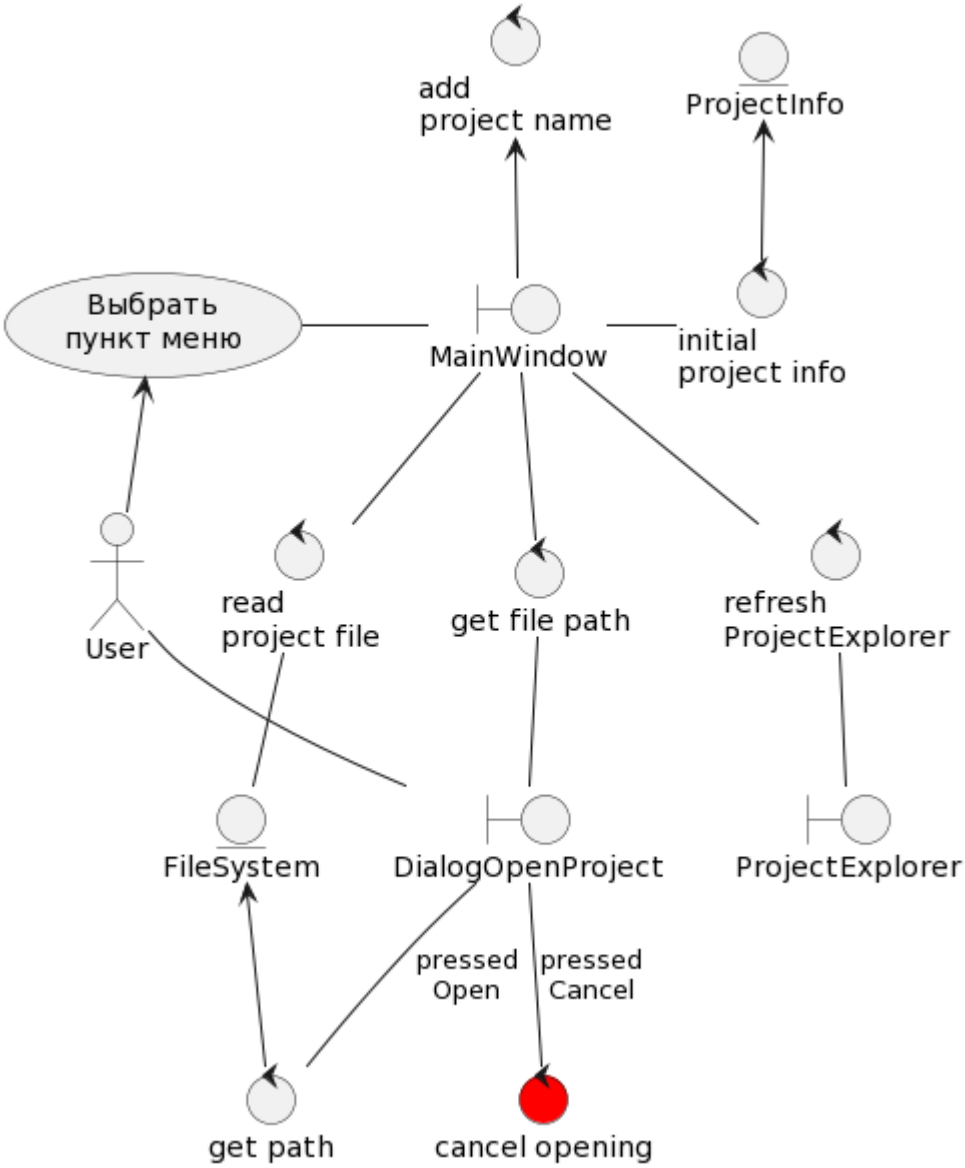

Рисунок 2.23 — Диаграмма пригодности для прецедента «Открыть проект»

После выполнения пользователем прецедента «Выбрать пункт меню» главное окно создаёт экземпляр диалогового окна и запрашивает у него путь к файлу. Пользователь выбирает файл в диалоговом окне или отменяет открытие проекта. Главное окно, получая путь к файлу, считывает из файла данные в структуру «Информация о проекте», добавляет название проекта к

наименованию окна и обновляет список файлов проекта в обозревателе проекта.

2.5 Диаграммы последовательности

На диаграммах последовательности описываются отношения объектов во времени. Они более приближённо иллюстрируют работу разрабатываемого приложения.

Диаграммы последовательности для каждого прецедента приведены на рисунках 2.24–2.44.

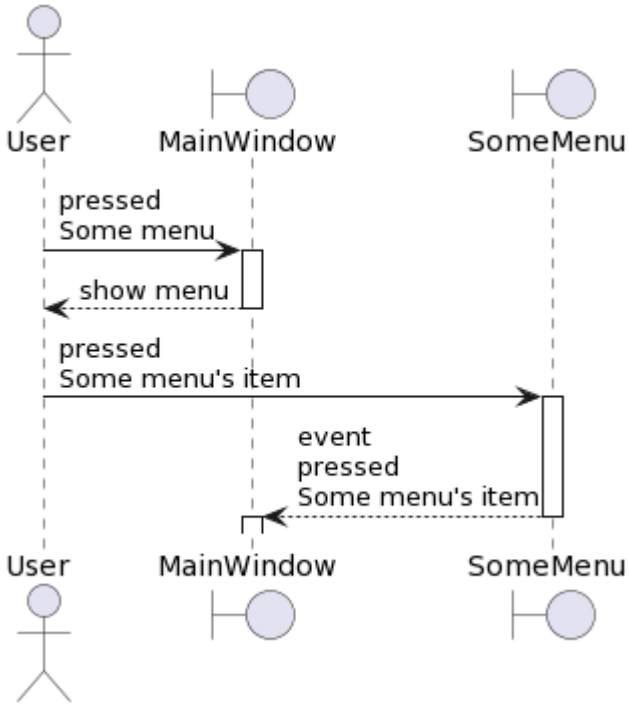

Рисунок 2.24 — Диаграмма последовательности для прецедента «Выбрать пункт меню»

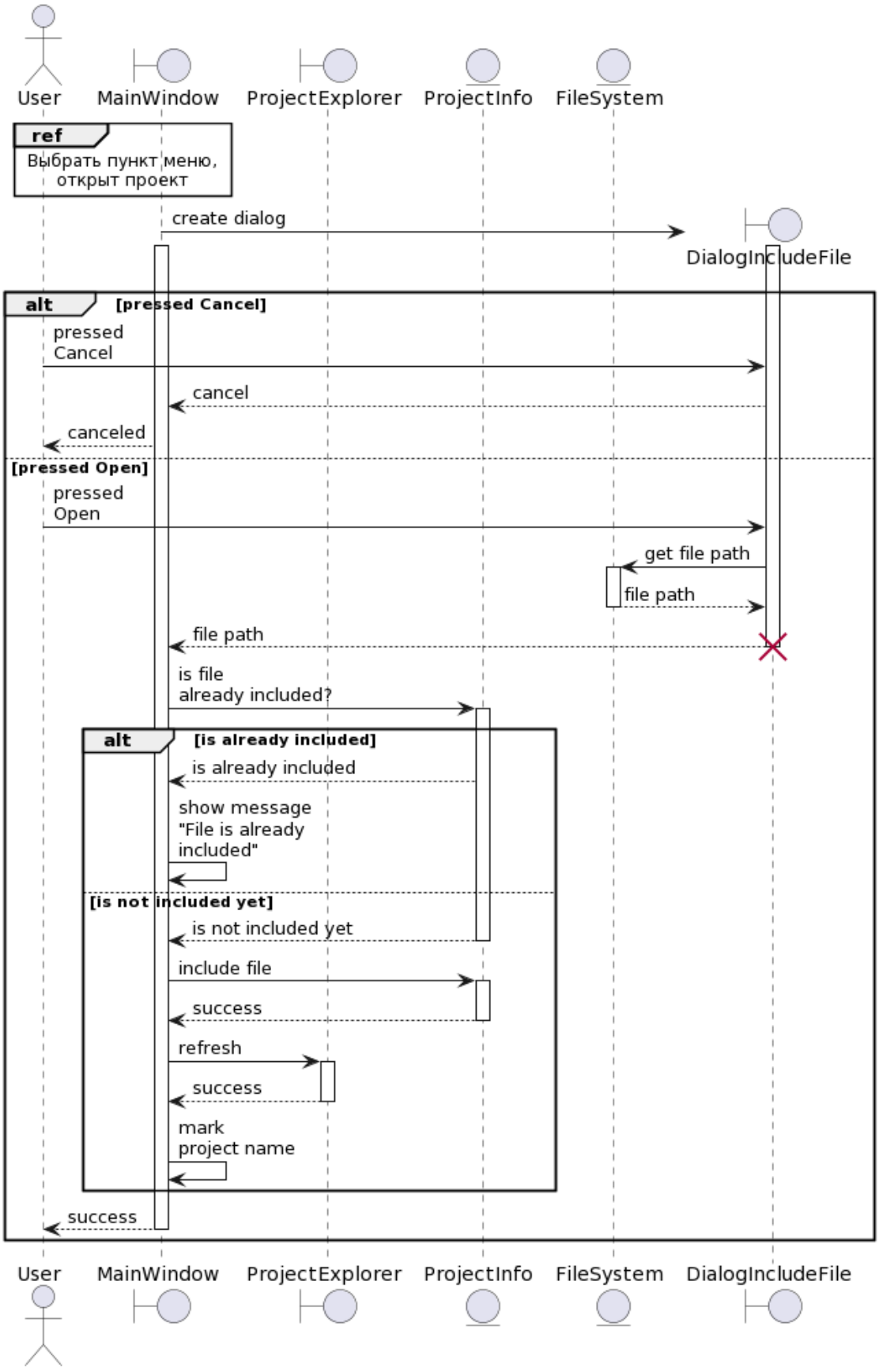

Рисунок 2.25 — Диаграмма последовательности для прецедента «Подключить файл»

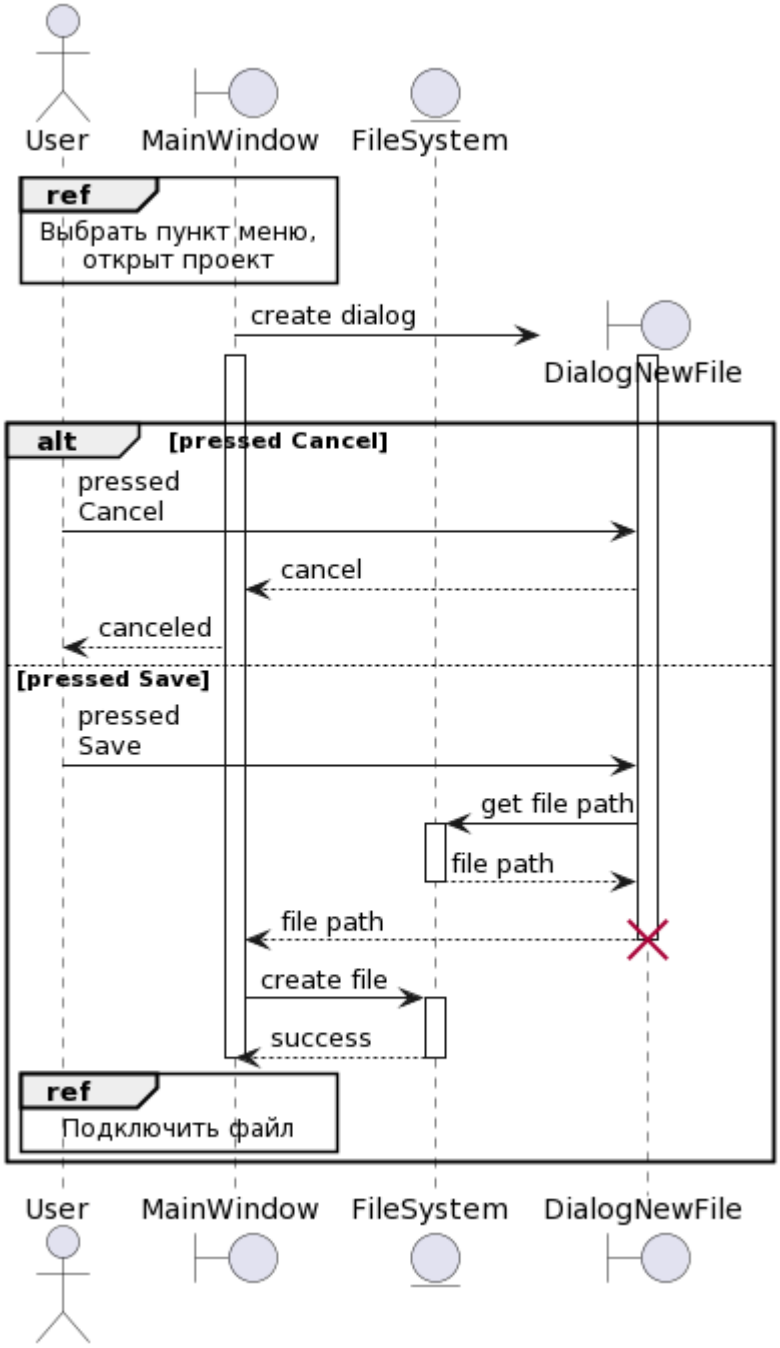

Рисунок 2.26 — Диаграмма последовательности для прецедента «Создать файл»

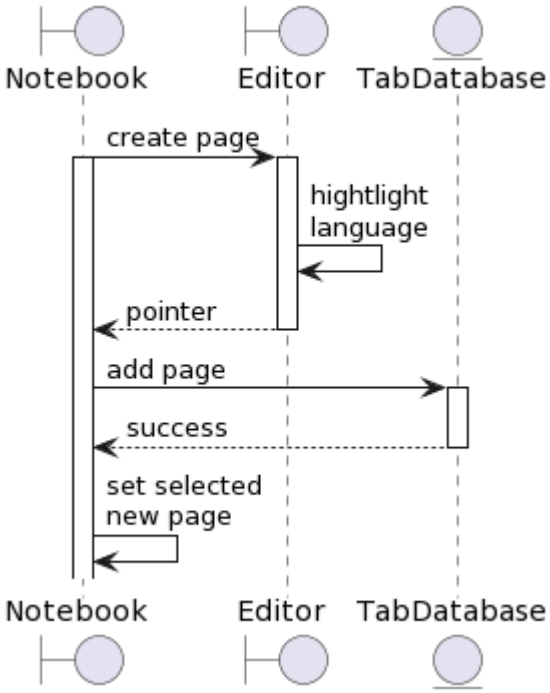

Рисунок 2.27 — Диаграмма последовательности для прецедента «Создать вкладку»

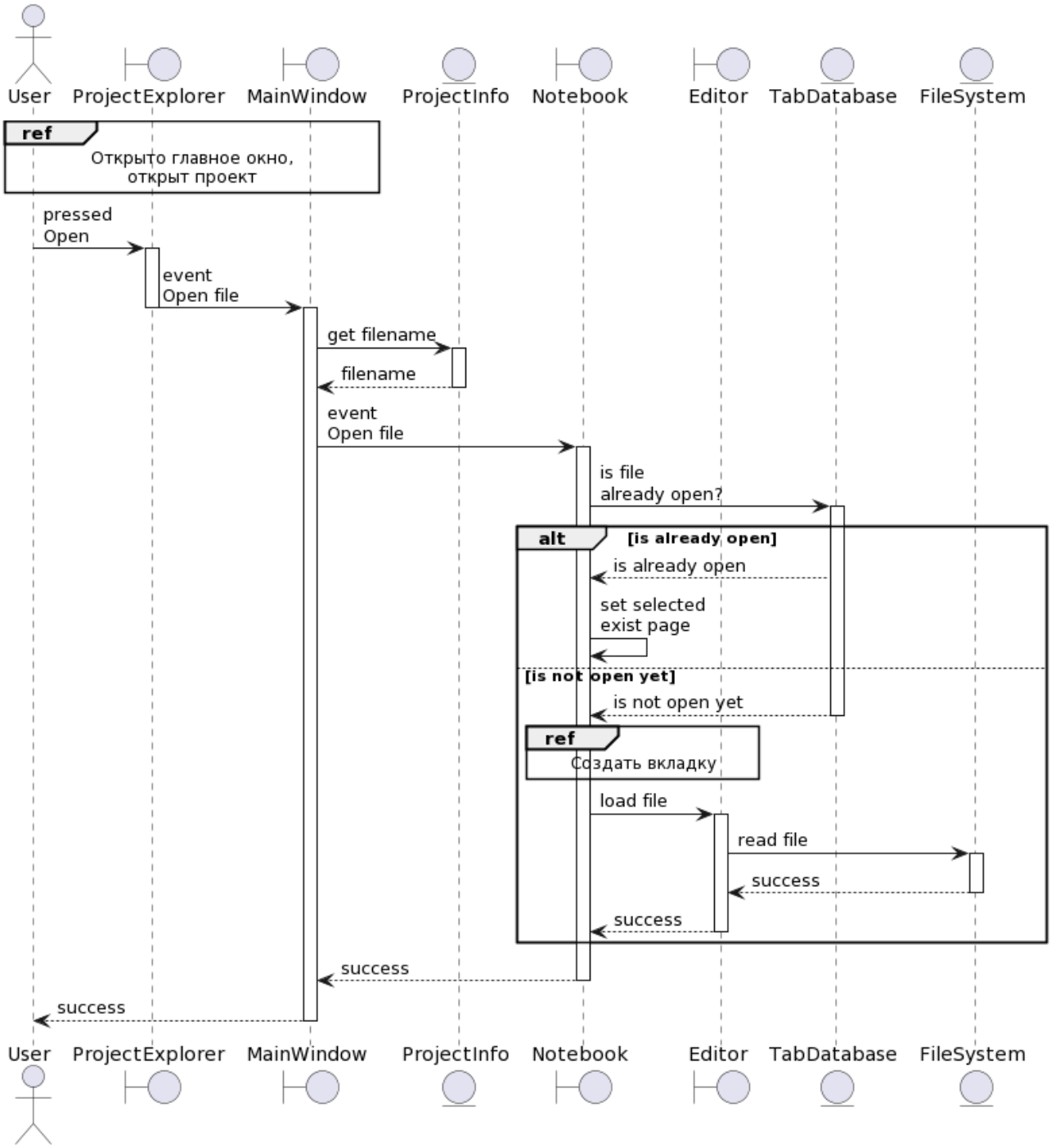

Рисунок 2.28 — Диаграмма последовательности для прецедента «Открыть файл»

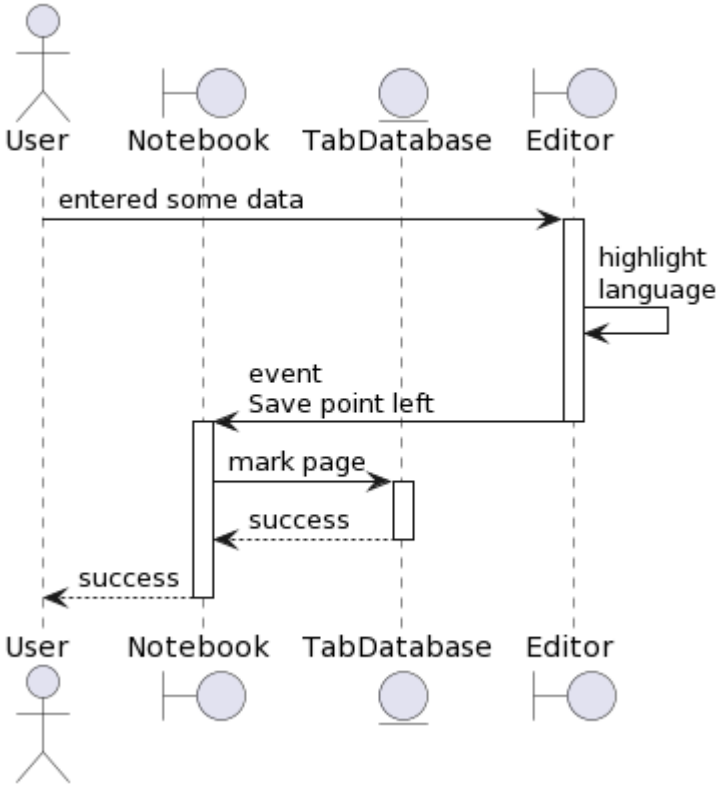

Рисунок 2.29 — Диаграмма последовательности для прецедента «Редактировать файл»

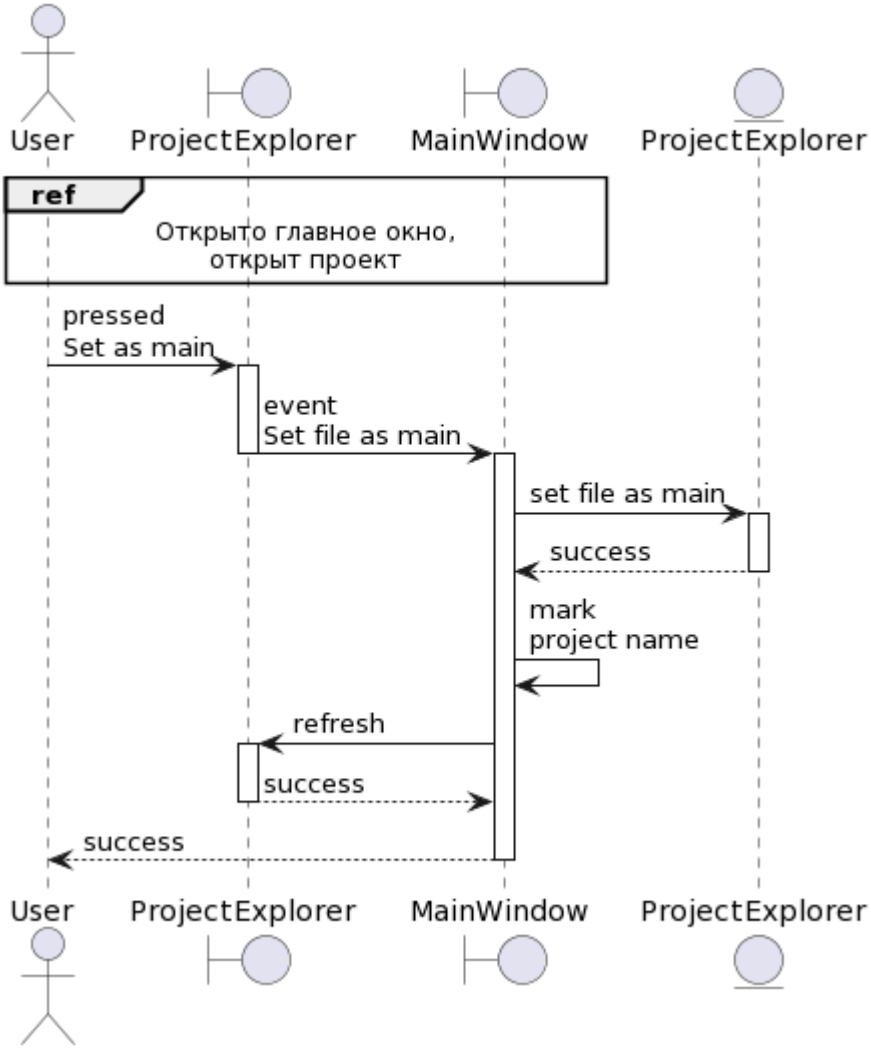

Рисунок 2.30 — Диаграмма последовательности для прецедента «Назначить файл главным в проекте»

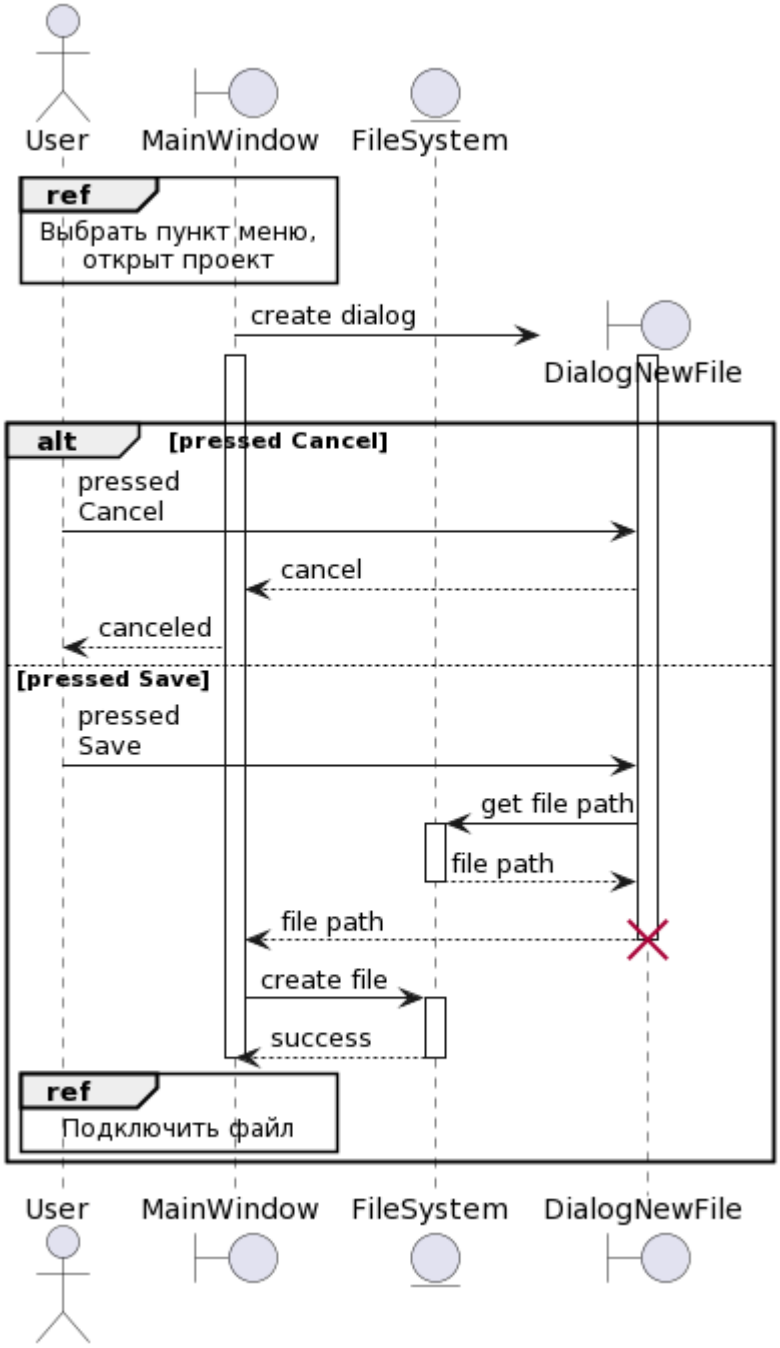

Рисунок 2.31 — Диаграмма последовательности для прецедента «Сохранить файл»

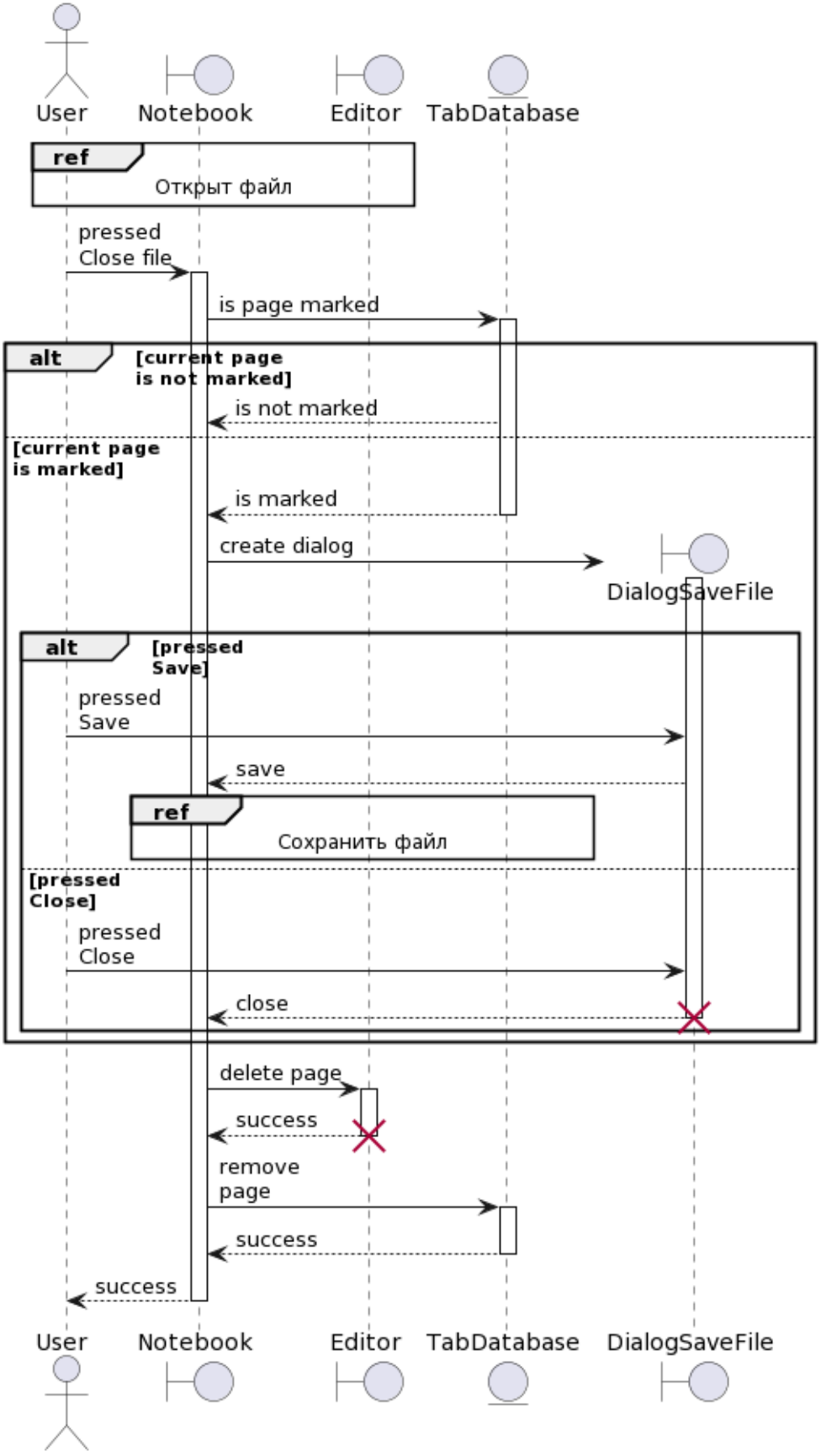

Рисунок 2.32 — Диаграмма последовательности для прецедента «Закрыть файл»

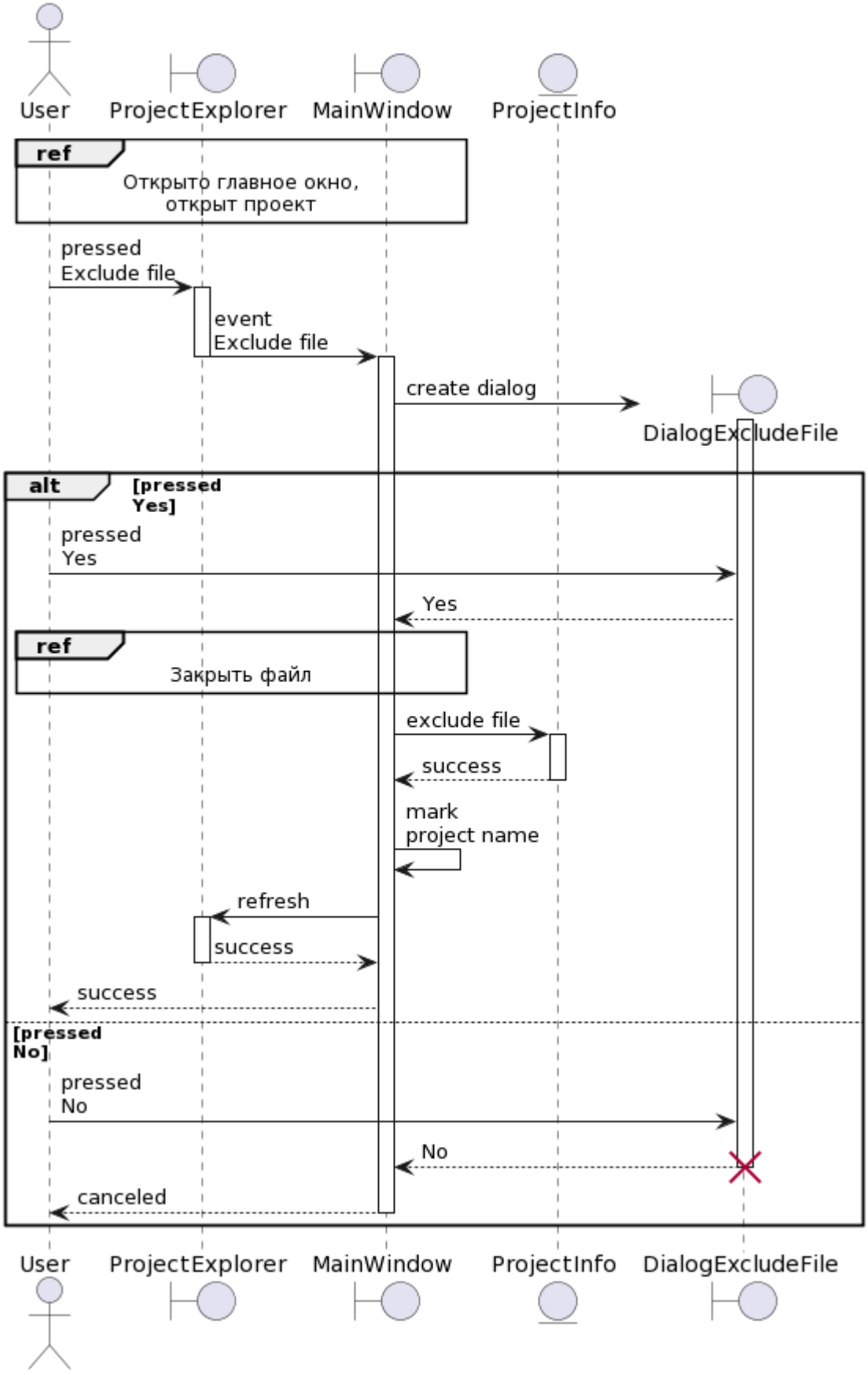

Рисунок 2.33 — Диаграмма последовательности для прецедента «Исключить файл»

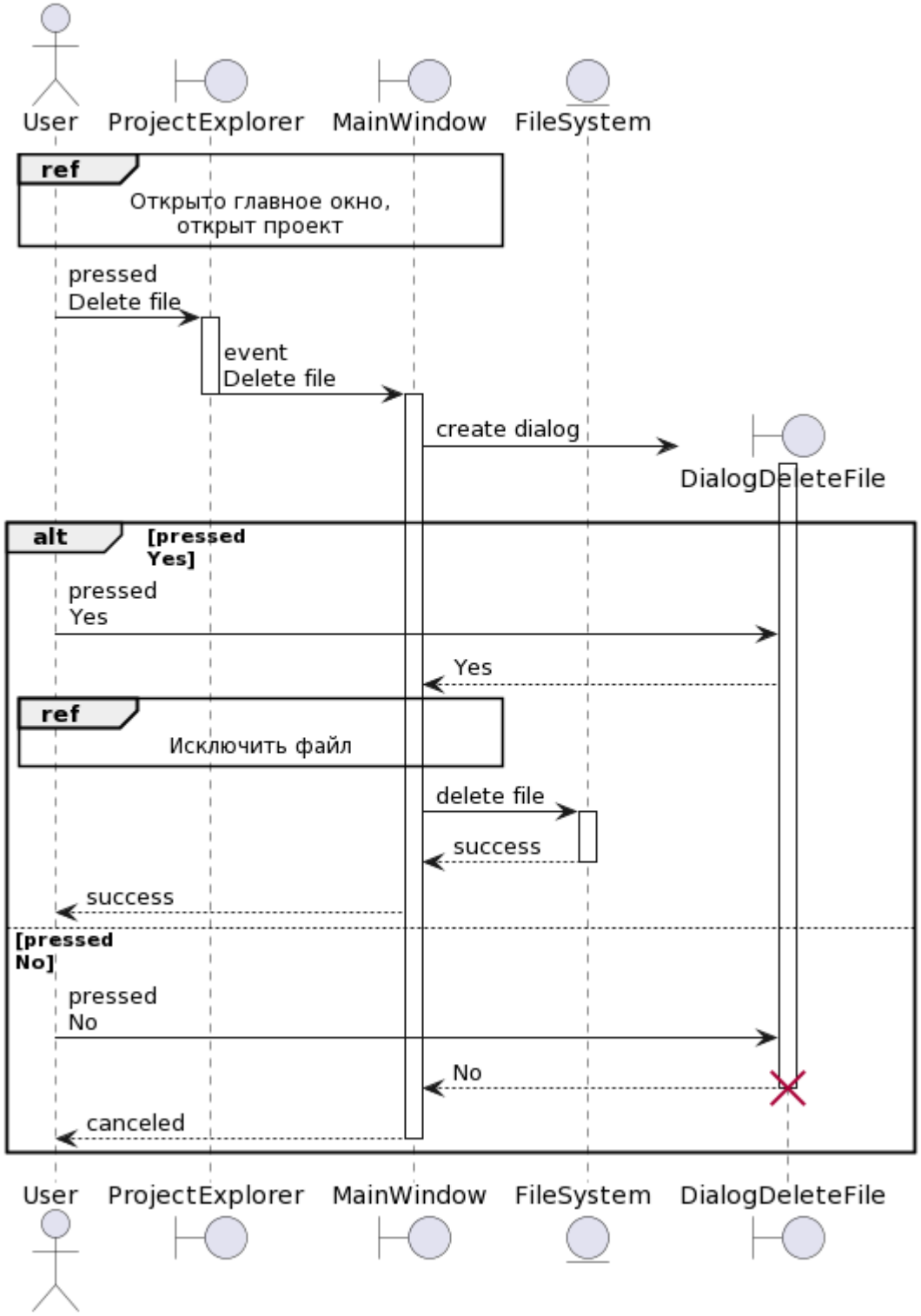

Рисунок 2.34 — Диаграмма последовательности для прецедента «Удалить файл»

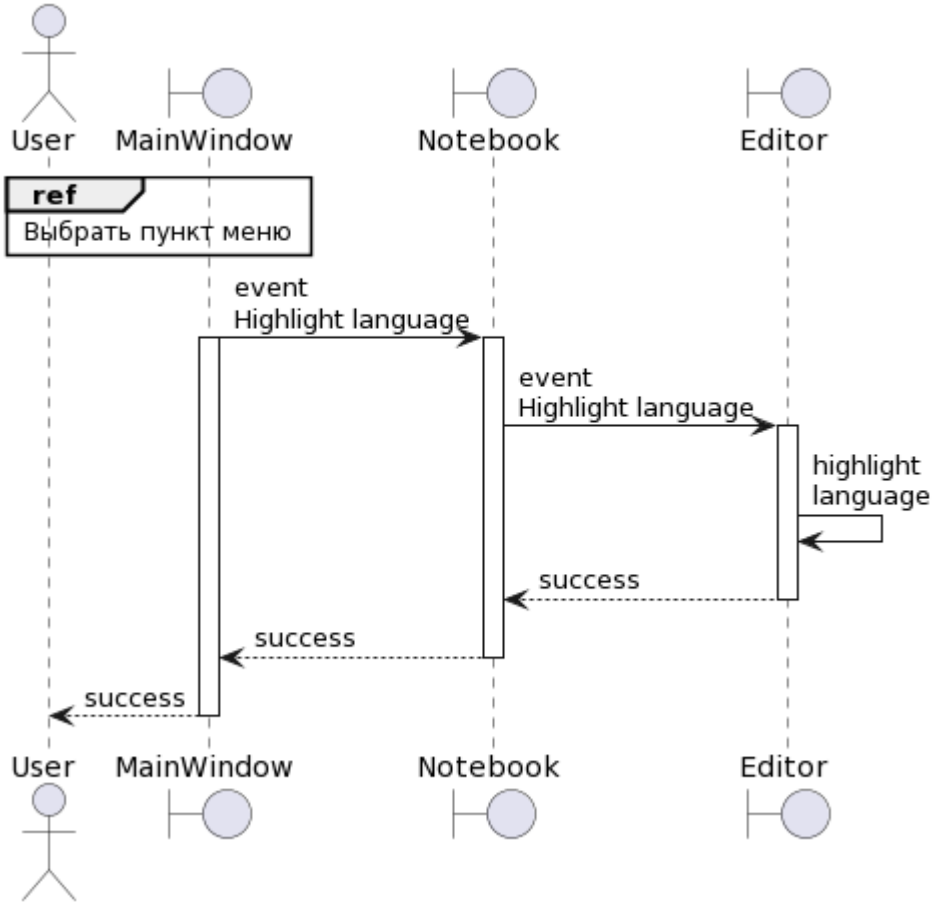

Рисунок 2.35 — Диаграмма последовательности для прецедента «Выбрать язык»

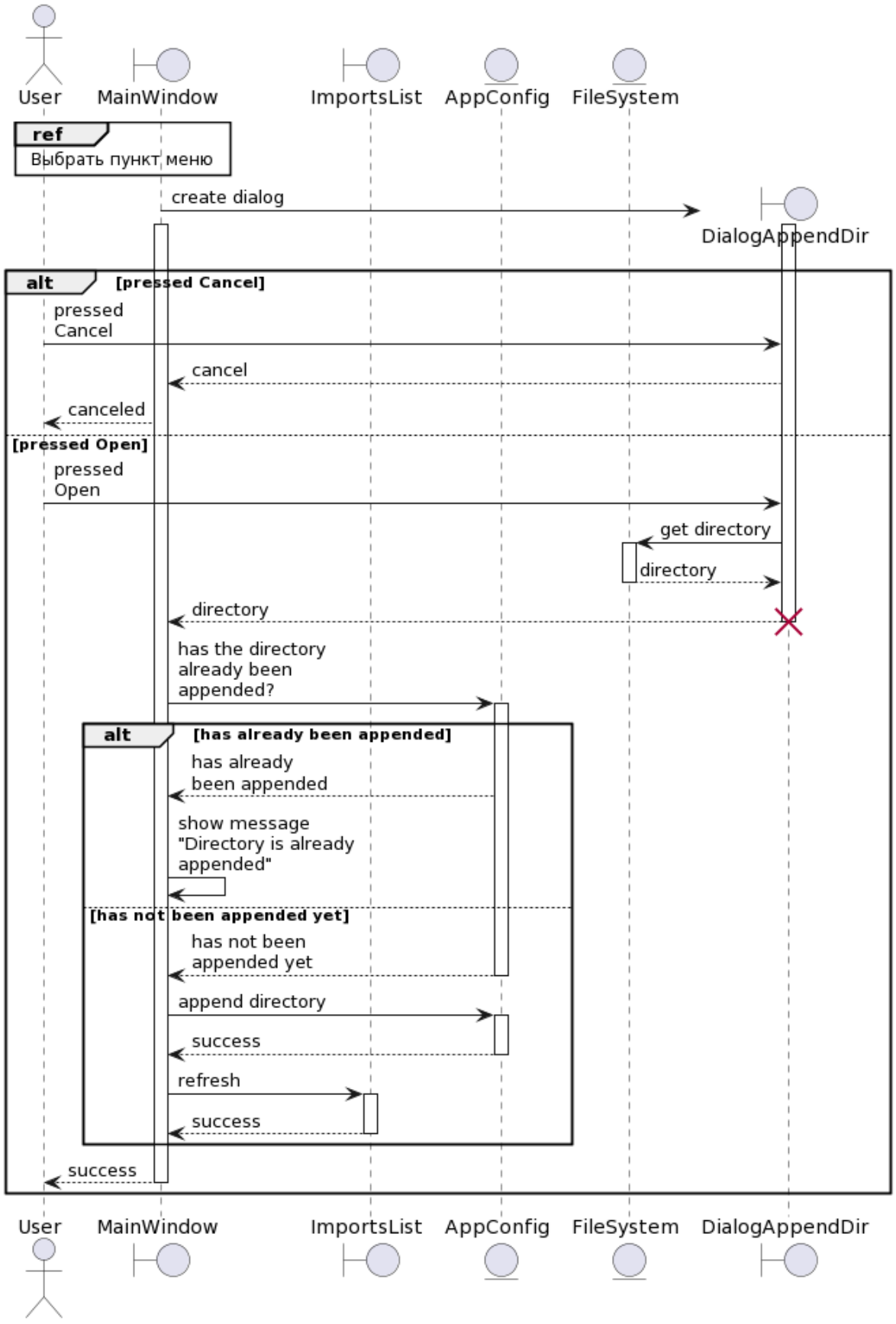

Рисунок 2.36 — Диаграмма последовательности для прецедента «Добавить путь к модулям»

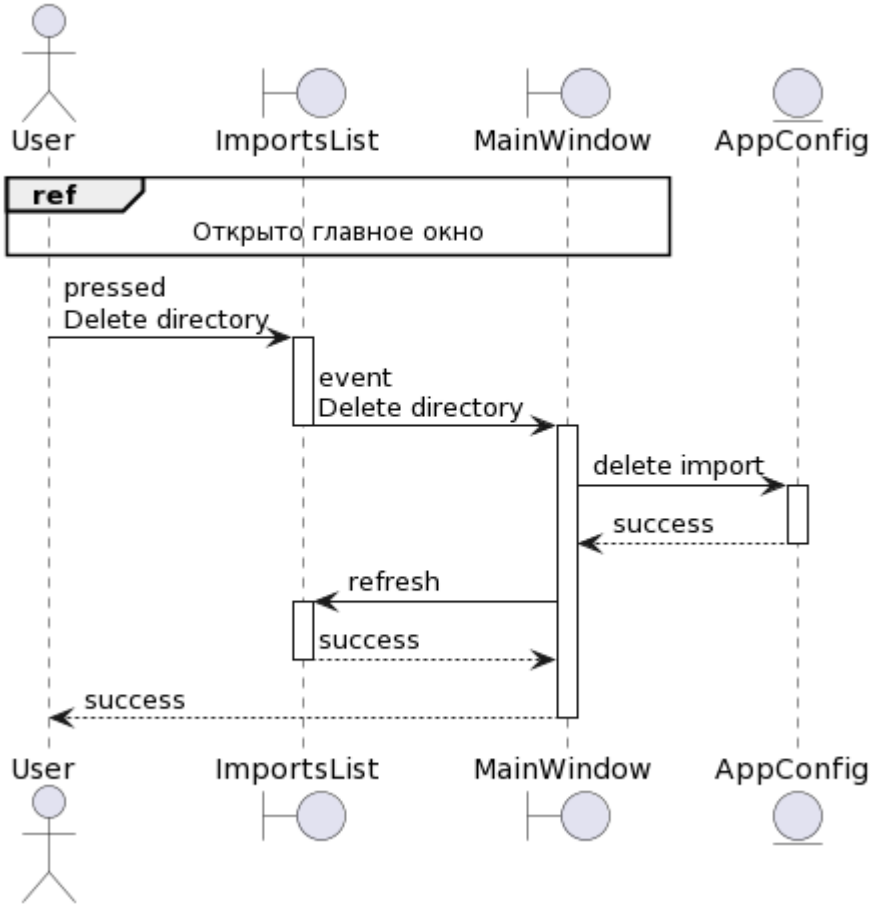

Рисунок 2.37 — Диаграмма последовательности для прецедента «Удалить путь к модулям»

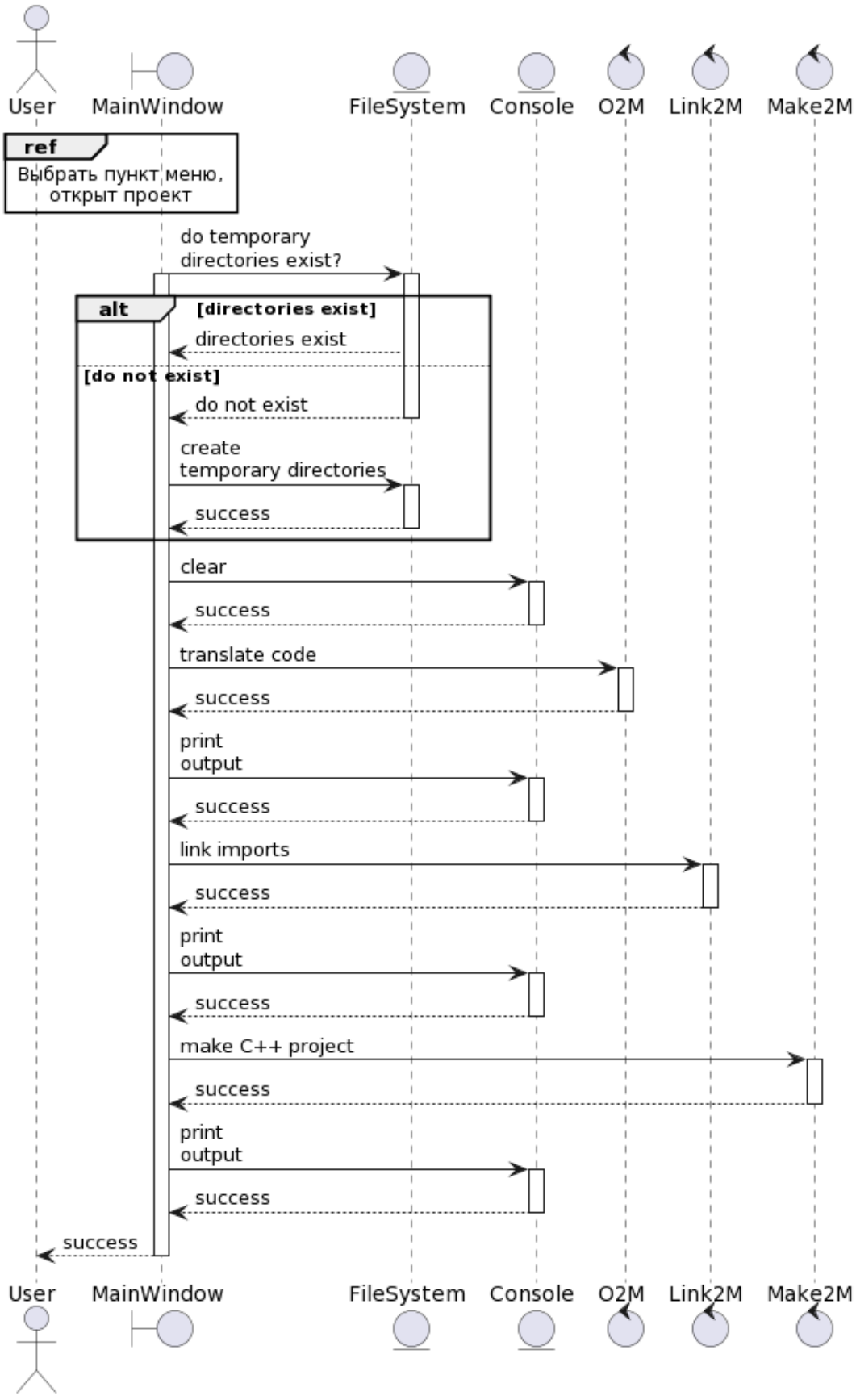

Рисунок 2.38 — Диаграмма последовательности для прецедента «Скомпилировать код проекта»

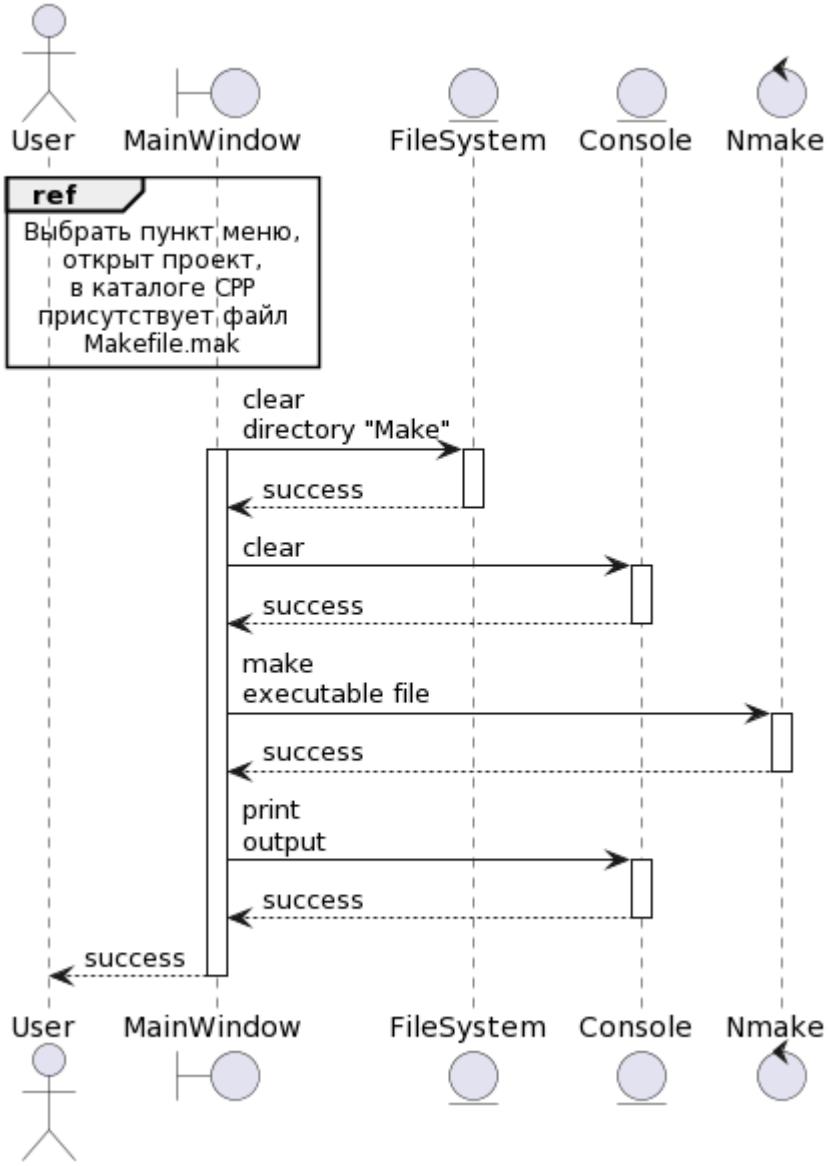

Рисунок 2.39 — Диаграмма последовательности для прецедента «Собрать проект»

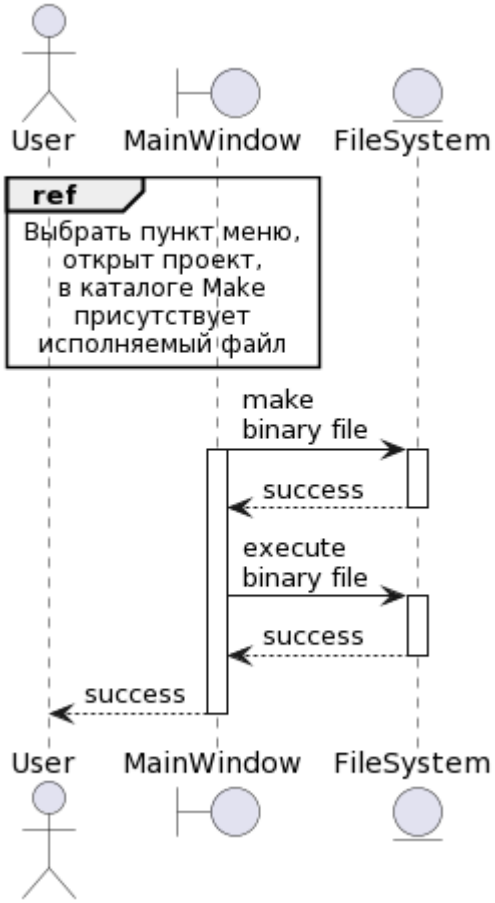

Рисунок 2.40 — Диаграмма последовательности для прецедента «Запустить проект»

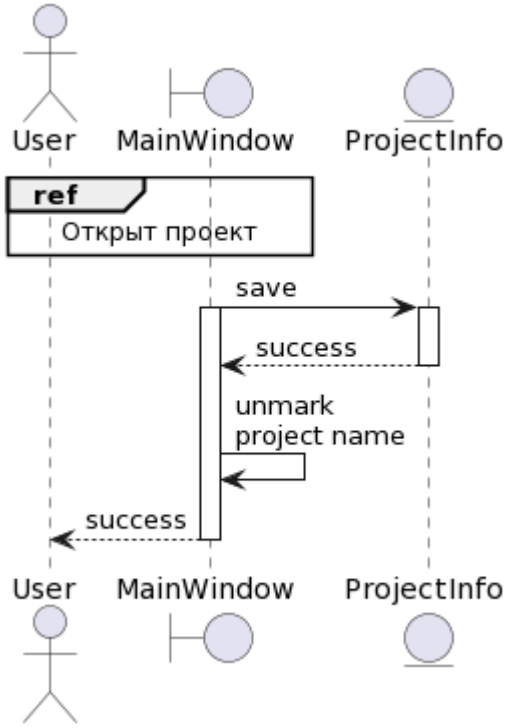

Рисунок 2.41 — Диаграмма последовательности для прецедента «Сохранить проект»

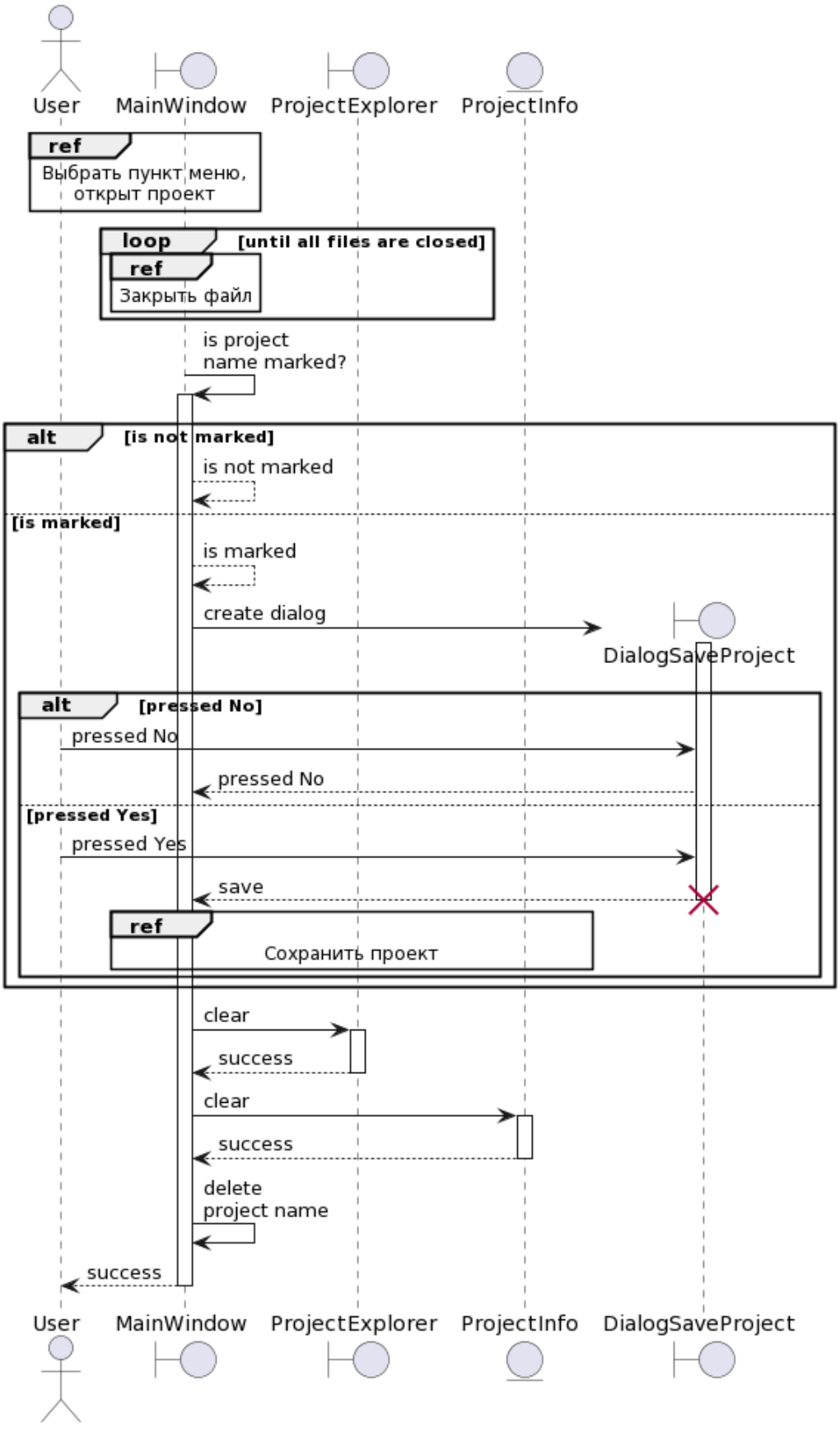

Рисунок 2.42 — Диаграмма последовательности для прецедента «Закрыть проект»

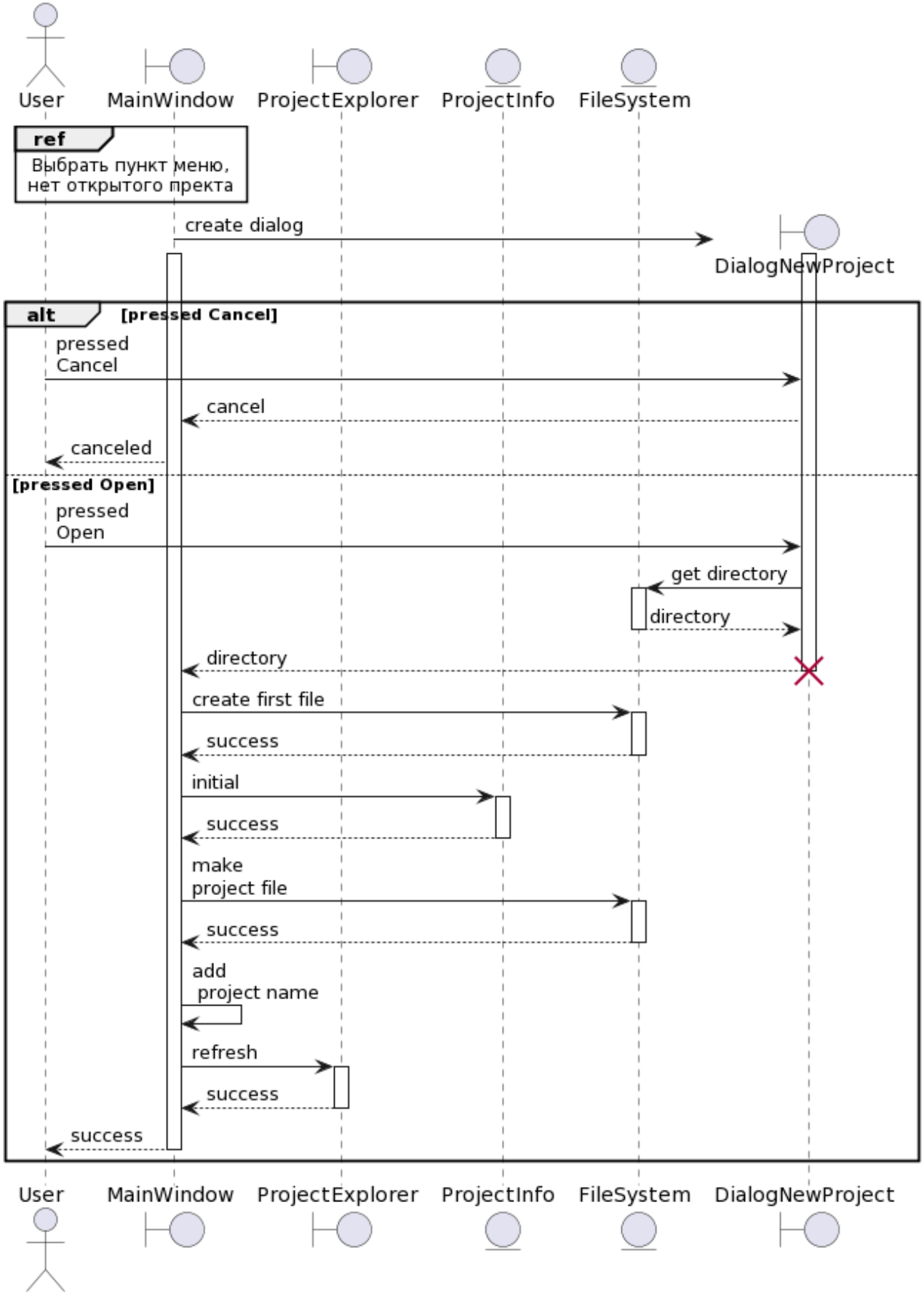

Рисунок 2.43 — Диаграмма последовательности для прецедента «Создать проект»

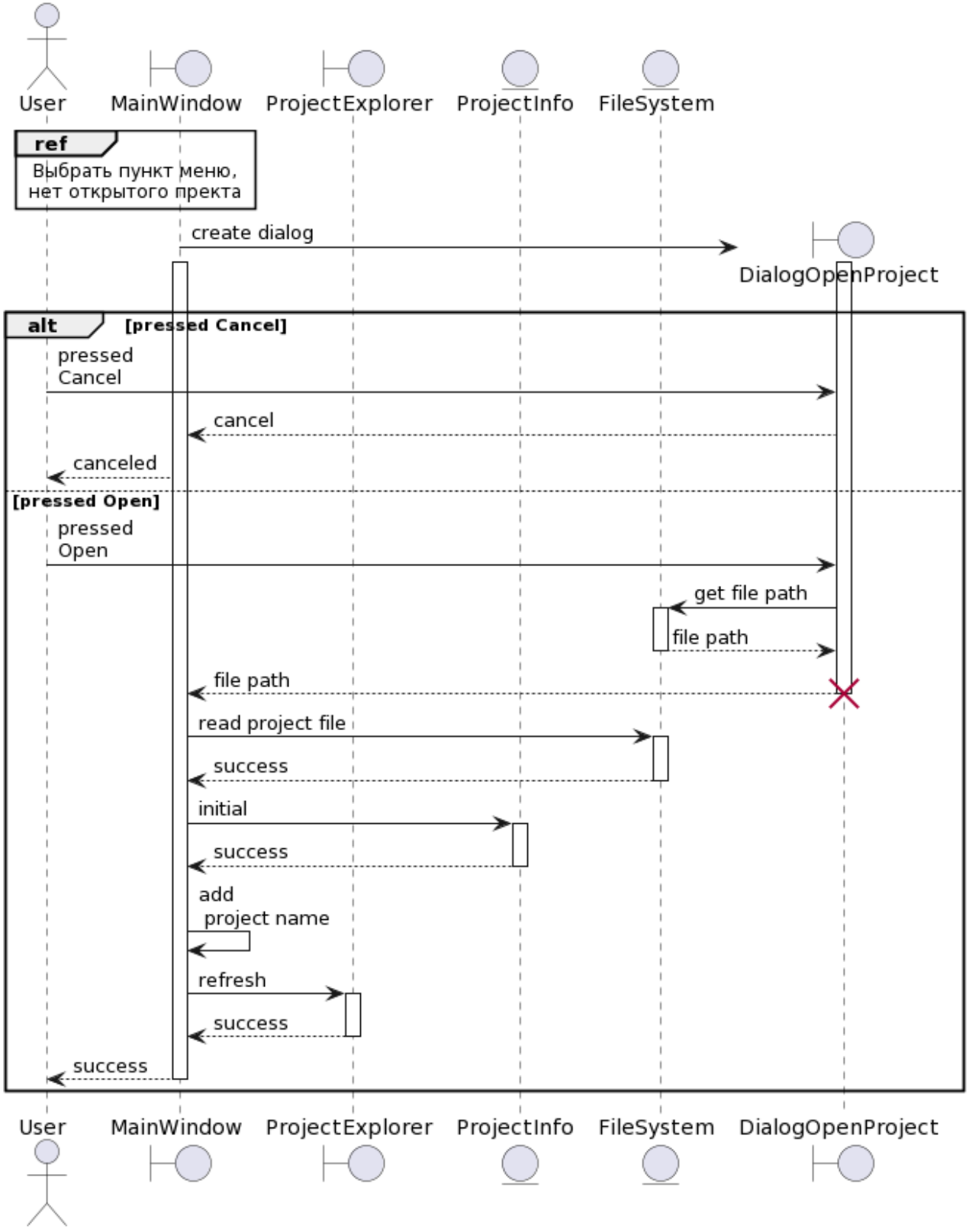

Рисунок 2.44 — Диаграмма последовательности для прецедента «Открыть проект»

2.6 Диаграмма классов

Диаграмма классов уровня проектирования представлена на рисунке 2.45.

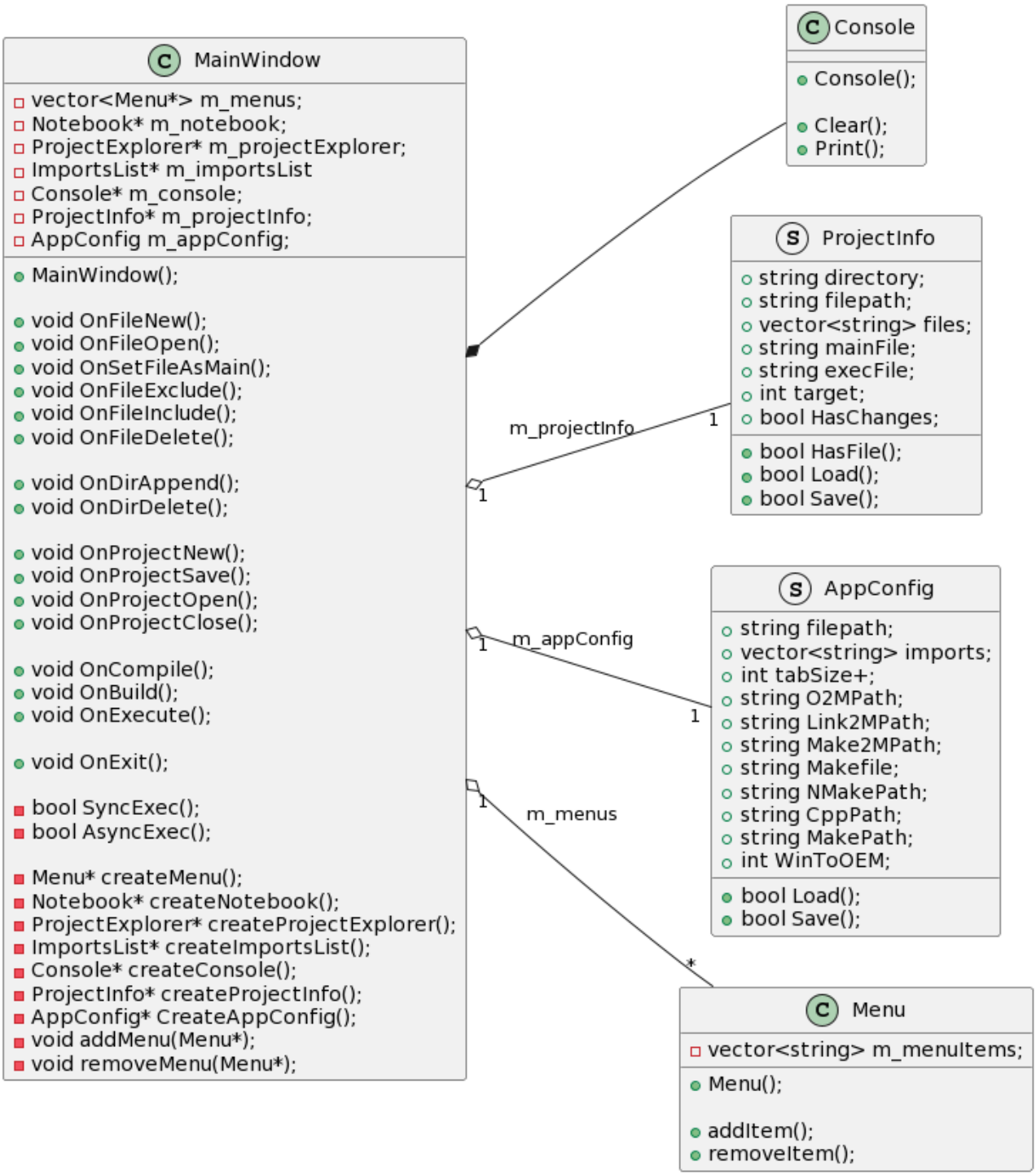

Рисунок 2.45 — Диаграмма классов уровня проектирования, лист 1

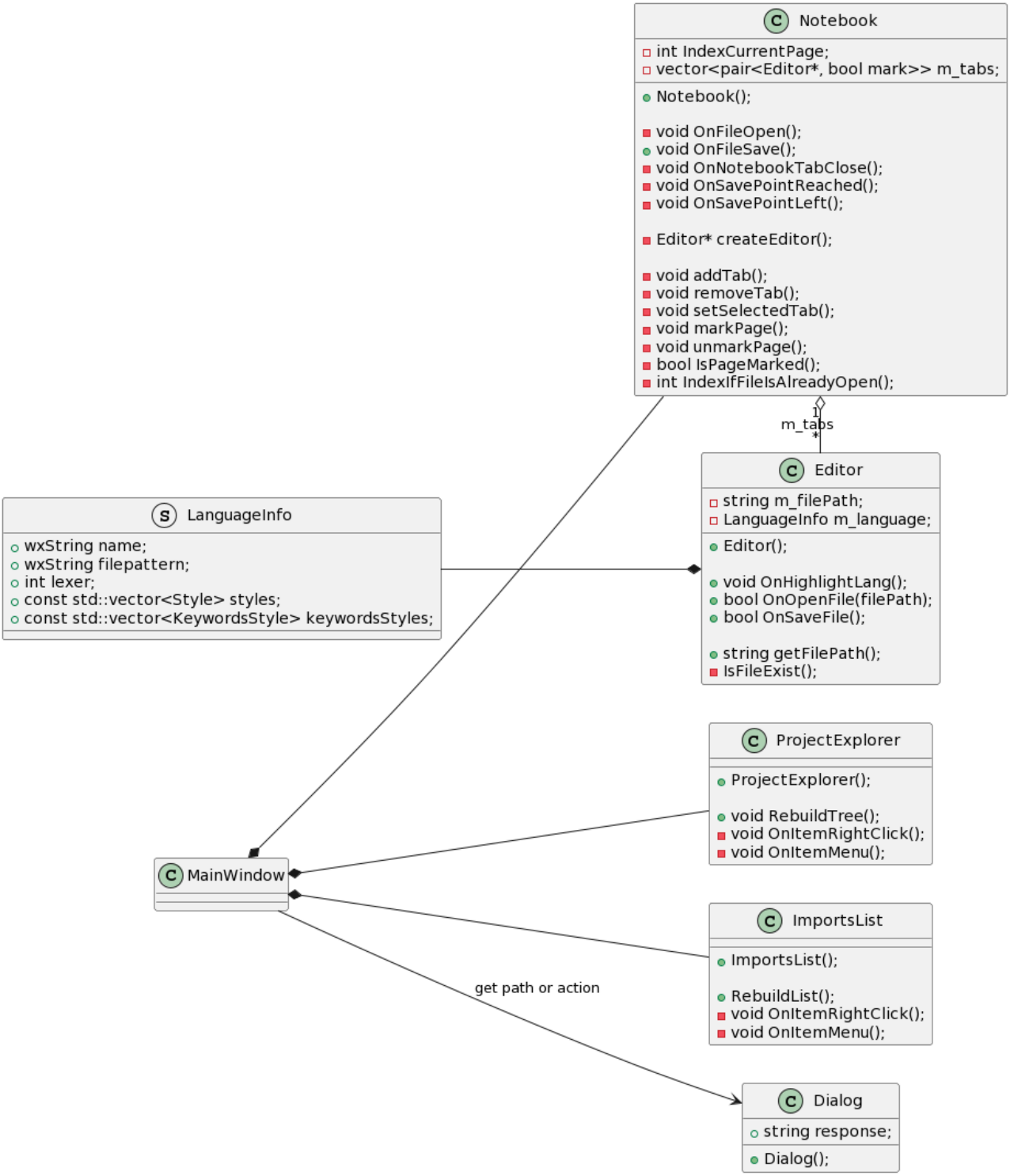

Рисунок 2.45, лист 2

2.7 Формат файлов, создаваемых приложением

При первом запуске интегрированной среды разработки в директории, где находится её исполняемый файл, создаётся файл «*CodeEditor.ini*» для сохранения настроек. Он имеет следующий формат:

– строки вида: «*IMPORT*=<полный путь к директории с закрывающим обратным слешем>»;

– строка вида: «*TABSIZE*=<размер символа табуляции (количество пробелов)>». По умолчанию установлено «4»;

– строка вида: «*O2M*=<полный путь к транслятору *O2M*>»;

– строка вида: «*Link2M*=<полный путь к компоновщику *Link2M*>»;

– строка вида: «*Make2M*=<полный путь к генератору проекта для *C++ Make2M*>»;

– строка вида: «*MAKEFILE*=<название файла>*.mak*»;

– строка вида: «*NMAKE*=<полный путь к утилите *nmake*>»;

– строка вида: «*CPP*=<путь к компилятору *C++*>»;

– строка вида: «*MAKE*=<полный путь к утилите *Make*>»;

– строка вида: «*WINTOOEM*=<признак необходимости перекодировки имен файлов из *Win1251* в *OEM Cyrill*>». По умолчание установлено «0».

При создании нового проекта создаётся файл проекта «*NewProject.pro*». В нём хранится информация о проекте. Он имеет следующий формат:

– строка вида: «*MAIN*=<название главного файла в проекте>». По умолчанию «*main.o*»;

– строка вида: «*EXEC*=<название исполняемого файла>». По умолчанию «*NewProject.exe*»;

– строки вида: «*FILE*=<название файла>»;

– строка вида: «*TARGET*=<число, обозначающее выбранную целевую платформу>». По умолчанию «0».

2.8 Выводы по разделу 2

Для проектирования был выбран процесс *ICONIX*. В ходе выполнения его этапов:

– построена диаграмма вариантов использования и сделано текстовое описание прецедентов;

– опираясь на описание прецедентов, построены диаграммы пригодности (робастости) для каждого прецедента;

– с помощью диаграмм пригодности построены диаграммы последовательности для каждого прецедента;

– основываясь на предыдущих этапах проектирования, построена диаграмма классов уровня проектирования.

Также, описан формат файла проекта и файла настроек, создаваемых редактором кода.

3 Разработка редактора кода

3.1 Выбор инструментов

Для реализации приложения были выбраны интегрированная среда разработки «*MS Visual Studio 2022*» и библиотека «*wxWidgets*» версии 3.2.2.1 на языке *C++*. Её ближайшим аналогом является «*Qt Framework*», но по сравнению с ним у «*wxWidgets*» есть ряд преимуществ:

– распространяется под лицензией, которая позволяет использовать библиотеку в проектах с открытым и закрытым исходным кодом без ограничений;

– использует стандартные окна платформы для создания пользовательского интерфейса;

– доступна разработчикам на территории России и не имеет намерения приостановить свою работу в стране.

3.2 Реализация компонентов

3.2.1 Структура данных настроек приложения

Данный компонент реализован с помощью структуры «*CodeEditorConfig*», которая представлена на рисунке 3.1.

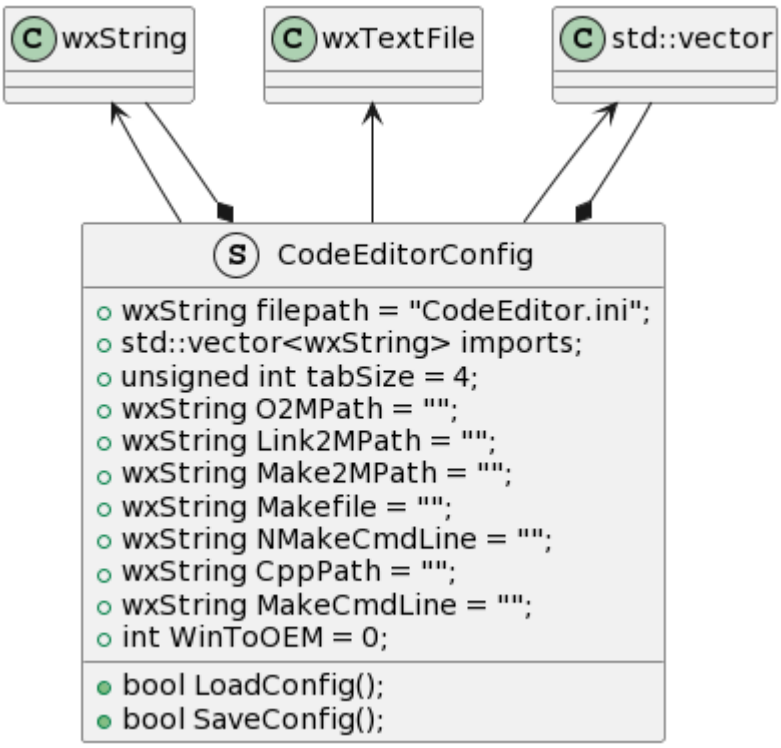

Рисунок 3.1 — Часть диаграммы классов со структурой «*CodeEditorConfig*»

Назначение полей структуры связано с форматом файла настроек приложения, который описан в пункте 2.7 данного документа.

Метод «*LoadConfig*» реализован следующим образом:

– с помощью методов класса «*wxTextFile*» проверяется существование файла *filepath*, и, если файл существует, он открывается и построчно считывается;

– с помощью методов класса «*wxString*» строка разбивается на две строки (ключ и значение) и, в зависимости от значения ключа, значение записывается в необходимое поле структуры;

– поле «*imports*» заполняется с помощью методов класса «*std::vector*»;

– после окончания считывания файл закрывается.

Метод «*SaveConfig*» реализован следующим образом:

– с помощью методов класса «*wxTextFile*» проверяется существование файла *filepath*;

– если файл не существует, то он создаётся;

– файл открывается и в него записываются строки, описанные в пункте 2.7 данного документа;

– после окончания записи файл закрывается.

3.2.2 Структура данных информации о проекте

Данный компонент реализован с помощью структуры «*ProjectInfo*», которая представлена на рисунке 3.2.

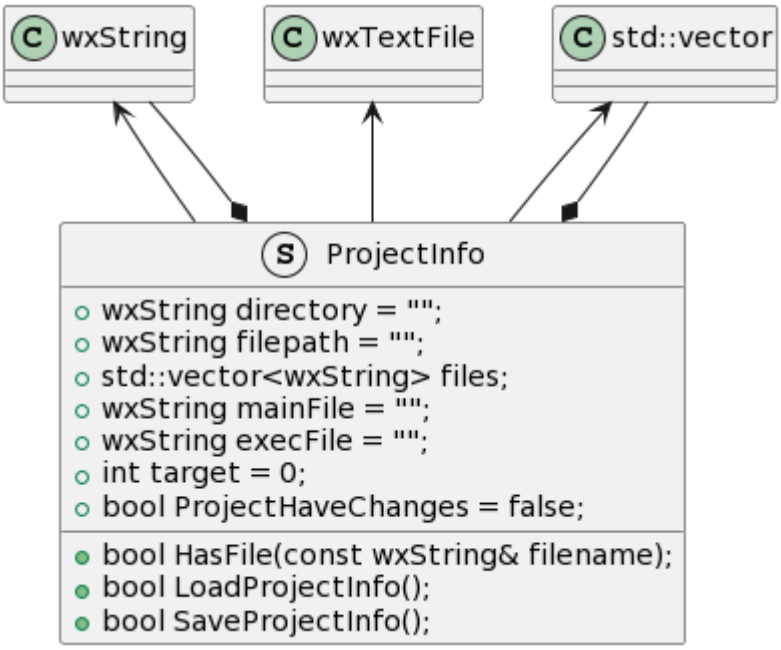

Рисунок 3.2 — Часть диаграммы классов со структурой «*ProjectInfo*»

Назначение полей структуры связано с форматом файла проекта, который описан в пункте 2.7 данного документа. Поле *ProjectHaveChanges* хранит состояние проекта (имеет изменения или нет).

Метод «*HasFile*» реализован следующим образом:

– с помощью методов класса «*std::vector*» проверяется наличие *filename* в поле *files*;

Метод «*LoadProjectInfo*» реализован следующим образом:

– с помощью методов класса «*wxTextFile*» файл *filepath* открывается и построчно считывается;

– с помощью методов класса «*wxString*» строка разбивается на две строки (ключ и значение) и, в зависимости от значения ключа, значение записывается в необходимое поле структуры;

– поле «*files*» заполняется с помощью методов класса «*std::vector*»;

– после окончания считывания файл закрывается.

Метод «*SaveProjectInfo*» реализован следующим образом:

– с помощью методов класса «*wxTextFile*» файл *filepath* открывается и в него записываются строки, описанные в пункте 2.7 данного документа;

– после окончания записи файл закрывается.

3.2.3 Главное окно

Данный компонент реализован с помощью класса «*MainWindow*», который представлен на рисунке 3.3.

Рисунок 3.3 — Упрощённая иллюстрация содержимого класса «*MainWindow*»

Поля класса хранят указатели на дочерние виджеты и на структуры данных.

Метод «*MainWindow*» реализован следующим образом:

– создаётся объект класса «*wxAuiManager*», который управляет расположением виджетов;

– в поле *m_NotebookStyle* сохраняется стиль виджета «Блокнот»;

– вызываются методы *CreateNotebook*, *CreateProjectExplorer*, *CreateImportList*, *CreateConsole*, *CreateMenuBar*, а результаты их вызовов сохраняются в соответствующие поля класса;

– созданные в предыдущем пункте дочерние элементы главного окна добавляются в окно с помощью методов объекта *m_AuiManager*;

– к событиям привязываются обработчики;

– из файла загружаются настройки приложения и заполняется виджет «Список подключаемых директорий» (если файла не было, то он создаётся, а приложение показывает сообщение, в котором просит заполнить файл).

Метод «*~MainWindow*» реализован следующим образом:

– удаляются все дочерние виджеты;

– сохраняются настройки приложения;

– удаляется объект *m_AuiManager*.

Метод «*OnProjectNew*» реализован следующим образом:

– производится проверка, открыт ли сейчас проект (если нет, то создаётся объект класса *ProjectInfo*);

– создаётся диалог, в котором пользователь выбирает директорию для проекта;

– производится проверка, существует ли в этой директории файл проекта (если нет, то в ней создаётся файл проекта и начальный файл с помощью метода «*CreateNewFile*»);

– заполняется начальными данными объект с информацией о проекте;

– вызывается метод объекта *m_ProjectExplorer* для построения дерева проекта.

Метод «*OnProjectSave*» реализован следующим образом:

– вызывается метод объекта *m_projectInfo* для сохранения информации о проекте;

– вызывается метод «*SetProjectState*».

Метод «*OnProjectOpen*» реализован следующим образом:

– производится проверка, открыт ли сейчас проект (если нет, то создаётся объект класса *ProjectInfo*);

– создаётся диалог, в котором пользователь выбирает файл проекта;

– из файла считываются данные в объект с информацией о проекте;

– вызывается метод «*SetProjectState*»;

– вызывается метод объекта *m_ProjectExplorer* для построения дерева проекта.

Метод «*OnProjectClose*» реализован следующим образом:

– в цикле по вкладкам создаётся событие закрытия вкладки и передаётся в объект *m_Notebook*;

– вызывается метод *OnProjectSave* (или нет, если пользователь откажется сохранять изменения в проекте);

– удаляется объект *m_projectInfo*;

– вызывается метод «*SetProjectState*»;

– вызывается метод объекта *m_ProjectExplorer* для очистки виджета.

Метод «*OnDirAppend*» реализован следующим образом:

– создаётся диалог, в котором пользователь выбирает директорию для подключения;

– производится проверка, присутствует ли в объекте *m_config* выбранная директория;

– если не присутствует, то добавляется в поле *imports* объекта *m_config*;

– вызывается метод объекта *m_ImportList* для обновления содержимого виджета.

Метод «*OnDirUp*» реализован следующим образом:

– выбранный элемент в объекте *m_ImportsList* перемещается в поле *imports* объекта *m_config* на одну позицию выше;

– вызывается метод объекта *m_ImportList* для обновления содержимого виджета.

Метод «*OnDirDown*» реализован следующим образом:

– выбранный элемент в объекте *m_ImportsList* перемещается в поле *imports* объекта *m_config* на одну позицию ниже;

– вызывается метод объекта *m_ImportList* для обновления содержимого виджета.

Метод «*OnDirDelete*» реализован следующим образом:

– выбранный элемент в объекте *m_ImportsList* удаляется из поля *imports* объекта *m_config*;

– вызывается метод объекта *m_ImportList* для обновления содержимого виджета.

Метод «*OnCreateFile*» реализован следующим образом:

– создаётся диалог, в котором пользователь выбирает название файла;

– производится проверка, находится ли создаваемый файл в директории проекта;

– вызываются методы «*CreateNewFile*» и «*IncludeFile*».

Метод «*OnOpenFile*» реализован следующим образом:

– производится проверка, существует ли файл выбранный в объекте *m_ProjectExplorer*;

– событие открытия файла передаётся в объект *m_Notebook*.

Метод «*OnSetFileAsMain*» реализован следующим образом:

– выбранный файл в объекте *m_ProjectExplorer* сохраняется в поле *mainFile* объекта *m_projectInfo*;

– вызывается метод «*SetProjectState*»;

– вызывается метод объекта *m_ProjectExplorer* для обновления содержимого виджета.

Метод «*OnFileUp*» реализован следующим образом:

– выбранный элемент в объекте *m_ ProjectExplorer* перемещается в поле *files* объекта *m_ projectInfo* на одну позицию выше;

– вызывается метод «*SetProjectState*»;

– вызывается метод объекта *m_ProjectExplorer* для обновления содержимого виджета.

Метод «*OnFileDown*» реализован следующим образом:

– выбранный элемент в объекте *m_ ProjectExplorer* перемещается в поле *files* объекта *m_ projectInfo* на одну позицию ниже;

– вызывается метод «*SetProjectState*»;

– вызывается метод объекта *m_ProjectExplorer* для обновления содержимого виджета.

Метод «*OnExcludeFile*» реализован следующим образом:

– создаётся диалог, в котором пользователь подтверждает (отменяет) действие;

– если пользователь подтвердил действие, то создаётся событие закрытия вкладки (с информацией о файле) и передаётся в объект *m_Notebook;*

– вызывается метод «*ExcludeFile*»;

– вызывается метод «*SetProjectState*»;

– вызывается метод объекта *m_ProjectExplorer* для обновления содержимого виджета.

Метод «*OnIncludeFile*» реализован следующим образом:

– создаётся диалог, в котором пользователь выбирает файл;

– производится проверка, находится ли выбранный файл в директории проекта;

– вызывается метод объекта *m_projectInfo* для проверки, присутствует ли файл в проекте;

– вызывается метод «*IncludeFile*».

Метод «*OnDeleteFile*» реализован следующим образом:

– создаётся диалог, в котором пользователь подтверждает (отменяет) действие;

– если пользователь подтвердил действие, то создаётся событие закрытия вкладки (с информацией о файле) и передаётся в объект *m_Notebook;*

– вызывается метод «*ExcludeFile*»;

– происходит удаление файла;

– вызывается метод «*SetProjectState*»;

– вызывается метод объекта *m_ProjectExplorer* для обновления содержимого виджета.

Метод «*OnCompile*» реализован следующим образом:

– производится проверка, открыт ли проект;

– производится проверка, присутствуют ли файлы в проекте;

– вызываются методы «*CreateTempDirectories*» и «*ClearTempDirectories*»;

– вызывается метод объекта *m_Console* для очистки виджета;

– с помощью методов «*CreateNewFile*» и «*SaveCompileInfo*» создаётся временный файл с информацией для трансляции языка;

– в цикле по файлам в проекте вызывается метод «*SyncExecWithCaptureOutput*» для трансляции языка;

– удаляется временный файл с информацией для трансляции языка;

– производится проверка, существует ли во временной директории *CPP* файл «*_O2M_make.2mk*»;

– вызывается метод «*SyncExecWithCaptureOutput*» для компоновки;

– вызывается метод «*SyncExecWithCaptureOutput*» для генерации проекта *C++*.

Метод «*OnBuild*» реализован следующим образом:

– производится проверка, открыт ли проект;

– вызывается метод объекта *m_Console* для очистки виджета;

– производится очистка временной директории *Make*;

– производится проверка, существует ли во временной директории *CPP make*-файл;

– вызывается метод «*SyncExecWithCaptureOutput*» для сборки.

Метод «*OnExecute*» реализован следующим образом:

– производится проверка, открыт ли проект;

– производится проверка, существует ли во временной директории *Make* исполняемый файл;

– если существует, то с помощью метода «*CreateNewFile*» создаётся бинарный файл для запуска исполняемого файла;

– вызывается метод «*AsyncExec*» для запуска бинарного файла.

Метод «*OnExit*» реализован следующим образом:

– вызывается унаследованный метод «*Close*».

Метод «*OnCloseWindow*» реализован следующим образом:

– создаётся событие закрытия проекта и передаётся в метод «*OnProjectClose*»;

– производится проверка, был ли удалён объект *m_projectInfo*;

– если был удалён, то происходит уничтожение главного окна, иначе закрытие окна отменяется.

Метод «*OnSetTargetPlatform*» реализован следующим образом:

– в поле *target* объекта *m_projectInfo* сохраняется выбранная платформа;

– вызывается метод «*SetProjectState*».

Методы «*OnUpdate___Menu*» реализованы следующим образом:

– разрешает взаимодействия с элементами меню при определённых условиях.

Метод «*SyncExecWithCaptureOutput*» реализован следующим образом:

– вызывается метод «*wxExecute*»;

– в объект *m_Console* печатается вывод и ошибки.

Метод «*AsyncExec*» реализован следующим образом:

– вызывается метод «*ShellExecute*».

Метод «*SetProjectState*» реализован следующим образом:

– происходит проверка, существует ли объект *m_projectInfo*;

– если существует, то в зависимости от параметра *changed* название проекта помечается звёздочкой или с названия проекта снимается отметка;

– если не существует, то название проекта удаляется.

Метод «*SaveCompileInfo*» реализован следующим образом:

– открывается файл, в который записываются:

а) строки с элементами поля *imports* объекта *m_config*;

б) строка со значением поля *mainFile* объекта *m_projectInfo*;

в) строка со значением поля *execFile* объекта *m_projectInfo*;

г) строка со значением поля *tabSize* объекта *m_config*.

Метод «*CreateImportsList*» реализован следующим образом:

– вызывается конструктор класса «*ImportsList*».

Метод «*CreateMenuBar*» реализован следующим образом:

– создаются объекты класса «*wxMenu*», которые наполняются элементами;

– создаётся объект класса «*wxMenuBar*», который наполняется созданными меню.

Метод «*CreateNotebook*» реализован следующим образом:

– вызывается конструктор класса «*CreateNotebook*».

Метод «*CreateProjectExplorer*» реализован следующим образом:

– вызывается конструктор класса «*CreateProjectExplorer*».

Метод «*CreateConsole*» реализован следующим образом:

– вызывается конструктор класса «*CreateConsole*».

Метод «*CreateNewFile*» реализован следующим образом:

– создаётся и закрывается файл по пути, переданному в параметре *filepath.*

Метод «*CreateTempDirectories*» реализован следующим образом:

– для каждой из трёх временных директорий происходит проверка, существует ли директория;

– если нет, то директория создаётся.

Метод «*ClearTempDirectories*» реализован следующим образом:

– для каждой из трёх директорий происходит удаление всех дочерних элементов.

Метод «*GetProjectInfo*» реализован следующим образом:

– возвращает объект, на который указывает *m_projectInfo.*

Метод «*IncludeFile*» реализован следующим образом:

– в поле *files* объекта *m_projectInfo* добавляется значение, переданное в параметре *filename*;

– вызывается метод «*SetProjectState*»;

– вызывается метод объекта *m_ProjectExplorer* для обновления содержимого виджета.

Метод «*ExcludeFile*» реализован следующим образом:

– из поля *files* объекта *m_projectInfo* удаляется значение с индексом, переданным в параметре *index*;

– производится проверка, присутствует ли значение в поле *mainFile* объекта *m_projectInfo*;

– если да, то значение заменяется на первый элемент в поле *files* объекта *m_projectInfo* или на пустую строку, если поле *files* не содержит элементов.

3.2.4 Виджет «Блокнот»

Данный компонент реализован с помощью класса «*Notebook*», который представлен на рисунке 3.4.

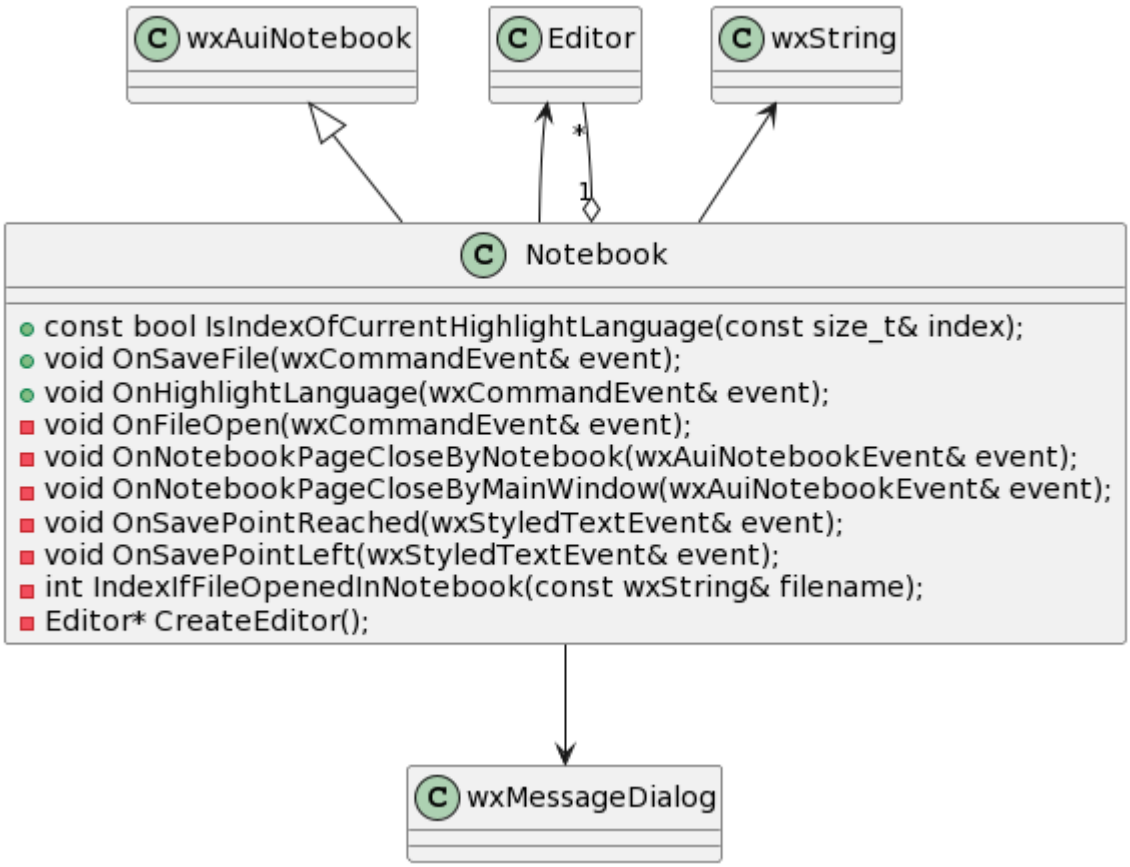

Рисунок 3.4 — Часть диаграммы классов с классом «*Notebook*»

Метод «*IsindexOfCurrentHighlightLanguage*» реализован следующим образом:

– с помощью методов классов «*wxAuiNotebook*» и «*Editor*» проверяется, является ли язык программирования с индексом, равным параметру *index*, текущим подсвечиваемым языком в активной вкладке.

Метод «*OnSaveFile*» реализован следующим образом:

– с помощью методов класса «*wxAuiNotebook*» событие отправляется на обработку в активную вкладку.

Метод «*OnHighlightLanguage*» реализован следующим образом:

– с помощью методов класса «*wxAuiNotebook*» событие отправляется на обработку в активную вкладку.

Метод «*OnFileOpen*» реализован следующим образом:

– с помощью методов класса «*wxTreeCtrl*» принимается информация из события о пути к файлу;

– вызывается метод «*IndexIfFileOpenedInNotebook*» для получения индекса вкладки, если файл уже открыт;

– если файл открыт, то вкладка с индексом, равным полученному индексу, делается активной, и метод завершается;

– если файл ещё не открыт, то вызывается метод «*CreateEditor*» и указатель на новый объект добавляется к вкладкам;

– событие отправляется на обработку в только что созданную вкладку.

Метод «*OnNotebookPageCloseByNotebook*» реализован следующим образом:

– с помощью методов классов «*wxAuiNotebook*» и «*Editor*» проверяется, присутствуют ли изменения в активной вкладке;

– если изменения присутствуют, то с помощью метода класса «*wxMessageDialog*» создаётся диалоговое окно, которое спрашивает пользователя, желает ли он сохранить файл;

– если пользователь нажимает на «Отмена», то событие закрытия вкладки отменяется, и метод завершается;

– если пользователь нажимает «Да», то файл сохраняется (даже если во время редактирования файла в приложении удалить его из директории проекта);

– событие отправляется в родительский класс для закрытия вкладки.

Метод «*OnNotebookPageCloseByMainWindow*» реализован следующим образом:

– с помощью методов классов «*wxAuiNotebook*» и «*Editor*» проверяется, присутствуют ли изменения в активной вкладке;

– если изменения присутствуют, то с помощью метода класса «*wxMessageDialog*» создаётся диалоговое окно, которое спрашивает пользователя, желает ли он сохранить файл;

– если пользователь нажимает на «Отмена», то событие закрытия вкладки отменяется, и метод завершается;

– если пользователь нажимает «Да», то файл сохраняется (даже если во время редактирования файла в приложении удалить его из директории проекта);

– вкладка закрывается.

Метод «*OnSavePointReashed*» реализован следующим образом:

– с помощью методов классов «*wxAuiNotebook*», «*Editor*» и «*wxString*» с названия активной вкладки убирается отметка символом «звёздочка».

Метод «*OnSavePointLeft*» реализован следующим образом:

– с помощью методов классов «*wxAuiNotebook*», «*Editor*» и «*wxString*» к названию активной вкладки добавляется отметка символом «звёздочка».

Метод «*IndexIfFileOpenedInNotebook*» реализован следующим образом:

– в цикле по всем вкладкам с помощью методов классов «*wxAuiNotebook*» и «*Editor*» проверяется, совпадает ли путь к файлу, который открыт во вкладке, с параметром *filepath*;

– если совпадает, то возвращается индекс вкладки, иначе возвращается «-1».

Метод «*CreateEditor*» реализован следующим образом:

– с помощью методов класса «*Editor*» создаётся объект этого класса, и возвращается указатель на этот объект.

3.2.5 Виджет «Редактор»

Данный компонент реализован с помощью класса «*Editor*», который представлен на рисунке 3.5.

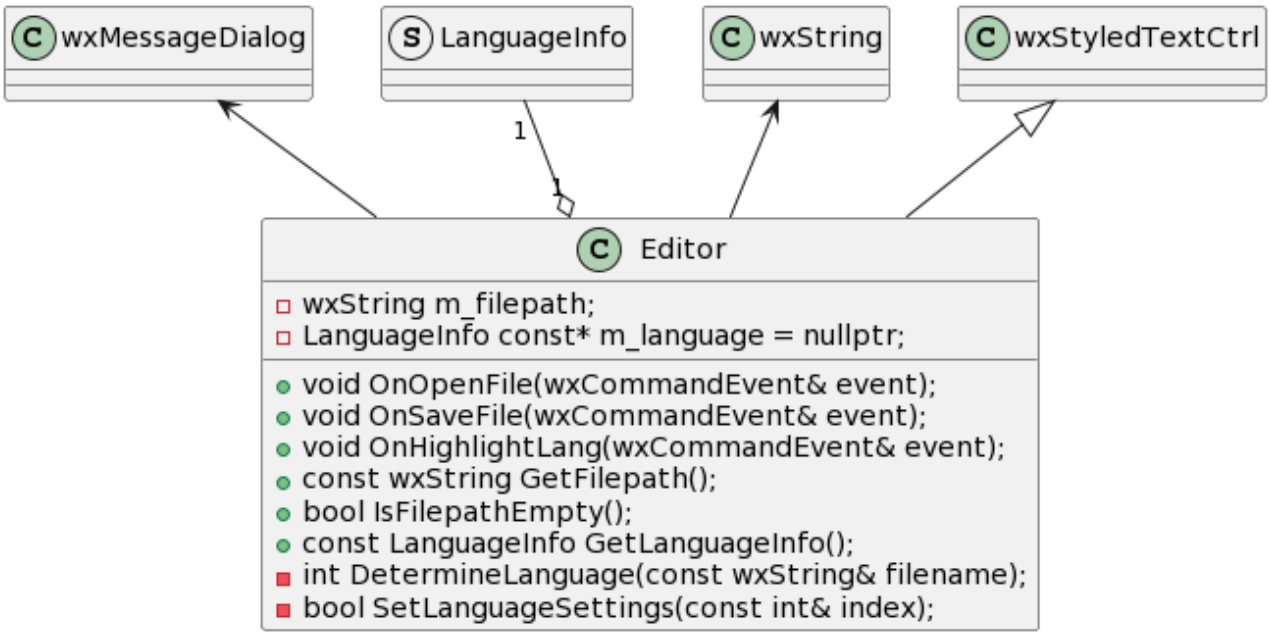

Рисунок 3.5 — Часть диаграммы классов с классом «*Editor*»

Поле *m_filepath* хранит путь к файлу. Поле *m_language* хранит указатель на структуру данных «Информация о языке».

Метод «*OnOpenFile*» реализован следующим образом:

– с помощью методов класса «*Editor*» принимается информация из события о пути к файлу;

– файл считывается во вкладку — виджет «Редактор»;

– последовательно вызываются методы «*DetermineLanguage*» и «*SetLanguageSettings*».

Метод «*OnSaveFile*» реализован следующим образом:

– с помощью методов класса «*Editor*» проверяется, присутствуют ли изменения во вкладке;

– если присутствуют, то проверяется, пусто ли поле *m_filepath* и существует ли файл по пути, хранящемуся в этом поле;

– если поле пустое или файл не существует, то с помощью метода класса «*wxMessageDialog*» создаётся диалоговое окно, которое сообщает пользователю о том, что путь к файлу неизвестен или файл не существует;

– если поле не пустое и файл существует, то содержимое вкладки записывается в файл по заданному пути.

Метод «*OnHighlightLang*» реализован следующим образом:

– вызывается метод «*SetLanguageSettings*» класса «*Editor*».

Метод «*GetFilepath*» реализован следующим образом:

– возвращается значение поля *m_filepath*.

Метод «*IsFilepathEmpty*» реализован следующим образом:

– с помощью метода класса «*wxString*» проверяется, пусто ли поле *m_filepath*.

Метод «*GetLanguageInfo*» реализован следующим образом:

– возвращается объект структуры «Информация о языке», на который ссылается указатель *m_language*.

Метод «*DetermineLanguage*» реализован следующим образом:

– в цикле по всем объектам структуры «Информация о языке» с помощью методов класса «*wxString*» поле *filepattern* структуры разбивается на подстроки, которые являются возможными расширениями файлов, написанных на языке, информацию о котором хранит объект структуры;

– проверяется, совпадает ли расширение файла, путь к которому передаётся в параметре *filename*, с подстроками из предыдущего этапа;

– если совпадает, то возвращается индекс объекта структуры, иначе возвращается «0» (под этим индексом всегда хранится информация о языке по умолчанию).

Метод «*SetLanguageSettings*» реализован следующим образом:

– с помощью методов класса «*Editor*» очищаются все стили;

– устанавливается тип, стиль и ширина отступа для отображения номера строки;

– устанавливается тип, ширина и чувствительность к кликам мыши разделителя, который находится между нумерацией строк файла и содержанием строк;

– в поле *m_language* сохраняется указатель на объект структуры «Информация о языке», имеющий индекс, равный параметру *index*;

– происходит обработка полей объекта структуры:

а) устанавливается лексер, название которого хранится в поле *lexer*;

б) в цикле по элементам поля *styles* устанавливаются стили с названиями, хранящимися в поле *type* структуры «*Style*», и описаниями, хранящимися в поле *spec* структуры «*Style*»;

в) в цикле по элементам поля *keywordsStyles* устанавливаются наборы ключевых слов с названиями, хранящимися в поле *type* структуры «*KeywordsStyle*», и списками, хранящимися в поле *wordsString* структуры «*KeywordsStyle*»;

– устанавливается размер символа табуляции.

3.2.6 Структура данных информации о языке

Данный компонент реализован с помощью структур «*LanguageInfo*» «*Style*» и «*Keywords*», которые представлены на рисунке 3.6.

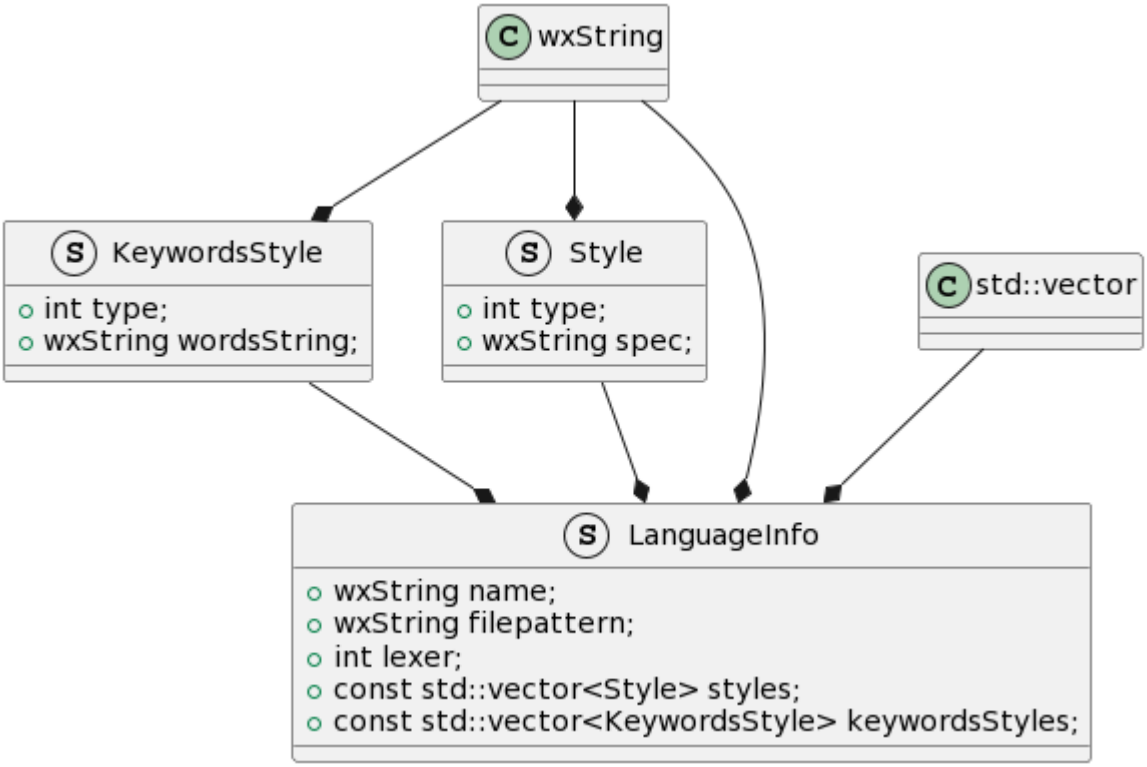

Рисунок 3.6 — Часть диаграммы классов со структурой «*LanguageInfo*»

Назначение полей структуры «*Style*»:

– поле *type* хранит название стиля;

– поле *spec* хранит описание стиля.

Назначение полей структуры «*KeywordsStyle*»:

– поле *type* хранит название набора ключевых слов;

– поле *wordsString* хранит список ключевых слов.

Назначение полей структуры «*LanguageInfo*»:

– поле *name* хранит название языка;

– поле *filepattern* хранит возможные расширения файлов, написанных на данном языке;

– поле *lexer* хранит название лексера;

– поле *styles* хранит массив стилей;

– поле *keywordsStyles* хранит массив списков ключевых слов.

3.2.7 Подсветка синтаксиса

Данный компонент реализован с помощью класса «*LexerO2M*», который представлен на рисунке 3.7.

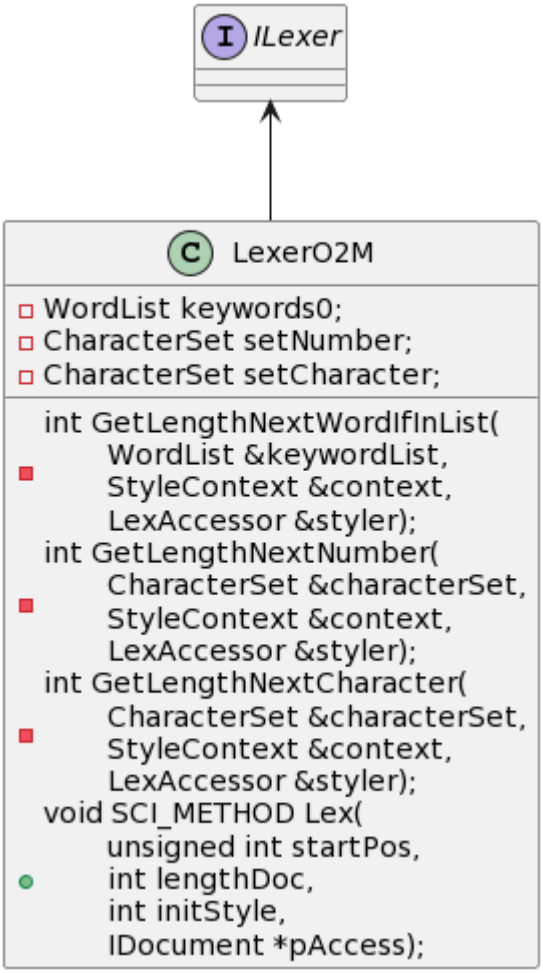

Рисунок 3.7 — Часть диаграммы классов с классом «*LexerO2M*»

Класс не является непосредственной частью приложения — он внедрён в библиотеку. Такое решение было связано с тем, что библиотека содержит в себе компонент «*Scintilla*», который уже имеет файлы лексеров для других языков программирования. Достаточно написать собственный файл для нового языка и пересобрать библиотеку.

Поле *keywords0* хранит список ключевых слов. Поле *setNumber* хранит список символов, которые могут встретиться в числах. Поле *setCharacter*

хранит список символов, которые могут встретиться в символьных константах (обозначаются порядковым номером символа в шестнадцатеричной записи, оканчивающейся буквой «*X*»).

Метод «*GetLengthNextWordIfInList*» реализован следующим образом:

– создаётся буфер *word* длинной на один символ больше длинны самого длинного ключевого слова языка;

– определяется текущая позиция обрабатываемого символа во вкладке;

– так как все ключевые слова языка «*O2M*» пишутся в верхнем регистре, в цикле происходит заполнение буфера символами, написанными только в верхнем регистре;

– проверяется наличие слова, записанного в буфере, в списке ключевых слов, который передаётся в параметре *keywordList*;

– если слово присутствует, то возвращается номер позиции символа, следующим за крайним символом слова.

Метод «*GetLengthNextNumber*» реализован следующим образом:

– создаётся буфер *word* размером с максимальную длину строки;

– создаются регулярные выражения, определяющие целые, шестнадцатеричные и вещественные числа;

– определяется текущая позиция обрабатываемого символа во вкладке;

– в цикле происходит заполнение буфера символами, которые присутствуют в списке символов, который предаётся в параметре *characterSet*;

– проверяется строка, записанная в буфере, на соответствие одному из регулярных выражений;

– если строка соответствует регулярному выражению, то возвращается номер позиции символа, следующим за крайним символом числа.

Метод «*GetLengthNextCharacter*» реализован следующим образом:

– создаётся буфер *word* размером с максимальную длину строки;

– создаётся регулярное выражение, определяющее символьные константы;

– определяется текущая позиция обрабатываемого символа во вкладке;

– в цикле происходит заполнение буфера символами, которые присутствуют в списке символов, который предаётся в параметре *characterSet*;

– проверяется строка, записанная в буфере, на соответствие регулярному выражению;

– если строка соответствует регулярному выражению, то возвращается номер позиции символа, следующим за крайним символом символьной константы.

Метод «*Lex*» реализован следующим образом:

– осуществляет лексический анализ текста на основе Расширенных Бэкуса-Наура Форм (РБНФ):

а) идент = буква {буква | цифра};

б) цифра = «0» | «1» | «2» | «3» | «4» | «5» | «6» | «7» | «8» | «9»;

в) шестнЦифра = цифра | «A» | «B» | «C» | «D» | «E» | «F»;

г) порядок = («E» | «D») [«+» | «-»] цифра {цифра};

д) целое = (цифра {цифра}) | (цифра {шестнЦифра} «H»;

е) вещественное = цифра {цифра} «.» {цифра} [порядок];

ж) число = целое | вещественное;

з) символКонст = цифра {шестнЦифра} «X»;

и) строка = («"» {буква} «"») | («'» {буква} «'»);

к) комментарий = «(*» {любой символ} «*)»;

л) ключСлово = «ARRAY» | «BEGIN» | «BY» | «CASE» | «CONST» | «DIV» | «DO» | «ELSE» | «ELSIF» | «END» | «EXIT» | «FOR» | «IF» | «IMPORT» | «IN» | «IS» | «LOOP» | «MOD» | «MODULE» | «NIL» | «OF» | «OR» | «POINTER» | «PROCEDURE» | «RECORD» | «REPEAT» | «RETURN» | «THEN» | «TO» | «TYPE» | «UNTIL» | «VAR» | «WHILE» | «WITH» | «FALSE» | «TRUE» | «ABS» | «ASH» | «CAP» | «CHR» | «ENTIER» | «LEN» | «LONG» | «MAX» | «MIN» | «ODD» | «ORD» | «SHORT» | «SIZE» | «ASSERT» | «COPY» | «DEC» | «EXCL» | «HALT» | «INC» | «INCL» | «NEW» | «BOOLEAN» |

«CHAR» | «INTEGER» | «LONGINT» | «LONGREAL» | «REAL» | «SET» | «SHORTINT» | «LOCAL»;

– существуют состояния символов, от которых зависит стиль отображения символов: *Default*, *BlockComment*, *LineComment*, *SingleString*, *DoubleString*, *Number*, *Character*, *Keyword*;

– в цикле по символам во вкладке проверяется, является ли текущее состояние *Default*;

– если является, то для текущего символа проверяются условия для перехода в другое состояние:

а) если текущий и следующий символы совпадают с «(*», то устанавливается состояние *BlockComment*, и указатель на текущий символ перемещается на два символа вправо;

б) если текущий и следующий символы совпадают с «//», то устанавливается состояние *LineComment*, и указатель на текущий символ перемещается на два символа вправо;

в) если текущий символ совпадает с «'», то устанавливается состояние *SingleString*, и указатель на текущий символ перемещается на один символ вправо;

г) если текущий символ совпадает с «"», то устанавливается состояние *DoubleString*, и указатель на текущий символ перемещается на один символ вправо;

д) если текущий символ является цифрой, то вызывается метод «*GetLengthNextCharacter*»;

е) если метод вернул длину больше нуля, то устанавливается состояние *Character*, и указатель на текущий символ перемещается на длину символьной константы вправо, иначе вызывается метод «*GetLengthNextNumber*»;

ж) если метод вернул длину больше нуля, то устанавливается состояние *Number*, и указатель на текущий символ перемещается на длину числа вправо;

з) если текущий символ написан в верхнем регистре, то вызывается метод «*GetLengthNextWordIfInList*»;

и) если метод вернул длину больше нуля, то устанавливается состояние *Keyword*, и указатель на текущий символ перемещается на длину ключевого слова вправо;

– если текущее состояние символа не *Default*, то происходит определение состояния и дальнейших действий:

а) если текущее состояние *BlockComment* и текущий и следующий символы совпадают с «*)», то указатель на текущий символ перемещается на два символа вправо, и устанавливается состояние *Default*;

б) если текущее состояние *LineComment* и текущий символ является концом строки, то указатель на текущий символ перемещается на один символ вправо, и устанавливается состояние *Default*;

в) если текущее состояние *SingleString* и текущий символ является концом строки или совпадает с «'», то указатель на текущий символ перемещается на один символ вправо, и устанавливается состояние *Default*;

г) если текущее состояние *DoubleString* и текущий символ является концом строки или совпадает с «"», то указатель на текущий символ перемещается на один символ вправо, и устанавливается состояние *Default*;

д) если текущее состояние *Number*, *Character* или *Keyword*, то устанавливается состояние *Default*;

– после обработки всех символов вызывается метод для отрисовки стилей.

3.2.8 Виджет «Обозреватель проекта»

Данный компонент реализован с помощью класса «*ProjectExplorer*», который представлен на рисунке 3.8.

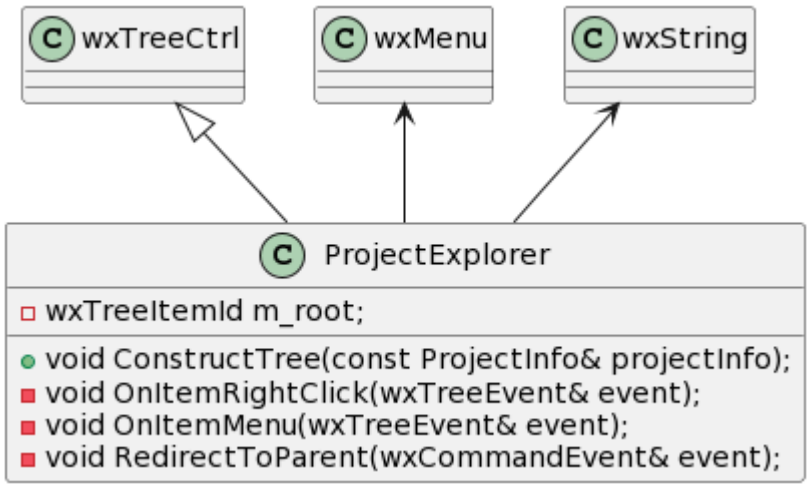

Рисунок 3.8 — Часть диаграммы классов с классом «*ProjectExplorer*»

Поле *m_root* хранит корень дерева в виджете.

Метод «*ConstructTree*» реализован следующим образом:

– с помощью методов класса «*wxTreeCtrl*» элемент *filepath* параметра *projectInfo* устанавливается как корень дерева в виджете, файлы из элемента *files* параметра *projectInfo* последовательно добавляются как дочерние элементы корня дерева в виджете.

Метод «*OnItemRightClick*» реализован следующим образом:

– с помощью методов класса «*wxTreeCtrl*» устанавливается активным елемент виджета, по которому был произведён клик правой кнопкой мыши;

– вызывается метод «*OnItemMenu*».

Метод «*OnItemMenu*» реализован следующим образом:

– с помощью методов класса «*wxTreeCtrl*» и «*wxMenu*» создаётся, заполняется элементами контекстное меню и открывается на месте, где был произведён клик правой кнопкой мыши.

Метод «*RedirectToParent*» реализован следующим образом:

– с помощью методов класса «*wxTreeCtrl*» к событию, переданному в параметре, добавляется информация об индексе активного элемента виджета;

– событие отправляется на обработку в родительское окно.

3.2.9 Виджет «Список подключаемых директорий»

Данный компонент реализован с помощью класса «*ImportsList*», который представлен на рисунке 3.9.

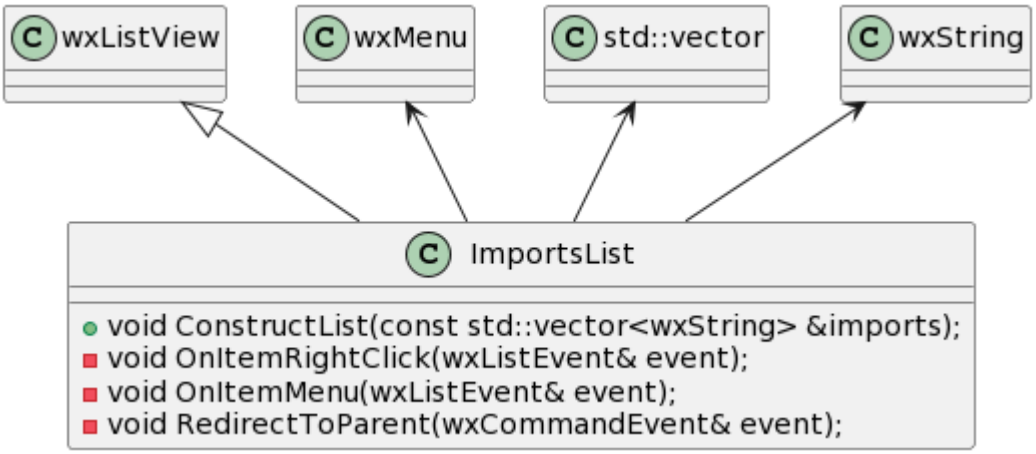

Рисунок 3.9 — Часть диаграммы классов с классом «*ImportsList*»

Метод «*ConstructList*» реализован следующим образом:

– с помощью методов классов «*wxListView*» и «*std::vector*» элементы параметра *imports* построчно выводятся в виджет в колонку «*Path*».

Метод «*OnItemRightClick*» реализован следующим образом:

– с помощью методов класса «*wxListBox*» устанавливается активным елемент виджета, по которому был произведён клик правой кнопкой мыши;

– вызывается метод «*OnMenuItem*».

Метод «*OnMenuItem*» реализован следующим образом:

– с помощью методов класса «*wxListBox*» и «*wxMenu*» создаётся, заполняется элементами контекстное меню и открывается на месте, где был произведён клик правой кнопкой мыши.

Метод «*RedirectToParent*» реализован следующим образом:

– с помощью методов класса «*wxListBox*» к событию, переданному в параметре, добавляется информация об индексе активного элемента виджета;

– событие отправляется на обработку в родительское окно.

3.2.10 Виджет «Консоль»

Данный компонент реализован с помощью класса «*Console*», который представлен на рисунке 3.10.

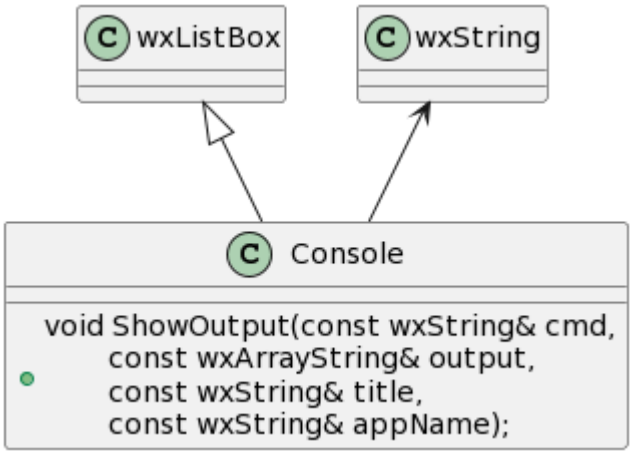

Рисунок 3.10 — Часть диаграммы классов с классом «*Console*»

Метод «*ShowOutput*» реализован следующим образом:

– с помощью методов классов «*wxListBox*» и «*wxString*» параметры преобразуются в строки определённого формата и выводятся в виджет.

3.3 Тестирование редактора кода

Тестирование редактора кода производилось на 30 тестовых проектах, представленных на рисунке 3.11.

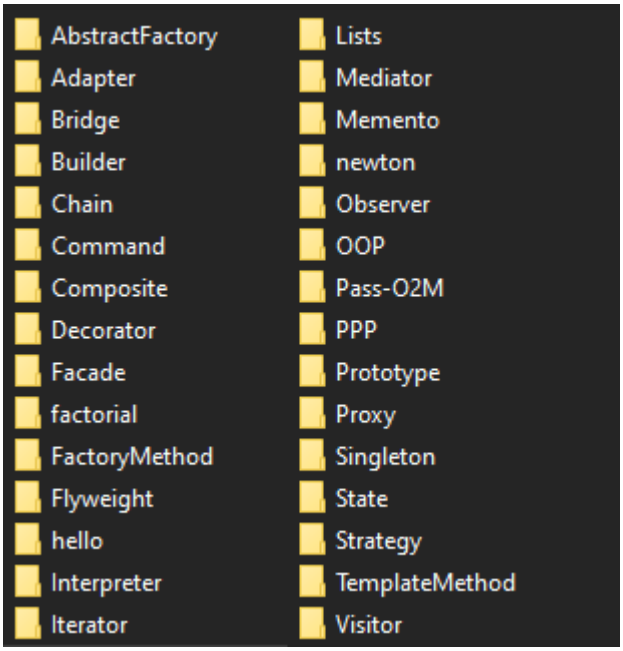

Рисунок 3.11 — Набор тестовых проектов

Все тестовые проекты выдали ожидаемый результат.

3.4 Документация к редактору кода

Инструкция по использованию редактора кода написана в меню «Справка» в приложении. Таже, в качестве инструкции пользователя могут использоваться диаграммы, разработанные на этапе проектирования (пункт 2 данного документа).

Для реализации интегрированной среды разработки использовалась «*MS Visul Studio 2022*».

Во время написания исходного кода разрабатываемого приложения была модифицирована часть библиотеки «*wxWidgets*». Поэтому важно знать этот факт и его ньюансы. Перед сборкой библиотеки необходимо выполнить следующее:

– создать системную переменную «*WXWIN* = <путь к библиотеке>»;

– создать строки в переменной среды:

а) «*%WXWIN%\lib\vc_x64_lib*»;

б) путь к утилите «*Nmake*»;

– Файл «*LexO2M.cxx*» положить в папку «*%WXWIN%\src\stc\scintilla\lexers*»;

– В файл «*%WXWIN%\src\stc\scintilla\include\SciLexer.h*» дописать строки по аналогии с другими лексерами:

а) «*#define SCLEX_O2M 122*»;

б) «*#define SCE_O2M_DEFAULT 0*»;

в) «*#define SCE_O2M_NUMBER 1*»;

г) «*#define SCE_O2M_SINGLESTRING 2*»;

д) «*#define SCE_O2M_DOUBLESTRING 3*»;

е) «*#define SCE_O2M_CHARACTER 4*»;

ж) «*#define SCE_O2M_KEYWORD 5*»;

з) «*#define SCE_O2M_BLOCKCOMMENT 6*»;

и) «*#define SCE_O2M_LINECOMMENT 7*»;

– В файлы «*%WXWIN%\include\wx\stc\stc.h*» и «*%WXWIN%\interface\wx\stc\stc.h*» дописать строки по аналогии с другими лексерами:

а) «*#define wxSTC_LEX_O2M 122*»;

б) «*#define wxSTC_O2M_DEFAULT 0*»;

в) «*#define wxSTC_O2M_NUMBER 1*»;

г) «*#define wxSTC_O2M_SINGLESTRING 2*»;

д) «*#define wxSTC_O2M_DOUBLESTRING 3*»;

е) «*#define wxSTC_O2M_CHARACTER 4*»;

ж) «*#define wxSTC_O2M_KEYWORD 5*»;

з) «*#define wxSTC_O2M_BLOCKCOMMENT 6*»;

и) «*#define wxSTC_O2M_LINECOMMENT 7*»;

– В файле «*%WXWIN%\src\stc\scintilla\lexers\descrip.mms*» дописать строки по аналогии с другими лексерами:

а) Добавить «*LexO2M.obj*» к одной из переменных *OBJECTS*;

б) Добавить «*LexO2M.cxx*» к одной из переменных *SOURCES*;

в) «*LexO2M.obj : LexO2M.cxx*»;

– В файл «*%WXWIN%\src\stc\scintilla\src\Catalogue.cxx*» дописать строку по аналогии с другими лексерами:

а) «*LINK_LEXER(lmO2M);*».

После описанных выше этапов можно собрать библиотеку. Алгоритмы и варианты сборки описаны на просторах интернета.

Для сборки приложения необходимо в проект включить файлы, находящиеся в репозитории [6] (кроме файла «*LexO2M.cxx*»), а также подключить библиотеку к проекту. Методы подключения библиотек полно и подробно описаны на просторах интернета, поэтому их не трудно найти.

3.5 Выводы по разделу 3

Для разработки редактора кода была выбрана библиотека *wxWidgets 3.2.2.1*, реализованная на языке *C++*.

Разработаны компоненты приложения, которые реализуют весь спроектированный функционал редактора кода.

Редактор кода успешно протестирован на наборе тестовых проектов.

Написаны инструкция для пользователя и инструкция для разработчика приложения.

ЗАКЛЮЧЕНИЕ

В результате проделанной работы был разработан редактор кода для экспериментального языка программирования O2M, поддерживающий уникальные возможности данного языка, что соответствует цели работы.

В процессе выполнения ВКР были полностью решены задачи:

– исследованы особенности существующих редакторов кода;

– разработана программная архитектура редактора кода в ходе проектирования на основе процесса «*ICONIX*»;

– выбраны инструменты разработки, которыми стали «*MS Visual Studio 2022*» и «*wxWidgets 3.2.2.1*», и реализован редактор кода на основе разработанной архитектуры;

– произведено тестирование редактора кода на готовых примерах программ на языке O2M, в результате чего достигнута стабильность работы приложения.

Перспективами данной работы являются:

– модификация интегрированной среды разработки для внедрения новых возможностей языка O2M, которые появятся в будущем;

– перенос редактора кода на другие платформы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Code::Blocks : [сайт]. – URL: https://www.codeblocks.org (дата обращения: 20.11.2022).

2 CodeLite IDE : [сайт]. – URL: https://codelite.org (дата обращения: 20.11.2022).

3 KDevelop : [сайт]. – URL: https://kdevelop.org (дата обращения: 20.11.2022).

4 Процесс разработки программного обеспечения ICONIX / Блог программиста — программирование и алгоритмы : [сайт]. – URL: https://pro-prof.com/archives/4126 (дата обращения: 15.05.2023).

5 Модели, парадигмы и методологии разработки ПО / ВКонтакте : [сайт]. – URL: https://vk.com/@foresttzar-modeli-paradigmy-i-metodologii-razrabotki-po (дата обращения: 15.05.2023).

6 Репозиторий «BelousovAD/CodeEditor» / GitHub : [сайт]. – URL: https://github.com/BelousovAD/CodeEditor (дата обращения: 18.06.2023).

Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой
_____ О.В. Непомнящий
"21" 06 2023 г.

БАКАЛАВРСКАЯ РАБОТА

090301 Информатика и вычислительная техника

Редактор кода для экспериментального языка программирования O2M

Руководитель	подпись	21.06.23 дата	доцент, канд. техн. наук *должность, ученая степень*	Д.А. Швец
Выпускник	подпись	21.06.23 дата		А.Д. Белоусов
Нормоконтролёр	подпись	21.06.23 дата		Д.А. Швец

Красноярск 2023