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
подпись
«_____» _____ 2022 г.

# БАКАЛАВРСКАЯ РАБОТА

09.03.01 – Информатика и вычислительная техника

Система контрактов для функционально-потокового языка параллельного программирования

| | | | |
|---|---|---|---|
| Руководитель | _____ подпись, дата | ст. преподаватель | В.С. Васильев |
| Выпускник | _____ подпись, дата | | Д.А. Елисеенко |
| Нормоконтролер | _____ подпись, дата | ст. преподаватель | В.С. Васильев |

Красноярск 2022

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
подпись
«_____» _____ 2022 г.

**ЗАДАНИЕ**
**НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ**
**в форме бакалаврской работы**

Красноярск 2022

Студенту Елисеенко Дмитрию Анатольевичу

Группа КИ18-07Б Направление (специальность) 09.03.01

Информатика и вычислительная техника

Тема выпускной квалификационной работы Система контрактов для функционально-потокового языка параллельного программирования

Утверждена приказом по университету № _____ от _____

Руководитель ВКР В.С. Васильев, старший преподаватель кафедры «Информатика и вычислительная техника» ИКИТ СФУ.

Исходные данные для ВКР:

1. Легалов А.И., Васильев В.С., Матковский И.В., Ушакова М.С. Инструментальная поддержка создания и трансформации функционально-потоковых параллельных программ. Труды ИСП РАН, том 29, вып. 5, 2017 г., стр. 165-184.
2. Васильев, В. С. Трансформация функционально-потоковых параллельных программ в императивные / В. С. Васильев, А. И. Легалов, С. В. Зыков // Моделирование и анализ информационных систем. – 2021. – Т. 28. – № 2. – С. 198-214.
3. Рекомендации руководителя.

Перечень разделов ВКР: введение, разработка спецификации требований, проектирование, реализация и интеграция с существующими инструментами, заключение.

Перечень графического материала: демонстрационное видео, презентация в формате pdf.

Руководитель ВКР _____ В.С. Васильев
подпись

Задание принял к исполнению _____ Д.А. Елисеенко
подпись

«____» _____ 2022 г.

# РЕФЕРАТ

Выпускная квалификационная работа по теме «Система контрактов для функционально-потокового языка параллельного программирования» содержит 42 страницы текстового документа, 19 иллюстраций, 2 таблицы, 1 приложение, 29 использованных источников.

ИНСТРУМЕНТАЛЬНОЕ СРЕДСТВО, ФУНКЦИОНАЛЬНО-ПОТОКОВЫЙ ЯЗЫК ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ, ПИФАГОР, КОНТРАКТНОЕ ПРОГРАММИРОВАНИЕ, QT, ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ, ООП, ШАБЛОНЫ ПРОЕКТИРОВАНИЯ.

Цель работы – разработать консольное приложение, формирующее описание типов данных для входных параметров и возвращаемого значения функций функционально-потокового языка параллельного программирования Пифагор.

Выпускная квалификационная работа состоит из трёх частей: введения, основной части, разделенной на три главы, и заключения.

Во введении приводится краткий обзор существующих инструментальных средств системы функционально-потокового параллельного программирования, обосновывается необходимость в создании данного инструмента.

В первой главе предлагается формат комментариев специального вида (контрактов), позволяющих формировать описание типов данных функций на основе анализа исходного кода программ. Формализуется спецификация требований к программе.

Во второй главе выполняется проектирование приложения в соответствии с методами объектно-ориентированного моделирования.

В третьей главе представлены детали реализации разработанной программы и её интеграции в набор существующих инструментальных средств.

В заключении подводятся итоги по выполненной работе.

# СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Для широко используемых языков программирования существуют инструментальные средства, автоматизирующие различные рутинные процессы с которыми сталкиваются программисты, сокращающие сроки и стоимость разработки. Среды разработки позволяют добавлять поддержку новых языков программирования и сторонних утилит посредством механизма плагинов.

Функционально-потоковая парадигма (ФПП) параллельного программирования обладает рядом особенностей, затрудняющих использование готовых решений для создания интегрированной среды разработки (ИСР). Наиболее развитым ФПП языком является Пифагор [1]. Для этого языка существует множество инструментальных средств, которые должна объединять ИСР. Среди них:

– транслятор [2], преобразующий текстовое представление программы в набор реверсивных информационных графов (РИГ) и управляющих графов (УГ), имеющих текстовое и графическое представление;

– интерпретатор [3], обеспечивающий возможность исполнения этих программ;

– система синтеза сверхбольших интегральных схем (СБИС) на основе РИГ [4];

– верификатор, позволяющий доказывать свойства программ;

– отладчик, предоставляющий возможность исполнения программы в пошаговом режиме и просмотра текущих значений переменных;

– система преобразования программ в императивную форму (программ языка C++) [5];

– система оптимизации кода, позволяющая выполнять 14 различных преобразований [6];

– система, реализующая хранение программ в различных репозиториях.

Некоторые из этих инструментов помимо исходного кода программ, РИГ и УГ ожидают информацию о типах данных параметров функций. В работе [7]

предложено задавать информацию о типах данных параметров функций непосредственно в исходном коде программ. **Актуальной** является проблема формирования на основе этой информации описаний типов данных в форматах, поддерживаемых различными инструментами.

**В первой главе** работы предложен формат описания типов данных функций языка Пифагор учитывающий, что различные инструменты используют отличающиеся спецификации типов.

**Во второй главе** рассмотрены детали проектирования инструментального средства, в результате которого сформированы статическая и динамическая модели системы.

**В третьей главе** рассмотрены вопросы реализации и тестирования инструмента, а также его интеграции в существующую систему ФПП программирования.

# 1 Разработка спецификации требований

## 1.1 Анализ существующего подхода к разработке ФПП программ

Для языка программирования Пифагор создано множество инструментальных средств, необходимых для разработки, отладки, оптимизации, верификации и исполнения программ, поддерживающих функционально-потоковую парадигму параллельного программирования [8]. Кроме того, существует ряд инструментов, транслирующих ФПП программы в программы для других архитектур параллельных вычислительных систем (ПВС) [4; 5].

На рисунке 1.1 представлена схема, демонстрирующая зависимости между имеющимися инструментами, а также зависимость каждого инструмента от входных данных, которые могут быть получены, как результат работы других инструментальных средств. Овал на рисунке соответствует инструментальному средству ФПП программирования, прямоугольник задает артефакт, используемый инструментом или формируемый в результате его работы, стрелки отражают потоки данных.

Транслятор ФПП программ преобразует исходный код функции в реверсивный информационный граф, являющийся формой её внутреннего представления. На основе РИГ для каждой функции строится УГ. Набор из РИГ и УГ используется другими инструментами для дальнейшего преобразования или интерпретации программы. Некоторым инструментальным средствам помимо РИГ и УГ требуется описание используемых типов данных, но язык Пифагор использует динамическую типизацию, которая не позволяет определить тип данных программной конструкции до интерпретации. Данная проблема решается в статически типизированном ФПП языке Smile [9], но он находится в стадии разработки.

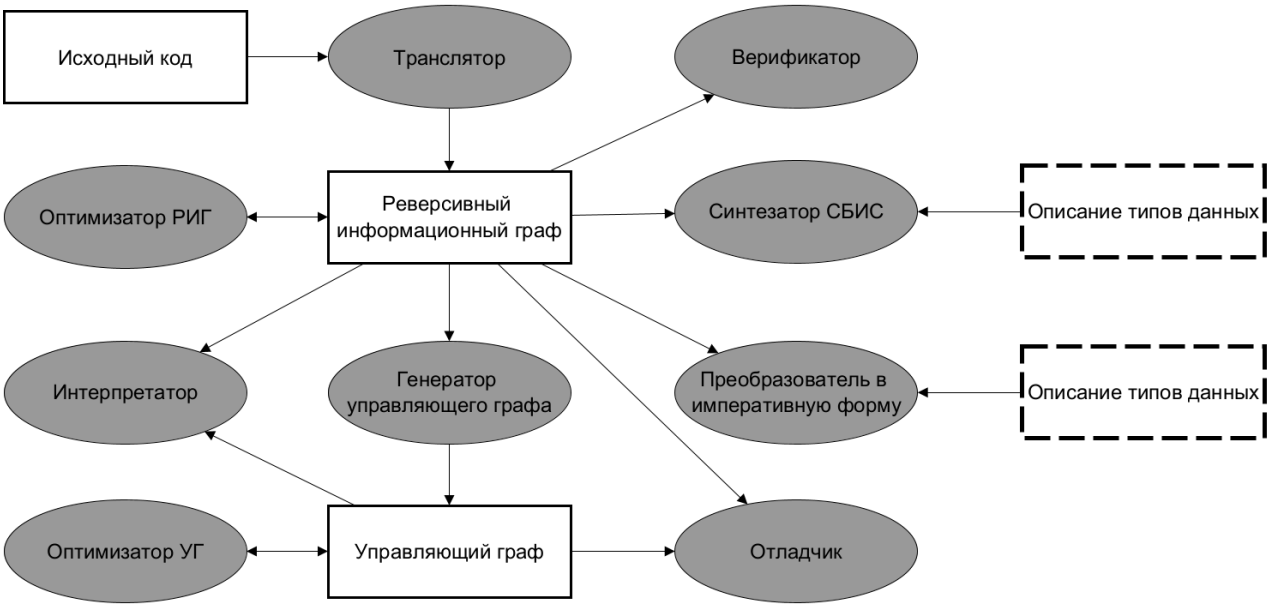


Рисунок 1.1 – Зависимости между инструментами функционально-потокового параллельного программирования

На данный момент те инструменты, которым требуется информация о типах данных, получают ее из текстовых файлов (прямоугольники с прерывистой контурной линией), содержащих описание типов в определенных форматах. Форматы описания типов данных отличаются для различных инструментов. Типизируются аргументы и возвращаемое значение для каждой функции, соответствующая информация вручную вводится программистом и размещается в определенном месте для каждой функции.

Артефакты ФПП программы хранятся в специальной системе-репозитории [10], она же используется для размещения информации о типах данных. В настоящий момент репозиторий использует файловую систему для хранения текста артефактов в виде файлов, однако возможно развитие этой системы, добавляющее поддержку других способов хранения информации. Все артефакты, полученные при работе с инструментами ФПП программирования, репозиторий хранит в дочерних поддиректориях директории repository. Для каждой функции создается своя поддиректория, которая содержит: файл с исходным кодом функции, РИГ, УГ, файл с описанием типов входных аргументов и возвращаемого значения функции, а также файлы для сторонних

инструментов, используемых для графического отображения разных внутренних представлений функции.

В рамках настоящей работы предложено описывать типы данных функций непосредственно в файле с исходным кодом, разработана система позволяющая формировать файлы с информацией о типах данных в форматах, поддерживаемых существующими инструментами ФПП программирования и размещать их в репозитории.

## 1.2 Разработка универсального формата описания типов данных

В работе [7] приведен формат описания типов, поддерживаемый системами оптимизации и трансформации ФПП программ в императивную форму. Определение этого формата в виде РБНФ представлено на рисунке 1.2.

```
1 <Тип функции> ::= <Тип аргумента> -> <Тип возвращаемого значения>.
2 <Тип аргумента> ::= @{<Элементы кортежа>}.
3 <Элементы кортежа> ::= <пусто> | <Тип> | <Тип> [,<Элементы кортежа>].
4 <Тип> ::= @char | @int | @double | <Тип списка> | <Тип функции>.
5 <Тип списка> ::= @(<Тип>) | @[<Тип>].
6 <Тип возвращаемого значения> ::= <Тип>.
```

Рисунок 1.2 – Определение формата описания типов данных в оптимизаторе и трансляторе в императивную форму

На рисунке 1.3 приведена функция, вычисляющая сумму двух векторов. На языке Пифагор она обработает вектора любых типов, поддерживающих операцию сложения, однако в первой строке примера эта функция дополнена комментарием, который фиксирует, что первый входной вектор хранит целые значения, второй – вещественные, а сама функция вернет вектор вещественных чисел.

В верификатор информация о типах данных вводится вручную в процессе доказательства свойств программ с помощью графического пользовательского интерфейса [11].

```
1 //@{@(@int),@(@double)}->@(@double)
2 vec_sum << funcdef vecPair {
3   A << vecPair:1;
4   B << vecPair:2;
5   C << (A,B):#:[]:+;
6   return << C;
7 }
```

Рисунок 1.3 – Пример использования формата описания типов для функции

В средство синтеза СБИС [12] информация о типах данных функций вводится таким же образом, как и в систему оптимизации, однако формат описания типов отличается. В открытых источниках не удалось найти формального описания поддерживаемых типов данных, а также способа привязки спецификации к функции.

При необходимости наложения ограничений на типы данных в языках с динамической типизацией к функциям дописывают контракт [13], нередко это делают с помощью комментариев специального вида. Например, такой подход используется в библиотеках PyContracts [14] и jsContract [15] для языков Python и JavaScript соответственно. В связи с тем, что инструменты ФПП программирования используют отличающиеся форматы описания типов данных, предложено с помощью специальных пометок внутри комментария задавать соответствие между спецификацией типов данных и инструментом. В связи с тем, что спецификация типа относится к параметрам функции, размещать ее решено непосредственно перед определением функции.

На рисунке 1.4 представлена спецификация универсального формата описания типов данных параметров функции (строка 1), а также продемонстрировано использование контрактов (строки 3-4) на примере функции vec_sum, рассмотренной в предыдущем примере. Контракт состоит из префикса, содержащего параметр <tool>, и спецификации. Префикс всегда начинается с последовательности символов «//**!**», позволяющей отличить контракт функции от обычного комментария, а параметр <tool> идентифицирует инструмент, для которого предназначена спецификация. Псевдонимы инструментальных средств, используемые в параметре <tool>, могут содержать

строчные буквы латинского алфавита, арабские цифры и символы подчёркивания. На данный момент поддерживаются псевдонимы opt и cpp для оптимизатора и преобразователя в императивную форму соответственно. Спецификация типов данных параметров функции, применяемая в контракте, должна быть однострочной.

```
1 //!<tool>:<specification>
2
3 //!opt:@{@(@int),@(@double)}->@(@double)
4 //!cpp:@{@(@int),@(@double)}->@(@double)
5 vec_sum << funcdef vecPair {
6   A << vecPair:1;
7   B << vecPair:2;
8   C << (A,B):#:[]:+;
9   return << C;
10}
11
```

Рисунок 1.4 – Использование контрактов для описания типов данных параметров функции

## 1.3 Разработка функциональных требований

### 1.3.1 Анализ функциональных требований

Требуется разработать консольное приложение, генерирующее файлы с описанием типов данных, используемых в инструментальных средствах ФПП программирования, формат которых определяется на основе распознанных комментариев-контрактов, задающих ограничение на типы входных и выходных параметров ФПП функций.

Для трансляции файлов с исходным кодом на языке Пифагор можно использовать опцию "-t", при этом результаты трансляции сохраняются в репозитории. К результатам относится не только набор РИГ и УГ, но и файл с исходным кодом отдельной функции. Анализ работы транслятора показал, что комментарии, расположенные перед телом функции, при трансляции не

удаляются, что позволяет использовать эти файлы для генерации описания типов данных аргументов функции.

В разрабатываемое инструментальное средство решено добавить поддержку следующего набора команд:

– *create*, для генерации файлов с описанием типов данных параметров функции на основе файла с исходным кодом;

– *update*, для обновления информации о типах данных ряда функций, хранящихся в репозитории;

– *add*, для непосредственного добавления заранее подготовленного файла с описанием типов данных в репозиторий;

– *show*, для вывода информации об имеющихся файлах с описанием типов данных;

– *help*, для отображения справочной информации об использовании данного инструментального средства.

Основной вариант использования системы – добавление информации о типах данных параметров функций с помощью команд *create*, *update* и *add*. Эти команды используют репозиторий ФПП функций для хранения и получения информации о каждой отдельной функции. Поэтому, прежде чем использовать данный инструмент, программист должен транслировать в РИГ все функции, которые он хочет снабдить статической информацией о типах, потому что репозиторий узнает о существовании функции только после её успешной трансляции.

Команда *create* создаёт файлы с описанием типов данных параметров функции согласно спецификации, указанной в комментарии-контракте. По умолчанию команда принимает список имён функций, для которых нужно сгенерировать файлы с описанием типов, при этом доступ к файлу с исходным кодом функции и размещение новых файлов с описанием типов осуществляются через репозиторий. Также у данной команды имеется опция *-i* или *--input*, указывающая программе работать напрямую с файлами вместо репозитория. Эта

опция добавлена для совместимости с существующим транслятором ФПП программ, который тоже умеет работать с файлами локально.

На ФПП языке программирования Пифагор написано множество программ и несколько библиотек [16; 17]. Многие программы можно снабдить информацией о типах данных. При этом использование команды *create* для генерации файлов с описанием типов будет неудобным, поскольку понадобиться вручную указать имена всех функций, у которых появилась информация о типах данных. Для решения подобных задач вводится команда *update*, позволяющая добавлять информацию о типах данных для множества функций сразу. Чтобы задать множество функций, подлежащих обработке, требуется указать пространство имен этих функций. Например, для обработки функций lib.math.fact, lib.math.power и так далее, достаточно указать их общее пространство имен lib.math. Для обновления информации о типах во всём репозитории нужно передать опцию --*all*.

Команда *add* нужна для добавления в репозиторий уже готового файла с описанием типов входных аргументов и возвращаемого значения функции. Данную команду следует использовать в случае, когда у функции нет файла с исходным кодом, а значит неоткуда взять информацию о типах. Такая ситуация возможна при использовании инструментальных средств, создающих файлы РИГ напрямую (например, при использовании оптимизатора).

Команда *show* отвечает за отображение информации о типах данных параметров функции. Информация берется из файлов, сгенерированных после выполнения предыдущих команд, доступ к которым производится с помощью репозитория или явного указания пути к файлу. У команды есть опция запуска --*tool*, позволяющая отфильтровывать выводимую информацию о типах данных по используемому инструментальному средству.

Команда *help* выводит краткую справку о доступных командах запуска данного инструментального средства. При передаче конкретной команды *help* выводит подробную инструкцию, связанную с этой командой.

### 1.3.2 Формализация функциональных требований

Функциональные требования к разрабатываемой системе формализованы в виде диаграммы вариантов использования [18], представленной на рисунке 1.5.


Рисунок 1.5 – Диаграмма вариантов использования системы

**Текстовое описание прецедентов:**

**Название прецедента:** создать описание типов данных.

**Цель сценария:** сгенерировать файл с описанием типов данных на основе файла с исходным кодом функции.

**Предусловия:** в репозитории есть файл с исходным кодом функции.

**Основной сценарий:**

1. Пользователь вводит команду *create*, после которой перечисляет через пробел полные имена функций, для которых нужно сгенерировать описание типов данных.

2. Система получает исходный код ФПП функции из репозитория.

3. Система находит контракты в исходном коде ФПП функции и обнаруживает в них описание типов данных параметров функции.

4. Система транслирует исходное описание типов данных в формат, специфичный для конкретного инструментального средства, указанного в контракте. Результат трансляции сохраняется в репозитории.

5. Система ничего не выводит в стандартный поток ошибок.

**Постусловия:** в репозитории появляется файл с описанием типов данных.

**Условие ввода в действие альтернативных сценариев**

**Условие 1.** Пользователь добавляет опцию *-i* или *--input*, после которой передает путь до файла с исходным кодом.

1. Система получает исходный код ФПП функции из файла, путь до которого был передан при запуске программы.

2. Система генерирует файл с описанием типов данных и сохраняет его в одну директорию с исходным кодом функции.

**Условие 2.** Файл с исходным кодом функции не найден.

1. Система выдает сообщение в стандартный поток ошибок.

**Название прецедента:** обновить описание типов данных в репозитории.

**Цель сценария:** рекурсивно обойти указанные директории с исходным кодом и обновить в них информацию о типах данных.

**Предусловия:** в директории с инструментальными средствами есть поддиректория repository.

**Основной сценарий:**

1. Пользователь вводит команду *update*, после которой перечисляет пространства имён функций, в которых нужно обновить информацию о типах данных каждой функции.

2. Система рекурсивно обходит все директории репозитория, соответствующие переданным пространствам имён, и находит все поддиректории, хранящие файлы ФПП функций.

3. Система формирует список ФПП функций, в директориях которых нет файлов с описанием типов данных.

4. Система генерирует описание типов данных для каждой функции из списка и сохраняет его в репозитории.

5. Система ничего не выводит в стандартный поток ошибок и возвращает управление пользователю.

**Условие ввода в действие альтернативных сценариев**

**Условие 1.** Вместо пространства имен передана опция *--all*.

1. Система обходит все директории репозитория.

**Условие 2.** Передана необязательная опция *-f* или *--force*.

1. Система создаст файлы с описанием типов данных, даже если они уже существуют.

**Условие 3.** В директории функции не найден файл с исходным кодом.

1. Система выдает предупреждение в стандартный выходной поток и переходит к следующей директории.

**Условие 4.** Директория repository не найдена.

1. Система выдает сообщение в стандартный поток ошибок и завершает работу с соответствующим кодом ошибки.

**Название прецедента:** добавить файл в репозиторий.

**Цель сценария:** разместить переданный файл в репозитории.

**Предусловия:** в репозитории существует целевая директория, в которую нужно записать файл.

**Основной сценарий:**

1. Пользователь вводит команду *add*, после которой передает путь до файла, в котором хранится описание типов данных функции, а также полное имя функции, которой нужно добавить это описание.

2. Система считывает описание типов из файла и сохраняет его в репозиторий для данной функции.

3. Система завершает работу без вывода сообщения в стандартный поток ошибок.

**Постусловия:** в директории указанной функции размещается копия переданного файла.

**Условие ввода в действие альтернативных сценариев**

**Условие 1.** В репозитории нет директории для указанной функции.

1. Система выдает сообщение в стандартный поток ошибок и завершает работу с ненулевым кодом возврата.

**Название прецедента:** посмотреть информацию о типах данных.

**Цель сценария:** вывести информацию о типах данных функции в стандартный выходной поток.

**Основной сценарий:**

1. Пользователь вводит команду *show*, после которой передается полное имя функции.

2. Система выводит в стандартный поток вывода информацию о типах данных, найденную в репозитории.

**Постусловия:** пользователь может ознакомиться с полученной информацией.

**Условие ввода в действие альтернативных сценариев**

**Условие 1.** Передана необязательная опция *--tool*.

1. Система отображает информацию о типах данных, предназначенную для конкретных инструментальных средств.

**Условие 2.** После имени функции передан относительный или абсолютный путь до директории.

1. Система будет искать информацию о типах не в репозитории, а в переданной директории.

**Условие 3.** В репозитории или переданной директории не найдены файлы с описанием типов данных или указаны неверный путь или имя функции.

1. Программа выдает соответствующее сообщение в стандартный поток ошибок и завершает работу с ненулевым кодом возврата.

**Название прецедента:** посмотреть инструкцию.

**Цель сценария:** отобразить пользователю справку по работе с программой.

**Основной сценарий:**

1. Пользователь вводит команду *help*.
2. Система выдает краткую справку по всем доступным командам.

**Условие ввода в действие альтернативных сценариев**

**Условие 1.** Пользователь передает одну из опций: *create*, *update*, *add*, *show*.

1. Система выдает подробную справку по конкретной команде.

### 1.4 Выводы по главе

Для некоторых инструментальных средств ФПП программирования нужно генерировать файлы с описанием типов данных входных аргументов и возвращаемого значения функций. Форматы описаний несовместимы между собой, потому что в них используются разные типы данных. Предложено ввести универсальный формат описания, с помощью которого можно указывать типы данных параметров функции непосредственно в исходном коде программ. Данный формат содержит специальный префикс, на основе которого разработанная система определяет в каком формате нужно генерировать файлы с описанием типов данных.

В настоящее время добавлена поддержка только одного формата описания, однако можно увеличить количество поддерживаемых форматов при помощи новых префиксов.

# 2 Проектирование

## 2.1 Модель предметной области системы

Приложение поддерживает набор команд, состоящих из позиционных аргументов и опций, передаваемых пользователем при запуске программы. Каждый аргумент может классифицироваться, как имя функции, пространство имен, или как путь к файлу. На рисунке 2.1 приведена модель предметной области разработанной программы в нотации диаграммы классов UML [19].

Кроме анализа ввода пользователя приложение выполняет работу, связанную с обработкой контрактов. Для этого оно взаимодействует с сущностью «Обработчик Контрактов», которая позволяет: **получать исходный код** ФПП функции с помощью репозитория или напрямую из файлов с помощью файловой системы операционной системы (ОС); **генерировать описание типов данных** параметров функций с помощью набора генераторов, специфичных для поддерживаемых форматов описаний; **сохранять описание типов данных** в репозитории или напрямую в текстовых файлах.

В настоящее время поддерживается только один формат описания типов данных для инструментального средства трансляции ФПП программ в императивную форму (**pfg2cpp**), однако существует возможность для добавления новых форматов. На рисунке это отражено с помощью объекта «Другие генераторы», расширяющего возможности абстрактной сущности «Генератор описаний типов».

Контракт является связующим элементом между исходным кодом и описанием типов данных.

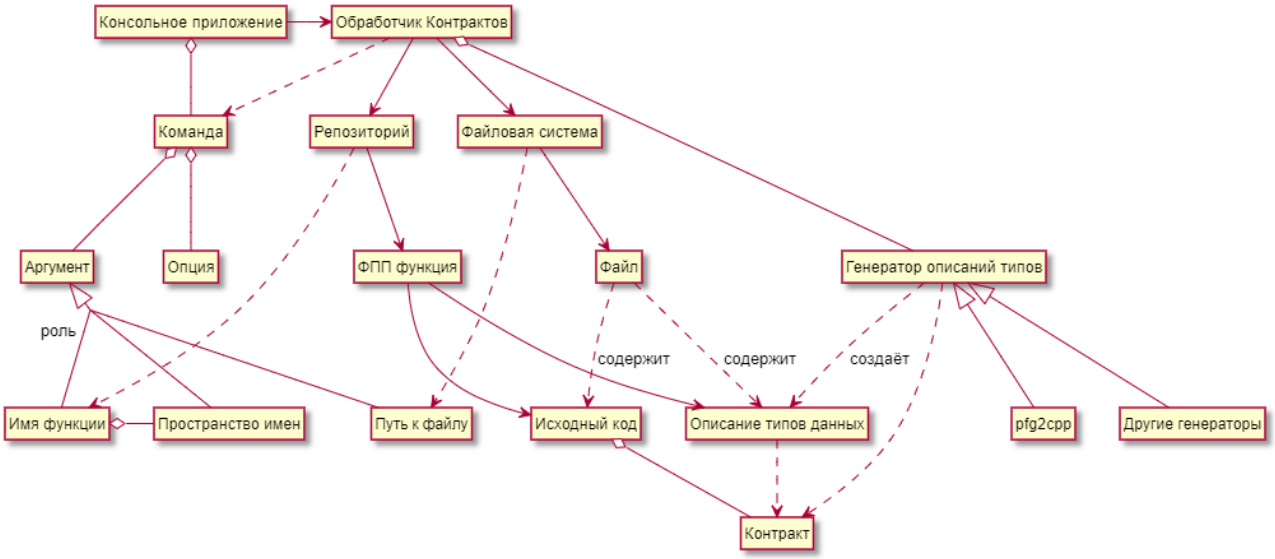


Рисунок 2.1 – Модель предметной области

## 2.2 Динамическая модель системы

Динамическая модель системы описывается с помощью диаграмм пригодности [20] и диаграмм последовательности [21], составленных для наиболее сложных прецедентов.

Для прецедента «создать описание типов данных» диаграммы пригодности и последовательности приводятся на рисунках 2.2 и 2.3 соответственно. На диаграмме пригодности видно, что логика, необходимая для реализации прецедента, инкапсулируется в следующих управляющих объектах:

- проанализировать опции;
- получить исходный код;
- найти контракты в исходном коде;
- сохранить описание типов данных;
- обработать ошибку.

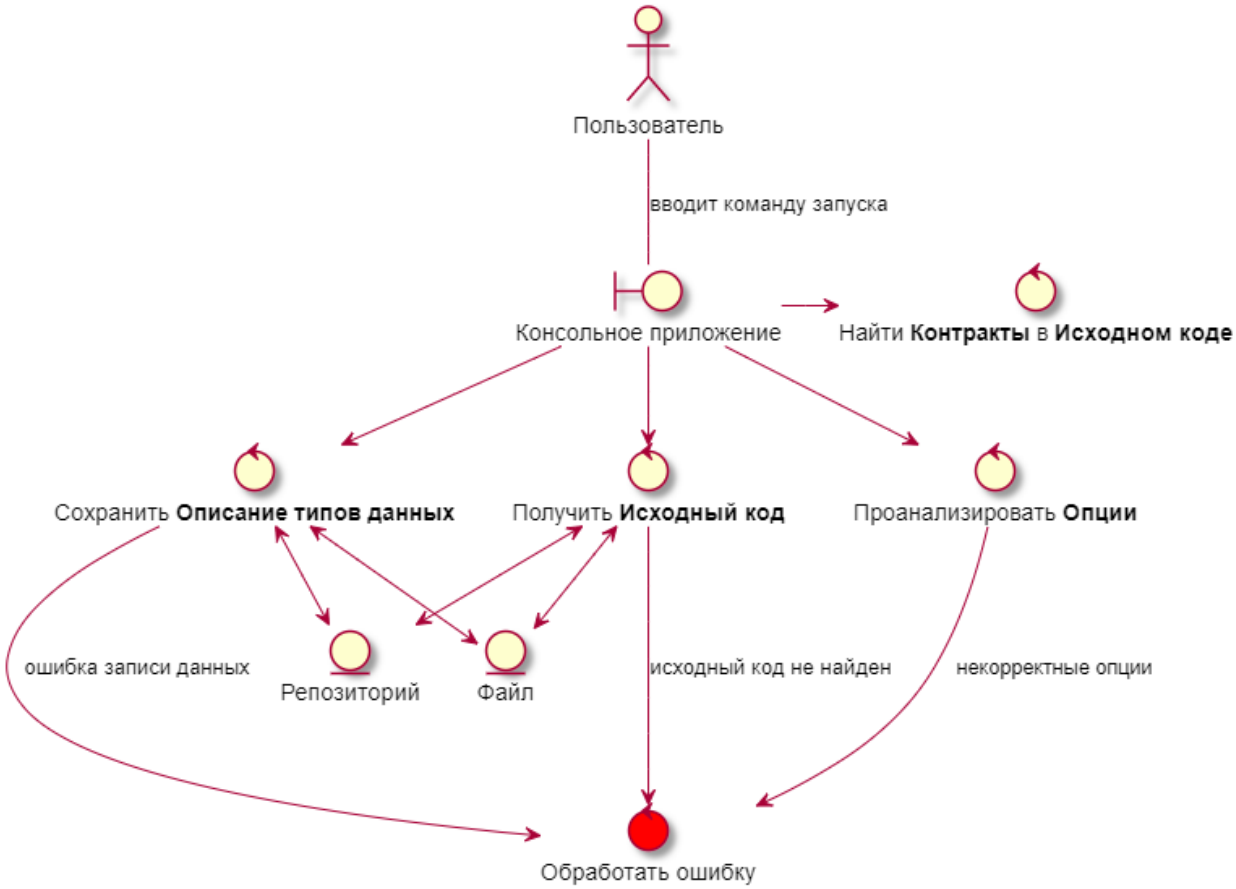


Рисунок 2.2 – Диаграмма пригодности прецедента «создать описание типов данных»

При этом большинство управляющих объектов диаграммы пригодности зависят от единственного граничного объекта, что приводит к смешиванию логики обработки пользовательского ввода и основной логики приложения. Для решения этой проблемы введён дополнительный управляющий объект фасада обработчика контрактов, скрывающий от граничного объекта логику, связанную с обработкой контрактов, за простым интерфейсом.

Дальнейший анализ диаграммы пригодности позволил выявить конкретные функции, необходимые для реализации прецедента. Диаграмма последовательности на рисунке 2.3 отражает, как эти функции были распределены между имеющимися объектами. На этой диаграмме видно, что по умолчанию программа использует репозиторий ФПП функций для хранения информации о типах данных, однако имеется возможность передать опцию *--input*, чтобы начать работать с файлами напрямую.

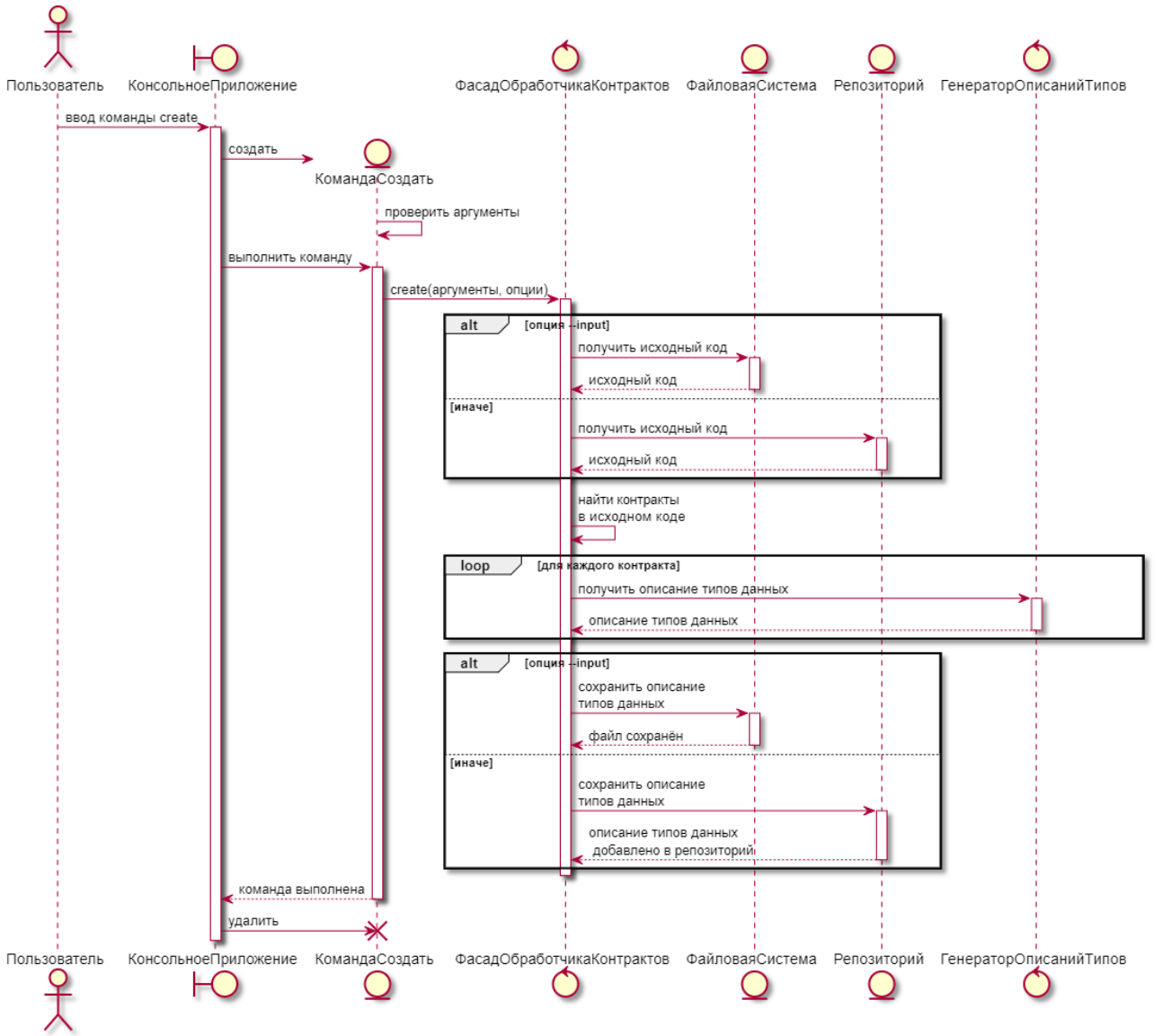


Рисунок 2.3 – Диаграмма последовательности прецедента «создать описание типов данных»

На рисунках 2.4 и 2.5 представлены диаграммы пригодности и последовательности прецедента «обновить описание типов данных в репозитории» соответственно. На диаграмме пригодности видно, что некоторые управляющие объекты уже встречались в диаграмме пригодности предыдущего прецедента (рисунок 2.2). Это говорит о том, что поведение, реализуемое данными объектами, полностью совпадает в обоих прецедентах. При этом следует учесть, что управляющие объекты «получить исходный код» и «сохранить описание типов данных», изображенные на рисунке 2.4, не взаимодействуют с файловой системой напрямую, как это делают аналогичные

объекты из предыдущего прецедента. Данное различие заметно на диаграммах последовательности (отсутствие блока alt).

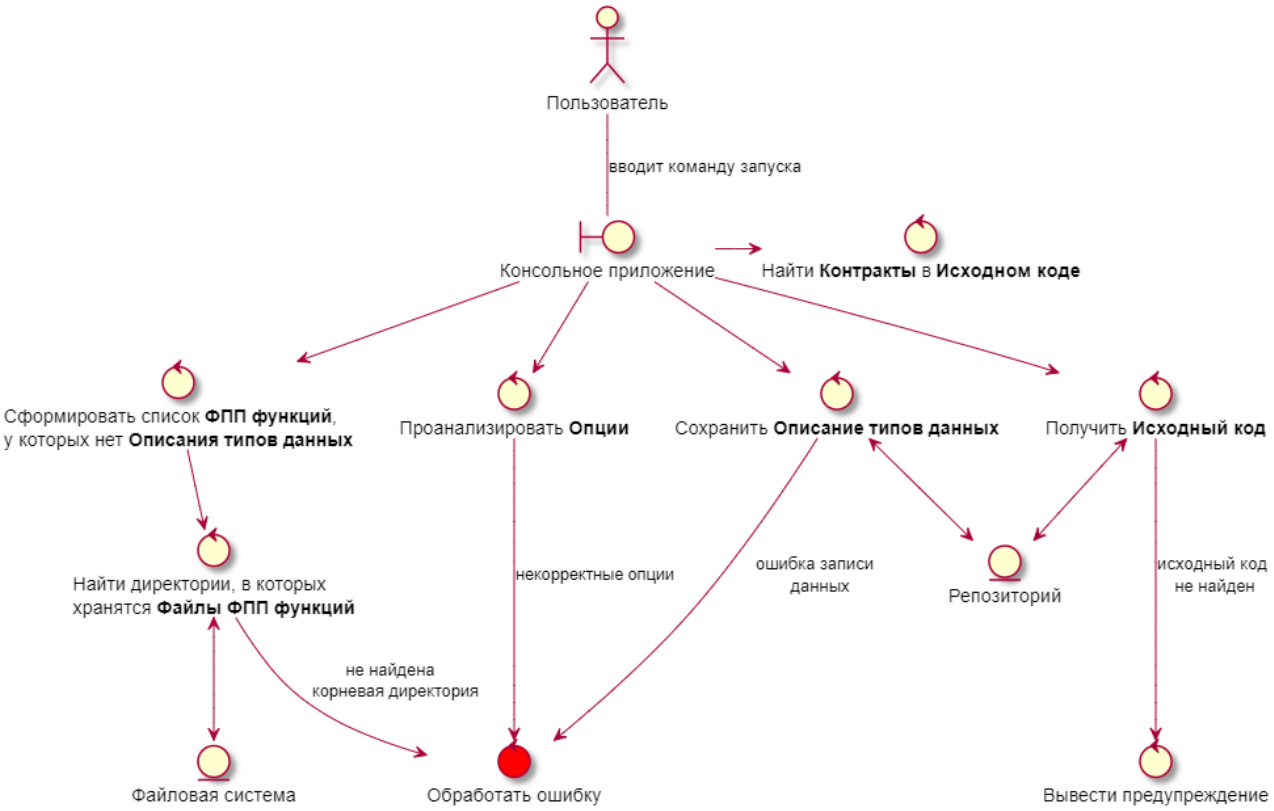


Рисунок 2.4 – Диаграмма пригодности прецедента «обновить описание типов данных в репозитории»

Помимо этого, на диаграмме пригодности, изображенной на рисунке 2.4, появились два новых управляющих объекта, связанных с поиском в репозитории всех ФПП функций, не имеющих описания типов данных. Однако существующая реализация репозитория не предоставляет возможности поиска ФПП функций по заданному критерию, поэтому принято решение самостоятельно решить данную задачу с помощь рекурсивного обхода директорий репозитория с использованием вспомогательного объекта файловой системы. Для снижения времени работы команды пользователь может дополнительно передать название пространства имён, в котором следует искать функции и создавать файлы с описанием типов данных. Кроме того, время работы команды уменьшается за счёт того, что по умолчанию программа не

обновляет существующие файлы с описанием типов данных, но пользователь может явно потребовать заменять имеющиеся файлы с помощью опции *--force*.

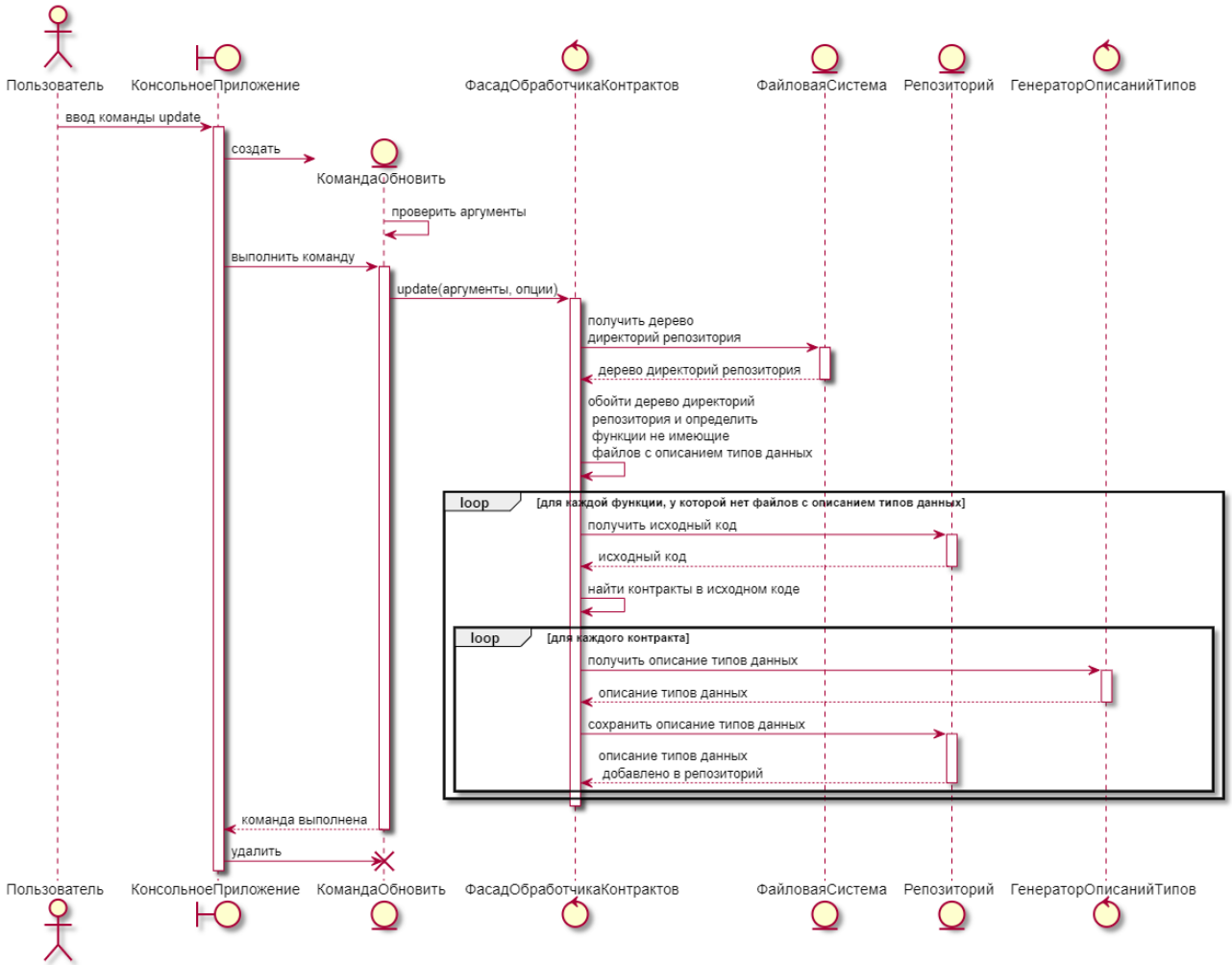


Рисунок 2.5 – Диаграмма последовательности прецедента «обновить описание типов данных в репозитории»

На рисунках 2.6 и 2.7 представлены диаграммы пригодности и последовательности прецедента «добавить файл в репозиторий» соответственно. Из диаграммы пригодности видно, что в данном прецеденте не используются новые управляющие объекты, следовательно нам достаточно функций, выделенных в ходе анализа предыдущих диаграмм динамической модели системы. На диаграмме последовательности показано, что данная команда не работает с исходным кодом ФПП функции, вместо этого она считывает заранее подготовленный файл с описанием типов данных и сохраняет его в репозитории.

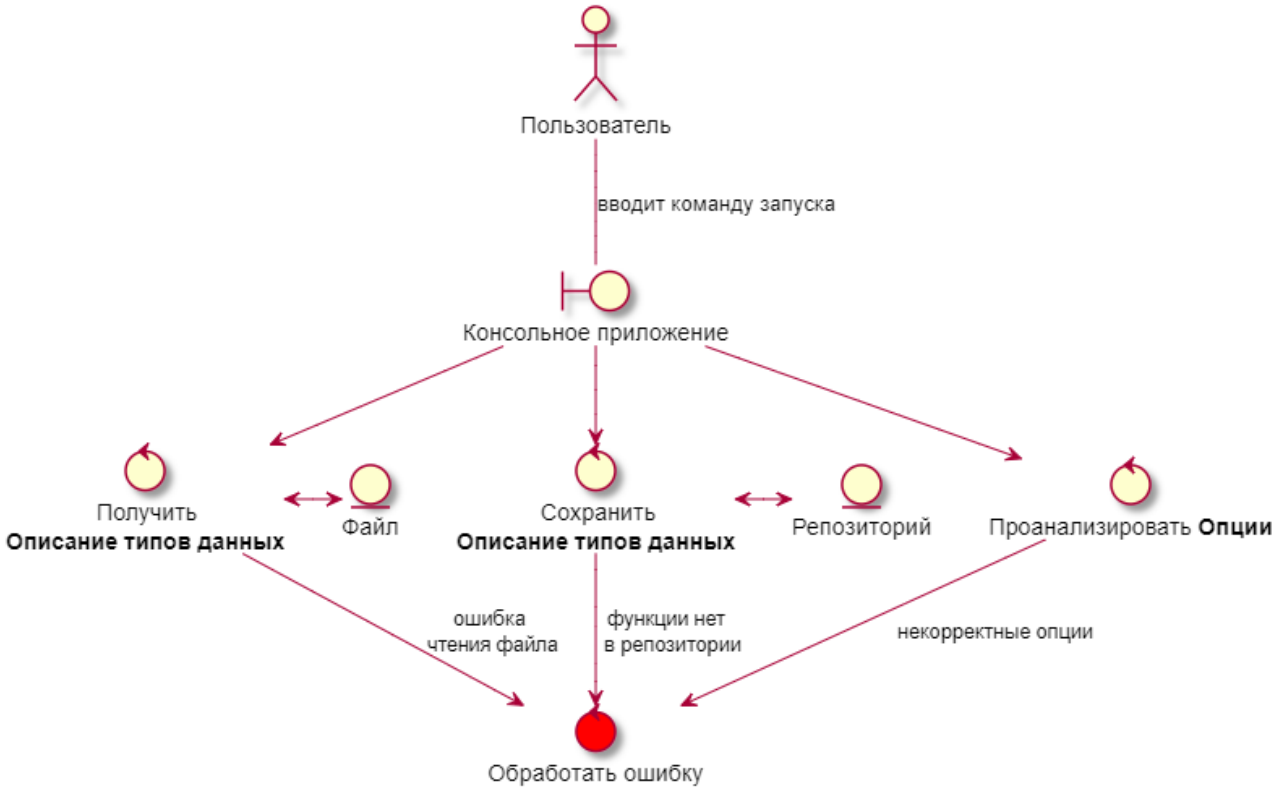


Рисунок 2.6 – Диаграмма пригодности прецедента «добавить файл в репозиторий»

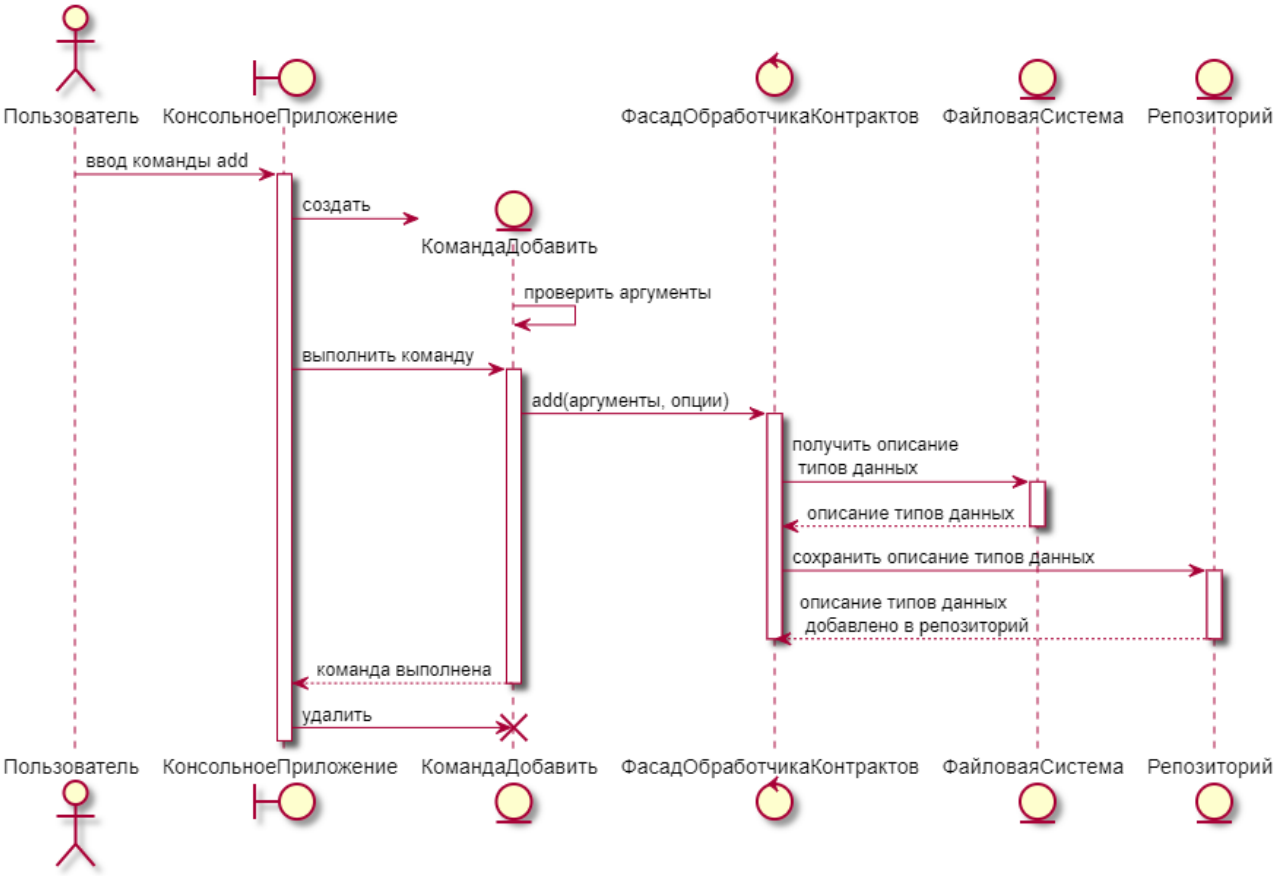


Рисунок 2.7 – Диаграмма последовательности прецедента «добавить файл в репозиторий»

### 2.3 Статическая модель системы

В ходе анализа динамической модели системы решено разделить сложное поведение программы на два логических модуля – **модуль анализа ввода пользователя** и **модуль обработки контрактов**.

На рисунке 2.8 представлена диаграмма классов модуля обработки контрактов. Данный модуль включает в себя набор классов, отвечающих за решение отдельных подзадач, возникающих в процессе обработки контрактов. Для уменьшения количества зависимостей классов модуля от внешнего кода добавлен класс «ФасадОбработчикаКонтрактов», реализующий структурный шаблон проектирования «Фасад» [22], скрывающий от вызывающего кода детали взаимодействия между элементами подсистемы с помощью простого интерфейса. Класс «Репозиторий» предоставляет простой интерфейс взаимодействия с существующим репозиторием для получения и хранения текстов исходного кода и описания типов данных параметров функций. Класс «ФайловаяСистема» решает вспомогательные задачи по предоставлению доступа к локальным файлам и поиску в дереве директорий файловой системы ОС. Классы «ОбработчикКонтрактов» и «ГенераторОписанийТипов» используются для поиска контрактов в исходном коде и формирования описания типов данных параметров ФПП функций. Реализация этих классов основывается на поведенческом шаблоне проектирования «Издатель-подписчик» [23], обеспечивающего исполнение **главной задачи проектирования** всей системы – получение модульной структуры для поддержки форматов описаний типов данных сторонних инструментов. В динамике процесс получения информации о типах данных выглядит следующим образом:

1. Внешний код регистрирует новые генераторы описаний типов данных у объекта класса «ФасадОбработчикаКонтрактов», передавая ему экземпляры классов, наследованных от абстрактного класса «ГенераторОписанийТипов».

2. Объект фасада подписывает полученные генераторы на объект обработчика контрактов, который сохраняет ссылки на генераторы в

ассоциативном массиве, где в качестве ключей используются псевдонимы генераторов.

3. Когда объект обработчика контрактов находит очередной контракт внутри исходного кода, он вызывает метод «обработатьОписаниеТипов()» для всех подписчиков, хранящихся в ассоциативном массиве, чей псевдоним совпадает с именем инструмента из контракта.

4. Обработчик контрактов возвращает фасаду текст полученного описания типов данных и имя файла, в котором оно будет храниться.

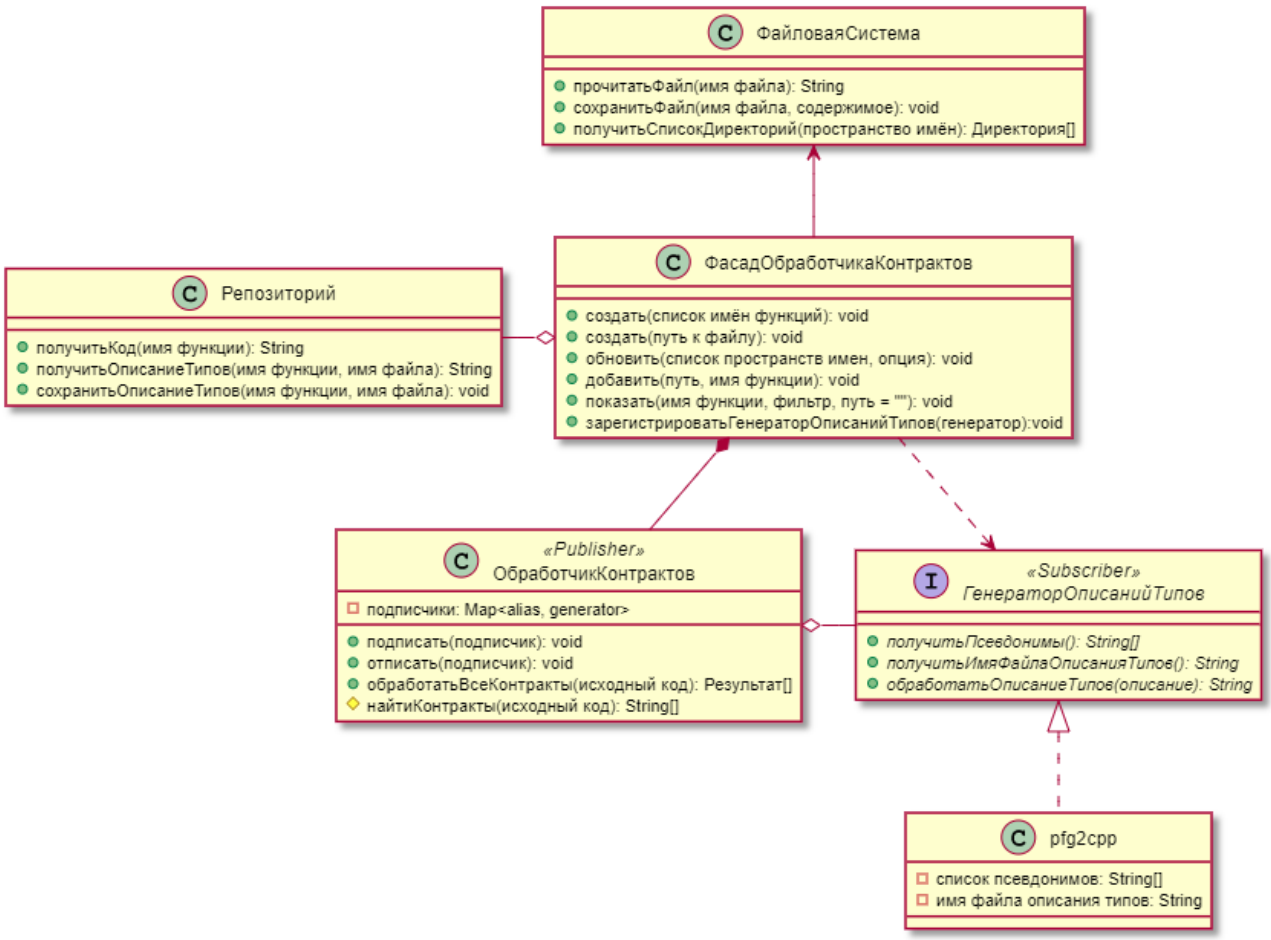


Рисунок 2.8 – Диаграмма классов модуля обработки контрактов

Описанная выше реализация модуля обработки контрактов обладает следующими преимуществами:

– модуль можно легко интегрировать в другие инструментальные средства с помощью подключения класса фасада;

– новые генераторы поддерживающие другие форматы описания типов данных можно добавлять динамически;

– критическая ошибка в каком-либо генераторе не приводит к аварийному завершению программы, что позволяет продолжить обработку других контрактов даже в том случае, если обработка предыдущего контракта привела к ошибке.

На рисунке 2.9 представлена диаграмма классов модуля анализа ввода пользователя. Для анализа ввода пользователя применяется упрощенная реализация поведенческого шаблона проектирования «Команда» [24]. На диаграмме показано, что для каждой команды запуска приложения существует свой класс, анализирующий переданные аргументы и опции. Аргументы команды представлены в виде набора шаблонных классов. После инициализации и разбора ввода пользователя, можно приступать к выполнению команды. Для этого каждая команда предоставляет метод «выполнить()», вызывающий подходящий метод фасада. Класс «CommandHelper» предоставляет доступ к статическим данным команды (имя, справка и т. д.), а также позволяет создавать объекты команд на основе ввода пользователя.

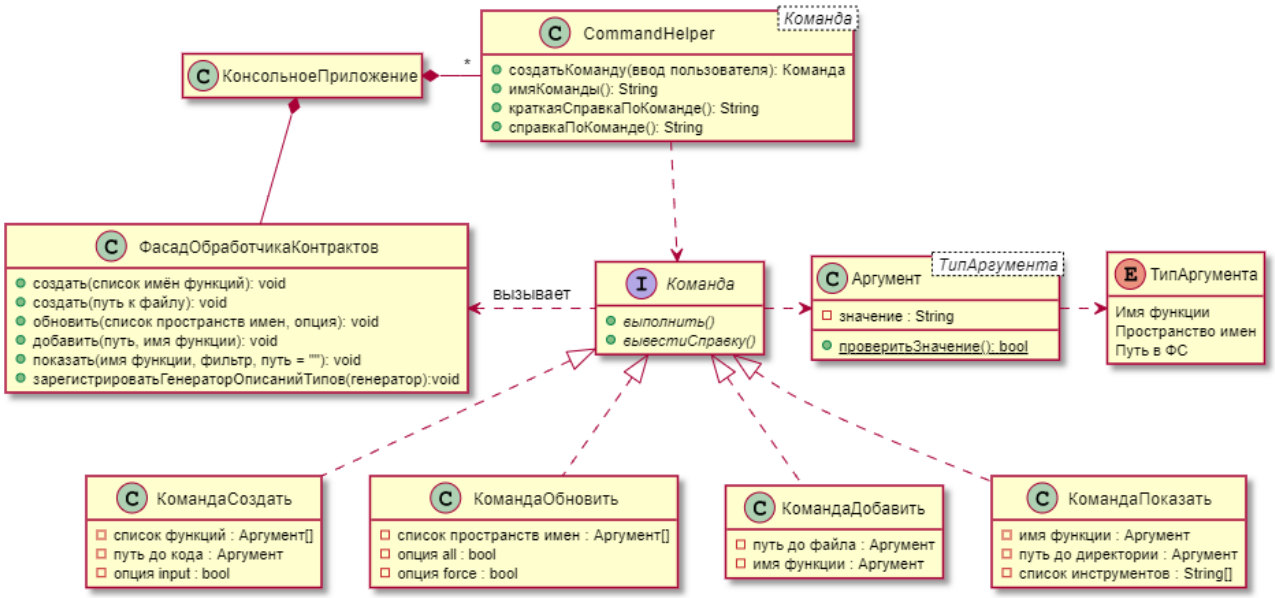


Рисунок 2.9 – Диаграмма классов модуля анализа ввода пользователя

## 2.4 Выводы по главе

В соответствии со спецификацией требований:

– **Составлены** модель предметной области решаемой задачи, а также динамическая и статическая модели системы.

– **Спроектирована** модульная архитектура приложения, поддерживающая динамическое добавление новых форматов описаний типов данных и независимую обработку ошибок времени выполнения.

– **Разработаны** диаграммы классов уровня проектирования, позволяющие перейти к реализации инструментального средства.

# 3 Реализация и интеграция с существующими инструментами

## 3.1 Модули обработки контрактов и анализа ввода пользователя

Существующие инструментальные средства ФПП программирования разработаны на языке программирования C++ с использованием фреймворка Qt [25]. В ряде случаев для интеграции разработанного инструмента необходимо использовать созданные ранее модули. Например, модуль репозитория применяется для получения исходного кода функций, а также хранения файлов с описанием их типов данных. В связи с этим в реализации настоящей работы используется аналогичный стек технологий.

Для поиска контрактов в исходном коде, получения спецификации описания типов данных параметров функции и анализа ввода пользователя применяются регулярные выражения. Модуль QtCore фреймворка Qt предоставляет набор классов для работы с ними. В частности, методы *processAllContracts* и *findContracts* класса ContractsHandler и метод *isValidFunctionName* класса FunctionNameArgument используют классы QRegularExpression и QRegularExpressionMatch.

Разработанный класс filesystem, используемый для обхода директорий репозитория и поиска файлов с описанием типов данных, реализован с помощью библиотеки std::filesystem. Класс std::filesystem::path используется для работы с путями файловой системы, при этом данный класс абстрагируется от особенностей специфичных для некоторых ОС, а также поддерживается классом QFile, начиная с версии Qt 6.0. Для рекурсивного обхода директорий репозитория используется класс std::filesystem::recursive_directory_iterator.

## 3.2 Обработка ошибок

Обработка ошибок в разработанном инструментальном средстве осуществляется с помощью механизма исключений C++. В большинстве случаев обработка исключений заключается в информировании пользователя о

возникновении исключительной ситуации с помощью функций *printMessage* и *printError*, при этом если ошибка мешает дальнейшему выполнению программы, то она экстренно завершит свою работу с ненулевым кодом возврата. В таблице 3.1 приводятся классы исключений, которые могут возникнуть при исполнении определенных функций.

Таблица 3.1 – Перечень выбрасываемых исключений

| **Имя функции / метода** | **Выбрасываемое исключение** |
|---|---|
| FunctionNameArgument::FunctionNameArgument | std::invalid_argument |
| NamespaceArgument::NamespaceArgument | std::invalid_argument |
| PathArgument::PathArgument | std::invalid_argument |
| ConsoleApp::processCommand | std::invalid_argument |
| ContractsHandlerFacade::show | std::invalid_argument |
| FileSystem::readFile | std::system_error |
| FileSystem::saveFile | std::system_error |
| ShowCommand:: parseToolOption | std::invalid_argument |

### 3.3 Интернационализация

В инструментальном средстве обработки контрактов ФПП программ реализована команда вывода справочной информации о поддерживаемых командах, их опциях запуска и принимаемых аргументах, кроме того, программа может выдавать пользователю информационные сообщения и сообщения об ошибках. Для перевода этого текста на другие языки применяются стандартные средства фреймворка Qt, а именно:

– статические методы *translate* классов QObject и QCoreApplication, в которые передаётся текст, подлежащий переводу;

– утилита lupdate, для формирования файлов переводов;

– приложение Linguist, для редактирования файлов переводов.

Всего добавлена поддержка двух языков – русского и английского. Язык выводимого текста определяется автоматически во время запуска программы на

основе языка операционной системы, то есть при запуске программы на ОС с русской локализацией используется русский перевод, иначе – английский.

В сценарий сборки инструментального средства добавлено правило для генерации двоичных файлов переводов, подключаемых при запуске программы. Таким образом, при сборке проекта не требуется предварительно запускать утилиту lrelease, что также облегчает последующую интеграцию программы в систему ФПП программирования.

### 3.4 Платформо-зависимые части системы

При добавлении поддержки русского языка было обнаружено, что приложение командной строки ОС Windows не поддерживает Unicode, поэтому для работы с данной ОС требуется изменить стандартную кодировку (UTF-8), применяемую в Qt при выводе текста. Отсюда появилась потребность в функциях, кодирующих и декодирующих текст во внутренний формат хранения строк (UTF-16 для класса QString) и формат текущей ОС. Для решения такой задачи добавлен заголовочный файл platform_specific.h, содержащий функции, реализация которых зависела от используемой ОС. Платформо-зависимые реализации данных функций представлены в файлах platform_specific_win.cpp и platform_specific_unix.cpp для Windows и Unix-подобных ОС соответственно. В файл проекта добавлено правило, исключающее из сборки файлы, не предназначенные для целевой ОС.

Помимо функций, работающих с кодировкой, файл platform_specific.h содержит функции, отвечающие за вывод форматированных сообщений в консоль. Например, функция printError не просто печатает текст ошибки, но и делает его красным. Для реализации подсветки текста в функции printError в Unix-подобных системах используются последовательности управляющих символов, а в Windows – API консоли.

### 3.5 Модульное тестирование

Для написания модульных тестов использовался набор классов и макроопределений из модуля Qt Test фреймворка Qt. В результате тестирования разработано две программы, одна из которых выполняет проверку методов класса обработчика контрактов, реализующего поиск и обработку контрактов, а вторая – тестирует корректность методов, анализирующих ввод пользователя.

В таблице 3.2 приводится полный перечень методов, для которых проводилось тестирование, а также указано количество добавленных тестов. В приложении А имеется инструкция по сборке и запуску тестирующих программ.

Таблица 3.2 – Перечень методов, для которых проводилось тестирование

| **Имя класса** | **Имя метода** | **Количество модульных тестов** |
|---|---|---|
| ContractsHandler | findContracts | 7 |
| ContractsHandler | formSpecificationFileName | 3 |
| ContractsHandler | processAllContracts | 4 |
| FunctionNameArgument | isValidFunctionName | 6 |
| NamespaceArgument | isValidNamespace | 4 |
| PathArgument | PathArgument | 3 |

### 3.6 Интеграция с существующими инструментами

Разработка системы ФПП программирования ведется с использованием распределенной системы контроля версий (СКВ) Git [26]. Удаленный репозиторий проекта [27] располагается на git-хостинге Bitbucket [28]. С помощью веб-интерфейса Bitbucket было создано ответвление (fork) от основного проекта, в которое вносились все изменения.

На рисунке 3.1 изображен граф ответвленного проекта после завершения работы над системой контрактов. Закрашенные круги, обозначают коммиты веток master и pfg2cpp исходного репозитория. Поскольку система контрактов в первую очередь разрабатывалась для инструментальных средств оптимизации и трансляции кода в C++, новые коммиты добавлялись в ветки pfg2cppContracts и pfg2cppPortingToQt6 (новые коммиты изображены в виде незакрашенных кругов с непрерывной линией).

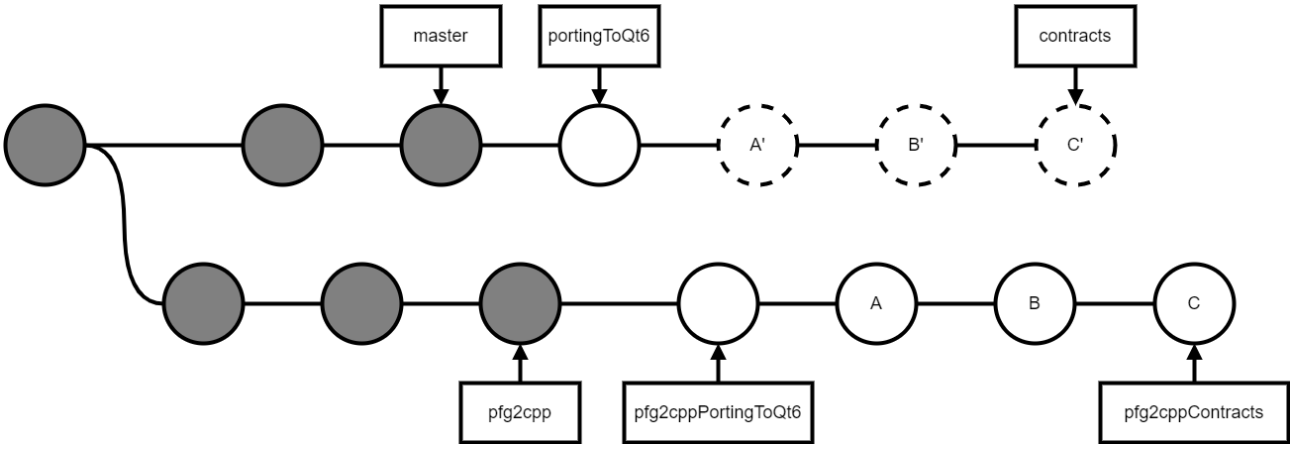


Рисунок 3.1 – Граф git репозитория после реализации системы контрактов

Интеграция системы контрактов заключается в добавлении коммитов A, B и C в основную ветку разработки master, однако простое слияние ветки pfg2cppContracts также добавит коммиты из ветки pfg2cpp, что не соответствует желаемому поведению. Поэтому изменения, внесенные в коммитах A, B и C, будут добавлены в ветку master в виде последовательности патчей (A’, B’ и C’ на рисунке 3.1). Для этого были выполнены следующие команды (рисунок 3.2).

```
1 git checkout portingToQt6
2 git checkout -b contracts
3 git cherry-pick pfg2cppPortingToQt6..pfg2cppContracts
```

Рисунок 3.2 – Команды для добавления сделанных изменений в ветку contracts

Первая команда, представленная на рисунке 3.2, переключает активную ветку на portingToQt6, вторая создаёт ветку contracts и переключается на нее,

третья вычисляет изменения внесенные в коммитах A, B и C и добавляет их в ветку contracts.

Для внесения изменений из ветки contracts ответвленного репозитория в ветку master исходного репозитория системы ФПП программирования создан запрос на слияние (pull request) с помощью веб интерфейса Bitbucket.

### 3.7 Выводы по главе

1. Реализовано инструментальное средство системы контрактов для ФПП языка программирования Пифагор.
2. Добавлены модульные тесты, проверяющие корректность созданной системы.
3. Приложение локализовано на английский и русский языки.
4. Система контрактов интегрирована в существующий набор инструментальных средств ФПП программирования.
5. Составлена инструкция по сборке разработанной системы и запуску тестирующих программ.

# ЗАКЛЮЧЕНИЕ

Спроектирована система поддержки контрактного программирования для ФПП языка Пифагор, позволяющая автоматизировать процесс создания информации о типах данных для различных инструментов ФПП программирования непосредственно на основе исходного кода программы. Такой подход упрощает процесс программирования и сокращает время разработки.

Инструментальное средство имеет гибкую архитектуру, которая позволяет с минимальными усилиями и без модификации существующего кода добавлять поддержку новых инструментов, использующих статическую информацию о типах данных.

После реализации система протестирована на программах, содержащих контракты, из модуля трансформации функционально-потоковых программ в императивную форму.

## СПИСОК СОКРАЩЕНИЙ

ИСР – интегрированная среда разработки

ОС – операционная система

ПВС – параллельная вычислительная система

ПО – программное обеспечение

РБНФ – расширенная форма Бэкуса-Наура

РИГ – реверсивный информационный граф

СБИС – сверхбольшая интегральная схема

СКВ – система контроля версий

УГ – управляющий граф

ФПП – функционально-потоковая парадигма

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Легалов, А. И. Функциональный язык для создания архитектурно-независимых параллельных программ / А. И. Легалов // Вычислительные технологии. – 2005. – Т. 10. – № 1. – С. 71-89.

2. Свидетельство о государственной регистрации программы для ЭВМ № 2018666577 Российская Федерация. Программа для трансляции функционально-потокового языка параллельного программирования : № 2018663780 : заявл. 03.12.2018 : опубл. 18.12.2018 / И. В. Матковский, А. И. Легалов, В. С. Васильев, А. И. Постников ; заявитель Федеральное государственное автономное образовательное учреждение высшего образования «Сибирский федеральный университет» (СФУ).

3. Свидетельство о государственной регистрации программы для ЭВМ № 2020612517 Российская Федерация. Программный интерпретатор функционально-потокового языка параллельного программирования : № 2020611538 : заявл. 14.02.2020 : опубл. 25.02.2020 / А. И. Легалов, И. В. Матковский ; заявитель Федеральное государственное автономное образовательное учреждение высшего образования «Сибирский федеральный университет» (СФУ).

4. Свидетельство о государственной регистрации программы для ЭВМ № 2021611582 Российская Федерация. Транслятор архитектурно-независимого описания автоматных и комбинационных схем : № 2021610682 : заявл. 25.01.2021 : опубл. 01.02.2021 / О. В. Непомнящий, Д. С. Романова, И. Н. Рыженко, А. И. Легалов ; заявитель Федеральное государственное автономное образовательное учреждение высшего образования «Сибирский федеральный университет» (СФУ).

5. Свидетельство о государственной регистрации программы для ЭВМ № 2021662788 Российская Федерация. Программа преобразования реверсивных информационных графов языка ПИФАГОР в программы на языке С++ : № 2021661762 : заявл. 26.07.2021 : опубл. 04.08.2021 / В. С. Васильев, О. В.

Непомнящий ; заявитель Федеральное государственное автономное образовательное учреждение высшего образования «Сибирский федеральный университет».

6. Свидетельство о государственной регистрации программы для ЭВМ № 2018666434 Российская Федерация. Программа для оптимизации функционально-потокового языка параллельного программирования : № 2018663789 : заявл. 03.12.2018 : опубл. 17.12.2018 / В. С. Васильев, А. И. Легалов, И. В. Матковский, О. В. Непомнящий ; заявитель Федеральное государственное автономное образовательное учреждение высшего образования «Сибирский федеральный университет» (СФУ).

7. Васильев, В. С. Трансформация функционально-потоковых параллельных программ в императивные / В. С. Васильев, А. И. Легалов, С. В. Зыков // Моделирование и анализ информационных систем. – 2021. – Т. 28. – № 2. – С. 198-214.

8. Инструментальная поддержка создания и трансформации функционально-потоковых параллельных программ / А. И. Легалов, В. С. Васильев, И. В. Матковский, М. С. Ушакова // Труды Института системного программирования РАН. – 2017. – Т. 29. – № 5. – С. 165-184.

9. Легалов, А. И. Статически типизированная версия языка функционально-потокового параллельного программирования / А. И. Легалов, И. А. Легалов, И. В. Матковский // Параллельные вычислительные технологии (ПаВТ'2020) : Короткие статьи и описания плакатов, Пермь, 31 марта – 02 2020 года. – Пермь: Издательский центр ЮУрГУ, 2020. – С. 185-192.

10. Легалов, А. И. Особенности хранения функционально-потоковых параллельных программ / А. И. Легалов, И. В. Матковский, А. В. Анкудинов // Вестник Сибирского государственного аэрокосмического университета им. академика М.Ф. Решетнева. – 2013. – № 4(50). – С. 53-57.

11. Ушакова, М. С. Инструментальная поддержка формальной верификации программ, написанных на языке функционально-потокового параллельного программирования / М. С. Ушакова, А. И. Легалов // Вестник

Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. – 2015. – Т. 4. – № 2. – С. 58-70.

12. Непомнящий, О. В. Методы, алгоритмы и программные инструменты архитектурно независимого высокоуровневого синтеза однокристальных цифровых систем / О. В. Непомнящий, И. Н. Рыженко, А. И. Легалов // Суперкомпьютерные технологии (СКТ-2018) : Материалы 5-й Всероссийской научно-технической конференции, Дивноморское, Геленджик, 17–22 сентября 2018 года. – Дивноморское, Геленджик: Южный федеральный университет, 2018. – С. 104-109.

13. Старолетов, С. М. Основы тестирования и верификации программного обеспечения : учебное пособие / С. М. Старолетов. – Санкт-Петербург : Лань, 2018. – 344 с.

14. PyContracts // GitHub: Where the world builds software [сайт]. – URL: https://github.com/AlexandruBurlacu/pycontracts (дата обращения: 27.05.2022).

15. jsContract // GitHub: Where the world builds software [сайт]. – URL: https://github.com/oyvindkinsey/jsContract (дата обращения: 27.05.2022).

16. Удалова, Ю. В. Библиотека математических функций для языка функционально-потокового параллельного программирования Пифагор / Ю. В. Удалова // Вестник Бурятского государственного университета. Математика, информатика. – 2019. – № 4. – С. 57-64.

17. Удалова, Ю. В. Библиотека обработки строк для языка функционально-потокового параллельного программирования Пифагор / Ю. В. Удалова // Международный научно-исследовательский журнал. – 2020. – № 12-1(102). – С. 83-87.

18. Фаулер, М. UML. Основы : краткое руководство по стандартному языку объектного моделирования / М. Фаулер, К. Скот; пер. с англ. – СПб.: Символ – Плюс, 2002. – 192 с. – ISBN 5-93286-060-X.

19. Васильев, В. Диаграммы классов UML / Блог программиста – программирование и алгоритмы : [сайт]. – 2017. – 7 июн. – URL: https://pro-prof.com/archives/3212 (дата обращения: 27.05.2022).

20. Васильев, В. Процесс ICONIX. Диаграммы пригодности / Блог программиста – программирование и алгоритмы : [сайт]. – 2016. – 29 апр. – URL: https://pro-prof.com/archives/2723 (дата обращения: 27.05.2022).

21. Васильев, В. Основы UML. Диаграммы последовательности / Блог программиста – программирование и алгоритмы : [сайт]. – 2016. – 19 июл. – URL: https://pro-prof.com/archives/2769 (дата обращения: 27.05.2022).

22. Фасад // Рефакторинг и Паттерны проектирования : [сайт]. – URL: https://refactoring.guru/ru/design-patterns/facade (дата обращения: 27.05.2022).

23. Наблюдатель // Рефакторинг и Паттерны проектирования : [сайт]. – URL: https://refactoring.guru/ru/design-patterns/observer (дата обращения: 27.05.2022).

24. Команда // Рефакторинг и Паттерны проектирования : [сайт]. – URL: https://refactoring.guru/ru/design-patterns/command (дата обращения: 27.05.2022).

25. Qt Кроссплатформенная разработка для встроенных и настольных систем : официальный сайт. – URL: https://www.qt.io/ (дата обращения: 27.05.2022).

26. Git : официальный сайт. – URL: https://git-scm.com/ (дата обращения: 27.05.2022).

27. Pifagor // Bitbucket : репозиторий проекта. – URL: https://bitbucket.org/alpha900i/pifagor/ (дата обращения: 27.05.2022).

28. Bitbucket : Решение Git для команд, использующих Jira : официальный сайт. – URL: https://bitbucket.org/product/ru (дата обращения: 27.05.2022).

29. Install Qt // Download Qt: Get Qt Online Installer : официальный сайт. – URL: https://www.qt.io/download-qt-installer (дата обращения: 27.05.2022).

# ПРИЛОЖЕНИЕ А

## Инструкция по сборке проекта и запуску модульных тестов

1. Установить Qt. Для этого можно воспользоваться официальной программой онлайн установки [29]. Для Unix-подобных ОС достаточно настроек, предлагаемых установщиком по умолчанию, а для Windows нужно дополнительно выбрать модули Qt, отмеченные на рисунке А.1.

2. Установить СКВ Git воспользовавшись инструкцией с официального сайта проекта [26].

3. Собрать набор инструментальных средств ФПП программирования следуя командам, представленным на рисунках А.2 и А.3 для ОС Windows и GNU/Linux соответственно. На Windows команды следует выполнять в программе Qt 6.2.4 (MinGW 11.2.0), которая предоставляет командную оболочку с окружением, настроенным для работы и инструментами Qt.

4. Запустить исполняемые файлы *test_arguments* и *test_contractshandler*, расположенные в директории src/contracts/bin проекта, для проведения модульного тестирования разработанного приложения.

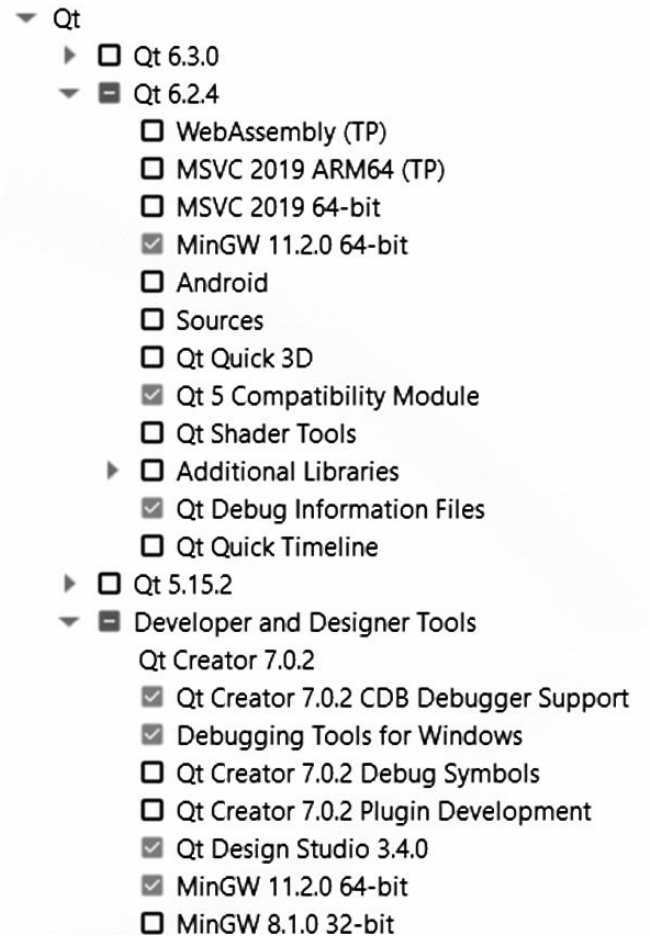


Рисунок А.1 – Модули Qt, подлежащие установке на ОС Windows

```
1 git clone https://bitbucket.org/alpha900i/pifagor.git
2 cd pifagor
3 mkdir build
4 cd build
5 qmake -spec win32-g++ ..\src\make_all.pro
6 mingw32-make
7 cd ..\bin
8 windeployqt contracts.exe
9 cd ..\src\contracts\bin
10windeployqt test_arguments.exe test_contractshandler.exe
```

Рисунок А.2 – Команды для сборки проекта в ОС Windows

```
1 git clone https://bitbucket.org/alpha900i/pifagor.git
2 cd pifagor
3 mkdir build
4 cd build
5 qmake ../src/make_all.pro
6 make
```

Рисунок А.3 – Команды для сборки проекта в ОС GNU/Linux

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
подпись
«20» 06 2022 г.

# БАКАЛАВРСКАЯ РАБОТА

09.03.01 – Информатика и вычислительная техника

Система контрактов для функционально-потокового языка параллельного программирования

Руководитель Васильев, 30.05.22 ст. преподаватель В.С. Васильев
подпись, дата

Выпускник Елисеенко 30.05.22 Д.А. Елисеенко
подпись, дата

Нормоконтролер Васильев, 30.05.22 ст. преподаватель В.С. Васильев
подпись, дата

Красноярск 2022