МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий
Кафедра параллельных вычислений

Направление подготовки 09.03.01 Информатика и вычислительная техника
Направленность (профиль): Программная инженерия и компьютерные науки

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Бочкарева Егора Николаевича

Тема работы:

РАЗРАБОТКА СИСТЕМЫ УПРАВЛЕНИЯ БАЗОЙ ФРАГМЕНТОВ КОДА ДЛЯ СИСТЕМЫ АКТИВНЫХ ЗНАНИЙ

«К защите допущена»
Заведующий кафедрой,
д.т.н., проф.
Малышкин В.Э./……..
(ФИО) / (подпись)
«30» мая 2025г.

Руководитель ВКР
к.т.н,
доц. каф. ПВ ФИТ
Перепёлкин В. А./……...
(ФИО) / (подпись)
«30» мая 2025г.

Соруководитель ВКР
ст. преп. каф. ПВ ФИТ
Городничев М. А./……..
(ФИО) / (подпись)
«30» мая 2025г.

Новосибирск, 2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)
Факультет информационных технологий
Кафедра параллельных вычислений
(название кафедры)

Направление подготовки 09.03.01 Информатика и вычислительная техника
Направленность (профиль): Программная инженерия и компьютерные науки

УТВЕРЖДАЮ

Зав. кафедрой Малышкин В.Э.
(фамилия, И., О.)

…….
(подпись)
«21» октября 2024г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

Студенту Бочкареву Егору Николаевичу, группы 21201
(фамилия, имя, отчество, номер группы)

Тема: Разработка системы управления базой фрагментов кода для системы активных знаний
(полное название темы выпускной квалификационной работы)

утверждена распоряжением проректора по учебной работе от 21 октября 2024 №0377

Срок сдачи студентом готовой работы 20 мая 2025 г.

Исходные данные (или цель работы):

Разработка системы управления базой фрагментов кода для системы активных знаний

Структурные части работы:

Обзор существующих средств стандартизации и хранения фрагментов кода, выявление требований для системы управления базой фрагментов кода, разработка теоретического решения удовлетворяющего требованиям, описание реализации, тестирование

Руководитель ВКР
к.т.н,
доц. каф. ПВ ФИТ
Перепёлкин В.А. /……..
(ФИО) / (подпись)
«21» октября 2024г.

Задание принял к исполнению
Бочкарев Е.Н. /……..
(ФИО студента) / (подпись)
«21» октября 2024г.

Соруководитель ВКР
ст. преп. каф. ПВ ФИТ
Городничев М. А. /……..
(ФИО) / (подпись)
«21» октября 2024г.

СОДЕРЖАНИЕ

Определения, обозначения и сокращения **4**

Введение **5**

1 Обзор существующих средств хранения и стандартизации фрагментов кода **7**

1.1 Обзор родственных работ 7

1.1.1 SPIRAL 7

1.1.2 GitHub Copilot 7

1.1.3 Protégé 8

1.1.4 KNIME 8

1.1.5 Galaxy 9

1.2 Выводы 9

2 Выявление требований для системы активных знаний. Разработка теоретического решения **10**

2.1 Система активных знаний 10

2.2 Роль и задачи СУБФК 12

2.3 Предлагаемое решение 13

2.3.1 Модель фрагмента кода 16

3 Описание реализации и тестирование **17**

3.1 Хранилище фрагментов кода 17

3.2 Хранилище плагинов 19

3.3 Система управления 21

3.4 Обоснование и особенности реализации 24

3.5 Испытания 25

Заключение **28**

Список использованных источников и литературы **31**

Приложение А — Руководство пользователя **32**

Приложение Б — Описание программы **42**

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

СУБФК — Система управления базой фрагментов кода.

Фрагмент кода — В контексте СУБФК — любая программа, код, часть кода, или даже интернет сервис, объеденненная с описывающими ее нефункциональными свойствами, кооторую мы хотим автоматически переиспользовать. Не в контексте СУБФК — любая программа, код, часть кода, или даже интернет сервис.

Плагин — Совокупность подпрограммы СУБФК и соглашения для фрагментов кода, которое позволяет системе стандартизованно работать с определенным форматом фрагментов кода.

Пакет — Единица хранения фрагментов кода в СУБФК. Может быть несколько фрагментов кода в одном пакете, фрагмента кода без пакета быть не может.

ВВЕДЕНИЕ

Разработка прикладных параллельных программ на высокопроизводительных вычислительных системах — сложный и трудоёмкий процесс. Однако в каждой индустрии, где используются такие программы, со временем накапливается множество уже созданного программного обеспечения, и усилия по разработке постепенно заменяются усилиями по поиску и применению готовых решений. Например, человеку необходимо искать различные библиотеки подпрограмм, читать документацию к ним (чтобы понять, подходит ли библиотека под поставленные задачи), а также определять, какая из библиотек предпочтительнее — исходя из различных характеристик (насколько быстро работает программа, сколько потребляет памяти, поддерживает ли параллельное выполнение и так далее).

Чтобы использовать уже разработанную программу в новом контексте, требуется программист. Он должен разобраться, как работает программа, что она делает, что принимает на вход, что выдаёт на выход и в каком формате. Даже программист, хорошо разбирающийся в своей области, вынужден тратить усилия на изучение особенностей программ в конкретной предметной области, и при переходе от одной области к другой этот процесс приходится проходить заново.

На данный момент эту работу нельзя пропустить или полностью автоматизировать. Система активных знаний представляет собой программное решение, предназначенное для автоматизации «стыковки» различных программ, а также для автоматического создания параллельных программ специалистом в предметной области, не обладающим навыками программирования. Используя технологии, описанные в данной работе, человек, не умеющий писать код, может создать параллельную программу, которая выполняет то, что он описал с помощью интуитивно понятного языка графов [1, 2, 3].

Автоматизация создания параллельных программ позволяет сократить время разработки. Более того, при автоматизации исключается влияние человеческого фактора, что дополнительно снижает ресурсоёмкость процесса.

Система управления базой фрагментов кода (СУБФК) разрабатывается в составе системы активных знаний и отвечает за хранение, а также стандартизированное переиспользование сохранённых в системе программ. Стандартизация переиспользования — важный шаг на пути к автоматизации создания параллельных программ. На данный момент существует множество работ, которые напрямую или косвенно касаются автоматизации создания параллельных программ, в том числе работы, связанные с концепцией активных знаний. Основные труды включают в себя: [1, 4, 5, 6, 7].

На момент написания данной работы проблема автоматизации переиспользования фрагментов кода остается не решенной, она находится в стадии активной проработки. Существует некоторое количество реализаций, которые пытаются решить эту проблему, однако предложенные в рамках этих реализаций решения не покрывают потребности системы активных знаний. Существующие решения подробно рассмотрены в первой главе данной работы. В институте ИВМиМГ СО РАН разрабатывается первый прототип системы активных знаний, частью которого и будет являться разработанный прототип СУБФК.

Цель работы — разработка системы управления базой фрагментов кода в составе системы LuNA.

Задачи:

- Конкретизировать потребности системы активных знаний, а также специфицировать требования, предъявляемые к СУБФК.
- Проанализировать существующие решения, доступные для решения проблемы хранения и стандартизации переиспользования существующих фрагментов кода. При соответствии решений требованиям — выбрать лучшее и адаптировать это решение под потребности системы активных знаний.
- Разработать технические решения для создания СУБФК и конкретизировать разработанное приложение.

В результате выполнения данной работы должна быть представлена модель переиспользуемого фрагмента кода вместе с алгоритмами работы с ним, а также конкретная программа с формализованным интерфейсом взаимодействия, реализующая работу с фрагментами кода через данную модель и алгоритмы.

1 Обзор существующих средств хранения и стандартизации фрагментов кода

Цель данного раздела — выявить существующие решения в области хранения, стандартизации, переиспользования фрагментов кода, а также управления ими. Также необходимо определить, насколько эти решения соответствуют потребностям системы активных знаний и можно ли их использовать в контексте этой системы. Основные параметры оценки включают в себя:

- Возможность автоматизированного переиспользования фрагментов кода без участия человека;
- Наличие средств модульности, позволяющих обобщать хранение множества фрагментов кода;
- Возможность добавления множества метаданных к фрагментам кода;
- Расширяемость. Должна быть возможность добавления поддержки новых типов фрагментов кода без переработки ядра системы.

1.1 Обзор родственных работ

1.1.1 SPIRAL

SPIRAL представляет собой систему для автоматической генерации библиотек математических преобразований. Она использует различные алгебраические правила и генерирует множество вариантов реализации, а затем выбирает наиболее производительный код для целевой архитектуры. Рассматривая SPIRAL в раках проблемы стандартизации переиспользования фрагментов кода, можно отметить ряд ограничений, не позволяющих адаптировать эту систему под нужды системы активных знаний. У системы SPIRAL ограниченная предметная область — эта система работает только с численными алгоритмами, что ограничивает область применения (в то время как система активных знаний требует возможности применения в различных предметных областях, таких как физика, геофизика, биоинформатика и другие). Более того, в системе SPIRAL отсутствуют механизмы переиспользования существующих фрагментов кода — код генерируется с нуля, что не позволяет использовать эту систему для работы с фрагментом кода в общем случае. Таким образом, SPIRAL не подходит для внедрения в систему активных знаний. [8]

1.1.2 GitHub Copilot

GitHub Copilot представляет собой ИИ-инструмент на базе OpenAI [9], предлагающий фрагменты кода по контексту во время написания программ. Данное решение предлагает удобный инструмент для переиспользования уже существующих фрагментов кода и успешно

автоматизирует поиск кода с учётом контекста. Однако у этого решения отсутствует большое количество функций, необходимых для успешной интеграции в систему активных знаний. Основная проблема заключается в том, что подход через GitHub Copilot не является полностью автоматическим. Он требует участия программиста, который будет встраивать фрагменты кода в разрабатываемую программу. Однако одна из ключевых идей системы активных знаний — исключить программиста из процесса разработки программ, чтобы человек без навыков программирования мог описать нужную ему программу на понятном языке графов и получить её без вмешательства разработчика. Более того, решения, использующие за основу искусственный интеллект имеют ряд недостатков, связанных с расширением контекста. Система активных знаний предполагает под собой длительное накопление фрагментов кода, однако системы, основанные на искуственном интеллекте, начинают работать менее точно, при расширении контекста, что повышает возможность присутствия ошибки в итоговой программе. Автоматическое использование системы и правильность генерируемой программы являются ключевыми факторами при выборе подходящего решения, поэтому GitHub Copilot не подходит в качестве СУБФК. [10]

1.1.3 Protégé

Protégé представляет собой онтологический редактор. Эта система, так же как и система активных знаний, формализует предметную область с помощью графов. Однако система Protégé не предназначена для генерации программ и, как следствие, в ней отсутствует система хранения фрагментов кода. Более того, система Protégé представляет собой общий язык, можно сравнить его с русским языком — описав нефункциональные свойства фрагмента кода на русском языке, и приложив это описание к фрагменту кода мы добавим нефункциональные свойства к фрагменту кода, однако таким образом проблема автоматического переиспользования решена не будет. [11]

1.1.4 KNIME

KNIME представляет собой систему для анализа данных, основанную на концепции визуального программирования. Система активных знаний использует двудольные графы для описания вычислительной модели, в то время как KNIME использует последовательные «pipelines», которые не поддерживают циклы. Она также использует язык графов для непосредственного выполнения программ, что является одним из способов работы с системой активных знаний. В сравнении с системой активных знаний в KNIME не предусмотрено автоматическое выведение VW-плана, который описан в разделе 2, это значит, что ответственность за выведение алгоритма вычисления итогового результата из заданных переменных ложится на пользователя.

С точки зрения СУБФК — реализация хранения и переиспользования фрагментов кода в системе не поддерживает хранение и использование нефункциональных свойств вместе с фрагментами кода. Более того, в KNIME предполагается переиспользование фрагментов только определенных типов. В системе существует ряд граничений на формат фрагментов кода: в системе поддерживаются только Java и языки, которые запускаются через Java (Python, R). Остальные языки должны быть реализованы через внешние обходные пути, например REST-сервисы. А от системы активных знаний требуется поддержка фрагментов кода в более общем случае. Таким образом, реализация хранения и стандартизации переиспользования фрагментов кода, реализованная в KNIME, не подходит для внедрения в систему активных знаний. [12]

1.1.5 Galaxy

Galaxy представляет собой веб-платформу, предназначенную для обработки данных, в основном в сфере биоинформационных исследований. Данная система позволяет конструировать и выполнять сложные вычислительные процессы (workflows), используя визуальный интерфейс. Непосредственное написание программного кода не нужно. С точки зрения СУБФК — возможности работы с произвольными фрагментами кода в системе Galaxy ограничены — в систему заложена работа с заранее определенными типами фрагментов кода. Более того, в системе Galaxy отсутствуют механизмы хранения нефункциональных свойств, относящихся к определенным фрагментам кода.[13]

1.2 Выводы

Из обзора родственных работ, разобранных выше, можно сделать вывод, что существующих решений недостаточно для удовлетворения потребностей системы активных знаний. На данный момент нет систем, которые бы сочетали в себе автоматическое переиспользование фрагментов кода с учётом нефункциональных свойств конкретных фрагментов в общем случае. Сравнение приведено в таблице 1.1. Исследование этой темы — целесообразно.

Таблица 1.1 – Сравнение систем по потребностям системы активных знаний

Система	Авт. переиспользование	Метаданные	Общий случай
SPIRAL	-	-	-
GitHub Copilot	+- (полуавтоматическое)	-	+
Protégé	-	-	-
KNIME	+	-	+-
Galaxy	+	-	+-

2 Выявление требований для системы активных знаний. Разработка теоретического решения

2.1 Система активных знаний

Прежде чем переходить к описанию свойств, которыми должна обладать СУБФК, приведем базовый принцип работы системы активных знаний. Ключевая идея, которая позволяет автоматически создавать параллельные (и не только) программы, состоит в выводе решения задачи, поставленной на вычислительной модели, и в последующей подстановке соответствующих модулей.

Вычислительная модель представляет собой ориентированный двудольный конечный граф, состоящий из двух типов вершин (двух долей): операционных узлов и переменных узлов. Дуги, входящие в операционный узел, задают входные переменные, а исходящие из него — выходные переменные. Эта модель формализует предметную область, где характеристики объектов выражаются через значения переменных, а зависимости между свойствами определяются операционными узлами, позволяющими вычислять одни параметры на основе других. Таким образом, структура графа отражает вычислительные связи между данными и преобразованиями в рамках рассматриваемой системы и однозначно задает, из каких переменных можно вывести другие, и какие другие переменные для этого понадобятся. Ниже приведен пример вычислительной модели, см. рисунок 2.1.

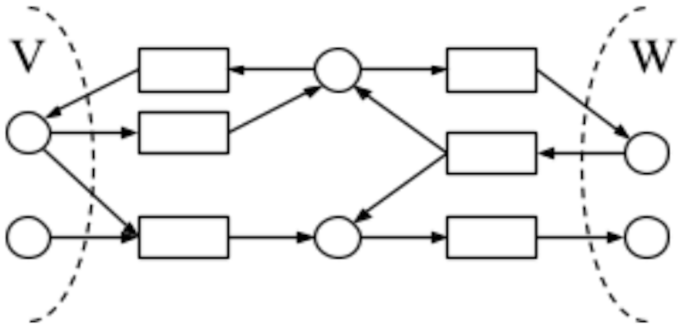

Рисунок 2.1 – Пример вычислительной модели. Круги — переменные, прямоугольники — операции

Одним из примеров вычислительной модели является тригонометрия. Возьмем предметную область ’треугольник’, тогда переменные будут описывать различные свойства треугольника (например длину сторон, углы, радиус описанной окружности и так далее), а операции будут описывать правила, по которым одни переменные выводятся из других (например, если известно 2 угла, то можно вычислить третий, отняв их от 180), см рисунок 2.2.

Для обеспечения вычисления выходных параметров на основе входных данных каждой операции ставится в соответствие определенный вычислительный блок (фрагмент кода). Под

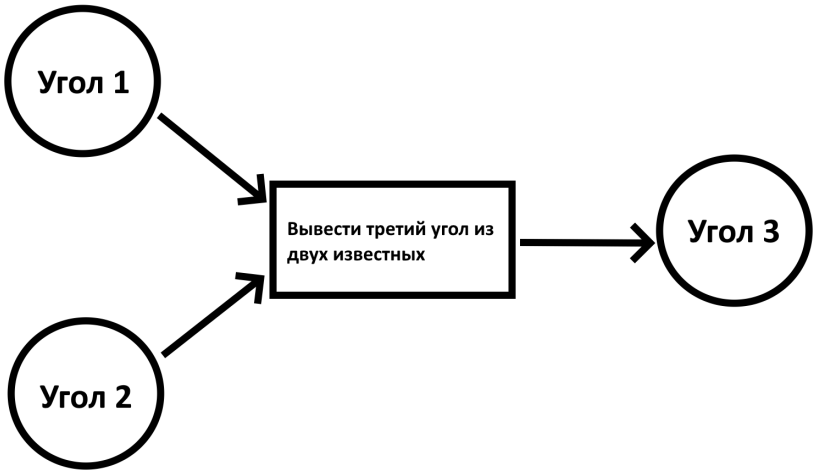

Рисунок 2.2 – Пример простейшей вычислительной модели с одной операцией

выполнением операции понимается применение этого блока к исходным данным с последующим получением результирующих значений.

В рамках вычислительной модели выделяются два специальных подмножества переменных: V - исходные параметры задачи, W - целевые параметры задачи. При фиксации значений всех переменных из V и W формируется так называемая VW-задача. Решение такой задачи возможно при наличии в модели набора операций, упорядоченное выполнение которых позволяет, начиная с заданных значений V, упорядоченно вычислять все требуемые значения из W. Такой упорядоченный набор операций именуется VW-планом. Важно отметить, что для конкретной VW-задачи может существовать несколько VW-планов или не существовать ни одного, и при успешном нахождении такого плана появляется возможность вычислить все целевые параметры W на основе известных исходных данных V (В случае примера с треугольником V может быть двумя углами, а W — третьим. Тогда если брать вычислительную модель из рисунка 2.2, то VW-план будет выглядеть также как граф на этом рисунке).

Каждому вычислительному модулю соответствует конкретное отображение, устанавливающее взаимосвязь между входными и выходными параметрами данной операции. Важно отметить, что подобная функциональная зависимость также распространяется и на те переменные, которые не связаны операционными преобразованиями напрямую. Выделение множеств V (входных параметров) и W (выходных параметров) представляет собой формальный механизм задания функциональных требований к решаемой задаче. Однако, для практического применения описанного подхода нужно также учитывать нефункциональные свойства конкретных подставляемых операций (для одного модуля, могут существовать различные операции с разными функциональными свойствами. К примеру, одна операция может быть быстрее, но тратить больше памяти во время выполнения, в то время как другая — наоборот) [1].

Пользователи, в свою очередь, делится на два типа: инженеры знаний и рядовые пользователи. На инженеров знаний ложится ответственность за добавление новых фрагментов кода в систему, а также за построение графов, описываюших предметные области. Рядовые пользователи, в свою очередь, не добавляют новые фрагменты кода в систему, а используют уже добавленные. Таким образом, работа рядового пользователя сводится к менее сложной и тудоемкой работе — построению высокоуровневой спецификации. [14]

2.2 Роль и задачи СУБФК

В описанном выше контексте СУБФК — это компонент, из которого достаются конкретные операции и нефункциональные свойства, подставляемые в модули перед вычислениями. В примере с тригонометрией СУБФК содержит в себе все возможные операции и их нефункциональные свойства для известных модулей.

На данном этапе можно сформулировать модель проблемы и требуемое решение, вместе с формальными требованиями к нему. Несмотря на наличие обширного арсенала существующих программных решений (библиотек, фрагментов кода), их повторное использование сопряжено с рядом фундаментальных трудностей, которые результируют в итоговом процессе, требующем значительных временных и профессиональных ресурсов:

- Необходимость ручного анализа и адаптации существующего кода
- Отсутствие стандартизированных механизмов интеграции компонентов
- Высокий порог входа для специалистов предметной области, не обладающих квалификациями в программировании

Для решения этих проблем требуется разработать систему управления базой фрагментов кода (СУБФК) как составную часть системы активных знаний, которая должна обеспечить часть функциональности системы активных знаний связанной с хранением добавленных в систему фрагментов кода и стандартизацией фрагментов кода для будущего эффективного переиспользования.

Требования:

- Должно быть конкретизированно представление фрагментов кода. Фрагмент кода, хранящийся в системе должен содержать информацию достаточную для повторного автоматизированого переиспользования в системе активных знаний
- Фрагмент кода должен быть способен содержать в себе неограниченное (в разумных пределах) количество нефункциональных свойств, которые могут менятся от предметной области к предметной области
- Должны быть возможны создание, чтение и удаление фрагментов кода
- После добавления фрагмента кода в систему он должен использоваться в системе автоматически, без вмешательства человека

- Остальные компоненты системы должны использовать фрагменты кода через СУБФК стандартизованно, без подстроек под конкретные фрагменты кода (независимость от внутренней реализации)
- Должна быть возможность постоянного добавления новых типов фрагментов кода (например, при появлении нового языка программирования). То есть, система должна быть расширяемая.
- Система должна работать с фрагментами стандартизованно. Она не должна зависеть от внутренней реализации конкретного фрагмента кода.

Критерии проверки:

- Успешная интеграция СУБФК в систему активных знаний (разработанные компоненты работают через СУБФК)
- Система активных знаний имеет возможность сгенерировать работающую программу, используя фрагменты кода из СУБФК
- СУБФК имеет возможность работать с практическими задачами

Стандартизованное переиспользование любых фрагментов кода в общем случае — едва ли возможно. У различных фрагментов кода есть различные особенности, которые нужно учитывать при включении их в программу (вплоть до языка на котором написан конкретный фрагмент кода).

2.3 Предлагаемое решение

Предлагается решить эту проблему системой плагинов. Общая идея такая: так как в общем случае переиспользовать фрагменты кода возможности нет — уменьшить множество фрагментов кода до такого размера, когда стандартизированное переиспользование — возможно (например, множество фрагментов кода написанных на языке C и представляющих собой одну процедуру стандартного вида). Для этого суженного множества можно написать ”плагин”— встраиваемый в СУБФК код, который добавляет возможность СУБФК работать с этим множеством фрагментов кода стандартизованно. По мере роста потребностей системы, и по мере появления необходимости в других множествах фрагментов кода, добавлять новые плагины в систему. Таким образом, для каждого отдельного множества фрагментов кода можно по мере расширения системы активных знаний добавить свой плагин, позволяющий СУБФК стандартизированно работать с конкретным типом фрагментов кода. А один фрагмент кода, может иметь возможность работы с несколькими плагинами. (В фрагменте кода может быть две реализации одной и той же программы на разных языках. Один плагин будет работать с одиной реализаций, второй, в свою очередь, со второй) [15, 16] При добавлении уже существующих фрагментов кода в систему, эти фрагменты кода подлежат стандартизации — это будет описано ниже.

Плагин является отдельной от СУБФК сущностью — его может разработать человек, не знающий, как внутри устроена СУБФК, но знающий формат плагина. Это дает возможность системе активных знаний постоянно расширять свои возможности и быть независимой от конкретного разработчика, создавшего систему. Таким образом решается проблема невозможности переиспользования фрагментов кода в большинстве случаев.

Для того чтобы автоматически переиспользовать фрагмент кода, он должен соответствовать определенному соглашению, в предложенном решении это достигается путем приложения к фрагменту кода некоторых метаданных в заранее определенном формате. Эти метаданные могут менятся в зависимости от плагина и множества фрагментов кода, например, для переиспользования процедуры написанной на языке C нужно знать сигнатуру этой процедуры, а для переиспользования фрагмента кода, который является интернет сервисом (например, если мы запрашиваем погоду через интернет), нужно знать способ построения ссылки, ключи, и так далее. Поэтому к каждому плагину прикладывается список метаданных которые нужно приложить к фрагменту кода при его добавлении в систему. Эти метаданные — отвечают за стандартизацию фрагментов кода перед добавлением в систему. Они делятся на обязательные и необязательные. Обязательные метаданные определены для каждого плагина и могут различаться, они нужны для правильного переиспользования фрагмента кода (например, сигнатура процедуры, написанной на языке C), необязательные метаданные, в свою очередь, зависят не от плагина, а от предметной области и могут использоваться для выводов более эффективных VW-планов (Например сложность алгоритмов выполняемых в этом фрагменте кода).

Поэтому, при добавлении фрагмента кода в систему нужно выбрать плагин(ы) с которым будет работать этот фрагмент кода, а затем нужно добавить к фрагменту кода все необходимые обязательные метаданные для каждого плагина с которым будет работать этот фрагмент кода. После чего СУБФК, используя знания о том, через какие плагины с этим фрагментом кода можно работать, и все дополнительные необходимые метаданные приложенные к фрагменту кода, может стандартизованно переиспользовать этот фрагмент кода, см рисунок 2.3.

Предлагаемое решение учитывает вышеописанные требования:

- Практическая часть описана ниже, в разделе с описанием реализации, однако на данном этапе работи с фрагментами кода организуется через соглашение для этих фрагментов кода путем определения метаданных для каждого типа фрагментов, а также механизмами валидации добавляемых фрагментов кода, которые могут быть реализованы в СУБФК. Информации из приложенных метаданных достаточно для повторного автоматизированного переиспользования, так как на этапе программирования плагина все необходимые метаданные конкретизируются.
- В предлагаемом соглашении для фрагментов кода оставлено место для добавления любых дополнительных необязательных метаданных, количество которых ограничено только с

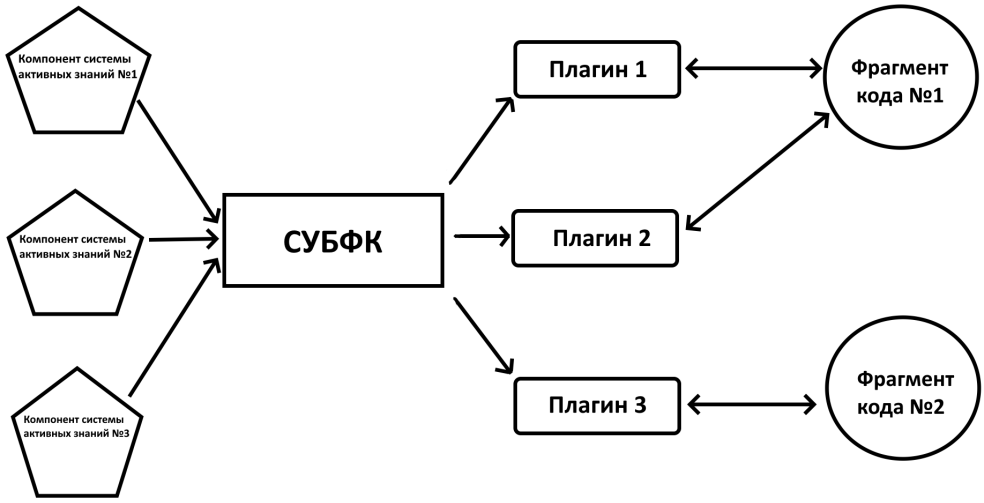

Рисунок 2.3 – Схема работы СУБФК

технической части. Также из-за стандартизованного хранения метаданных у фрагментов кода в СУБФК могут быть реализованы механизмы валидации этих метаданных.

- Учитывая вышеописанный дизайн СУБФК операции по добавлению, удалению и чтению фрагментов кода могут быть реализованы (также могут быть реализованы операции по обновлению уде добавленных в СУБФК фрагментов кода, однако это напрямую не влияет на итоговую работоспособность системы). Более того, из-за разделения плагинов и фрагментов кода на две разные сущности, мы снимаем с пользователя, который будет добавлять фрагменты кода в систему, требования к знаниям программирования. Плагины разрабатываются отдельно и ему не обязательно знать, как внутри устроен плагин, или как с плагином взаимодействует СУБФК.
- Предложенный вариант СУБФК предполагает под собой автоматическое переиспользование фрагментов кода без вмешательства человека. После добавления фрагмента кода в систему, инженеру знаний не нужно взаимодействовать с ним напрямую (рядовой пользователь работает с уже добавеленными в систему фрагментами кода)
- СУБФК напрямую не работает с фрагментами кода, но работает с сущностью ’плагин’, поэтому информации про внутреннюю реализацию фрагмента кода у неё — нет.
- Расширяемость системы напрямую заложена в предложенный вариант СУБФК.
- Так как взаимодействие с фрагментами кода осуществляется через систему плагинов, у СУБФК нет информации про внутреннюю реализацию конкретных фрагментов кода.

Также можно выделить ряд приемуществ данной архитектуры:

- Возможна постепенная эволюция СУБФК через добавление новых плагинов. Это делает систему расширяемой.

- Полная независимость разработки плагинов от информации о внутреннем устройстве СУБФК. Учитывая предложенный дизайн системы, для разработки нового плагина достаточно знать только требования к самому плагину, предъявляемые от СУБФК.
- Сохранение обратной совместимости с ранее добавленными в систему фрагментами кода. Если случается ситуация, когда нужно добавить новый фрагмент кода который отличается от предыдущих и требует перестроения уже существующего плагина — это делать не обязательно. Достаточно добавить новый плагин в СУБФК, который соотвтетствовал бы новым потребностям этого фрагмента кода.

Однако, у предложенного дизайна также существует ряд ограничений:

- Необходимость разработки плагинов. Как рядовой пользователь, как использующий саму систему активных знаний, так и пользователь, добавляющий фрагменты кода в систему, не имеющий навыков программирования, не имеет возможности добавить новый тип фрагментов кода в систему, так как это требует продвинутых навыков программирования.
- Ответственность за заполнение метаданных ложится на пользователя. В то время как СУБФК имеет возможность проверить наличие обязательных метаданных, в общем случае возможности гарантировать правильность этих данных — нет. Это следует из того, что у СУБФК нет информации о внутренней структуре фрагмента кода. Этот факт позволяет стандартизированно переиспользовать добавленные фрагменты кода, но ограничивает систему в возможностях валидации приложенных метаданных. Если данные введены некорректно (по значениям — допустим неверно указана сигнатура процедуры), то это может выясниться только после создания итоговой сгенерированной программы, а может и не выясниться вовсе.

2.3.1 Модель фрагмента кода

Из этого вытекает использованная в этой работе модель фрагмента кода. Она состоит из двух частей — корневых файлов программы и метаданных, сохраненных придерживаясь известного соглашения для подобных метаданных. Сам формат корневых файлов программы и метаданных не играет решающей роли, однако у корневых файлов программы должна быть возможность быть переиспользованными компетентным программистом, в свою очередь у СУБФК должна быть возможность однозначно определить и прочитать метаданные приложенные к корневым файлам программы, а в самих метаданных должно быть достаточно информации для ответов на вопросы о количестве и типах входных и выходных аргументов программы, а также о количестве и идентификаторах плагинов, с которыми может взаимодействовать конкретный фрагмент кода.

Предложенное решение демонстрирует достаточную степень соответствия требованиям, а также представляет собой основу, с помошью которой можно расширять систему.

3 Описание реализации и тестирование

Для реализации решений, предложенных в прошлом разделе, были разработаны проектные решения по реализации. Описание разработанных решений представлено ниже. Сначала приводится изложение архитектуры и конкретных реализаций, обоснование представленных решений приводится после.

Описанное решение реализовано в виде микросервиса на языке Java с помощью фреймворка ”Spring Framework”. Фреймворк ”Spring Framework”используется только для реализации микросервисной части системы — работа с плагинами и поиск фрагментов кода в хранилище реализован вручную. Система включает в себя три части — хранилище фрагментов кода, хранилище плагинов и систему управления.

3.1 Хранилище фрагментов кода

Хранилище фрагментов кода представляет собой определенную папку в системе, где была запущена система управления. В случае отсутствии нужной папки — она инициализируется сама. В случае наличия этой папки считается, что хранилище уже установлено. Местонахождением папки по умолчанию является ’C:/codeFragmentControlSystem/code_fragments’. В случае запуска на Windows — на диске C в папке codeFragmentControlSystem, в случае Linux - в той же папке где запускается система управления создается папка с именем ”C: в нее помещается codeFragmentControlSystem.

В хранилище фрагментов кода содержатся пакеты фрагментов кода добавленные в систему. Пакет фрагментов кода — единица добавления фрагментов кода в систему. В реализованном прототипе пакет может содержать в себе большое количество фрагментов кода, фрагмент кода, в свою очередь, не может существовать без пакета. Пакет представляет собой папку, содержащую в себе файл ’description.xml’ и папку ’src’. В папке ’src’ хранятся все файлы внутренней реализации фрагмента кода (в общем случае они могут быть не связаны друг с другом, например, если внутри фрагмента кода присутствуют реализации на разных языках). У системы нет информации о том, что находится внутри этой папки, и у неё не должно быть этого знания. Файл ’description.xml’ является ’мостом’ от фрагмента кода к плагину. В нём указывается вся метаинформация нужная выбранным плагинам для работы с конкретным фрагментом кода, а также вся дополнительная метаинформация которая может использоваться для оптимизации создания VW-планов остальной системой. Абстракция ”пакет”создана для уменьшения дублирования кода между фрагментами кода. Если два фрагмента кода находятся в одной предметной области или решают похожие задачи, существует значительная вероятность повторения в этих фрагментах кода других подпрограмм. Добавление абстракции ”пакет добавляет возможность сохранять такие подпрограммы в СУБФК один раз для нескольких

фрагментов кода, вместо того, чтобы для каждого фрагмента кода сохранять свою копию каждой используемой подпрограммы. Более того, такая абстрация добавляет в систему удобный способ передачи фрагментов кода между людьми — фрагмент кода является недостаточной по размеру единицей передачи. Пакеты могут объединять в себе множество фрагментов кода.

В общем случае процесс добавления фрагментов кода в систему выглядит так:

1. Выбрать фрагмент кода, который мы хотим добавить в систему
2. Добавить все нужные файлы внутренней реализации в в папку 'src'
3. Заполнить 'description.xml' используя установленный формат
4. Архивировать получившуюся папку и отправить её в систему управления по API (Эти шаги поменяются при появлении у системы активных знаний пользовательского интерфейса. Когда это случится — создание фрагмента кода будет производтся через интерфейс)

Ниже приводится листинг формата description.xml файла, см листинг 3.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<code_fragments>
 <cf>
  <name>Имя фрагмента кода, которое будет отображаться пользователю</name>
  <description>Описание фрагмента кода на естественном языке, которое поможет пользователю понять что делает этот фрагмент кода при выборе</description>
  <input_var_count>Количество едииниц данных требуемых на вход. В пользовательском интерфейсе будет отображаться столько входов, какое число здесь указано</input_var_count>
  <ivars>
   <ivar>Здесь находится текстовое описание входных значений. Эти описания будут отображаться пользователю когда он будет строить итоговый граф. С помощью этих описаний пользователь сможет понять в какой вход должны идти конкретные переменные</ivar>
   <ivar>Описание второй входной переменной</ivar>
  </ivars>
  <output_var_count>Количество единиц данных передаваемых на выход</output_var_count>
  <ovars>
   <ovar>Здесь находится текстовое описание выходных значений.</ovar>
   <ovar>Опичание второй выходной переменной</ovar>
  </ovars>
  <ivar_types>
   <ivar_type>Здесь определяются типы входных переменных. Список доступных входных и выходных переменных доступен в документации к СУБФК допустим(может быть int - число или local_path - локальный путь к выходному значению)</ivar_type>
   <ivar_type>Тип второго входного значения</ivar_type>
  </ivar_types>
```

```
    <ovar_types>
     <ovar_type>Здесь определяются типы выходных переменных</ovar_type>
     <ovar_type>Тип второго выходного значения</ovar_type>
    </ovar_types>
    <local_paths>
     <local_path>Если в выходных значениях присутствует тип 'local_path' или '
    global_path', тогда в этой секции указываются пути до выходных значений</
    local_path>
     <local_path>Локальный путь до второй выходной переменной</local_path>
    </local_paths>
    <implemented_plugins>
     <plugin>Здесь указывается плагин, с которым будет работать этот фрагмент
    кода</plugin>
    </implemented_plugins>
    <plugin_specific> Здесь// указываются метаданные которые относятся к
    конкретным плагинам
     <plugin>
      <name>Название плагина. Все плагины перечислены в документации к СУБФК</
    name>
       <первое свойство которое требует плагин например( exe_path)>corr.out<
    первое/ свойство которое требует плагин например( exe_path)>
     </plugin>
    </plugin_specific>
    <id>Здесь хранится уникальный идентификатор фрагмента кода в системе. При
    добавлении фрагмента кода в систему этого поля быть не должно, оно
    добавляется само</id>
   </cf>
  </code_fragments>
```

Листинг 3.1 – Формат description.xml файла [17]

3.2 Хранилище плагинов

Хранилище плагинов представляет собой определенную папку в системе, где была запущена система управления. В случае отсутствии нужной папки — она инициализируется сама. В случае наличия этой папки считается что хранилище уже установлено. Местонахождением папки по умолчанию является 'C:/codeFragmentControlSystem/plugins'. В случае запуска на Windows - на диске C в папке codeFragmentControlSystem, в случае Linux — в той же папке, где запускается система управления, создается папка с именем "C: в нее помещается codeFragmentControlSystem.

Внутри папки хранилища плагинов хранятся плагины СУБФК. Они хранятся в виде папок. Папка с плагном имеет название этого плагина. Плагин — программа-дополнение для

системы управления. Эта программа описывает как система управления должна обрабатывать определенный тип фрагментов кода. Плагин является java классом реализующим интерфейс Plugin, см. листинг 3.2. Имя класса должно строго совпадать с названием плагина, известным системе управления.

```
public interface Plugin {
  public List<String> getResult(String cfId, String urlToControlSystem);
}
```

Листинг 3.2 – Интерфейс Plugin. id — идентфиикатор фрагмента кода в библиотеке. urlToControlSystem — url для доступа к API системы управления

При разработке нового плагина создается новый класс реализующий этот интерфейс, метод getResult должен, используя идентификатор фрагмента кода и ссылку до API системы управления, выполнить то, что ожидается от этого плагина. Ниже приведены шаги разработки и добавления нового плагина в систему.

1. Выбрать класс программ который будет покрывать плагин. Допустим, реализуем плагин который будет работать с фрагментами кода написанными на языке C. Пусть эти фрагменты кода содержат в себе файл с расширением ’.c’ в котором содержится программа на языке C
2. Выбрать, что ождается от этого плагина и выбрать название. Допустим, мы ожидаем, что этот плагин будет возвращать код из этого файла на языке C. Название - getCProcedure
3. Продумать, какая метаинформация нужна этому плагину чтобы сделать то, что от него ожидается. В этом примере нам понадобится путь до файла
4. Зафиксировать в документацию по плагину информацию о необходимой метаинформации для этого плагина. Пусть свойство будет называться path, тогда ’description.xml’ у фрагментов кода, которые будут работать с этим плагином, должен выглядеть следующим образом, см. листинг 3.3.

```
...
<plugin_specific>
...
 <plugin>
  <name>getCProcedure</name>
  <path>exampleFolder/exampleCProcedureFile.c</path>
 </plugin>
...
</plugin_specific>
...

```

Листинг 3.3 – Формат description.xml файла

5. Разработать плагин. С помощью API системы управления мы получаем всю необходимую метаинформацию, в частности путь до файла, а также путь до самого фрагмента кода. Реализуемый метод в классе должен с помощью метаинформации полученной по API получить доступ к необходимоому файлу, считать его, и вывести итоговую программу как выходной List<String> аргумент. Например первый String — список include выражений, а второй — вся остальная программа
6. На этом этапе плагин разработан. Он, вместе с документацией передается разработчику, поддерживающему СУБФК. Если плагин приемлемый, то поддержка этого плагина добавляется в СУБФК, а информация о плагине добавляется в общий список доступных плагинов. Затем инженеры знаний, добавляющие новые фрагменты кода в систему, могут добавлять фрагменты кода, которые работают через этот плагин.

3.3 Система управления

Система управления представляет собой Java программу, использующую Spring Framework для реализации своих микросервисных функций. Система управления отвечает за удаление и добавление фрагментов кода, предоставление конкретной информации о фрагментах кода и плагинах, а также за работу с фрагментами кода через плагины. Системе управления через определенное API передаются запросы от системы активных знаний. Для передачи сообщений используется протокол HTTP.

Ниже приводится список запросов к системе управления (возвращаемые значения):

1. code_fragments
   - Описание: Возвращает список [id] фрагментов кода
   - Формат возвращаемых значений: json
   - Пример возвращаемого значения, см. листинг 3.4:

```
{
  "code_fragments":["b642e476-8d79-4783-b1da-a016b0d226aa"]
}

```

Листинг 3.4 – Пример возвращаемого значения запроса code_fragments

   - Пример запроса: GET /code_fragments
2. info
   - Описание: Возвращает всю доступную метаинформацию про фрагмент кода
   - Формат возвращаемых значений: json
   - Пример возвращаемого значения, см. листинг 3.5:

```
    {
     "package": "Convolution",
     "inputVarCount": 3,
     "outputVarCount": 2,
     "description": "Correlation convolution of seismic traces. The report for the resulting correlograms includes the signal-to-noise ratio (SNR) and equivalent signal amplitude (A) in ADC units.",
     "name": "Correlation Convolution of Seismic Traces",
     "inputVarTypes": ["any_path", "any_path", "int"],
     "inputVarDescriptions": [Text file with a list of seismic traces in PC-A format",Reference signal file in PC-A format",Correlation interval in seconds"],
     "outputVarTypes": ["local_path", "local_path"],
     "outputVarDescriptions": ["Work report in .log format","Final convolved seismic trace file"],
     "implementedPlugins": ["ExeStartCProcedure"]
    }

```

Листинг 3.5 – Пример возвращаемого значения запроса info

- Дополнительные параметры: ?specific=name (вместо name поставить любой ключ из возврщаемого значения при запросе без specific, также можно комбинировать запросы specific: /info?specific=inputVarCount@outputVarCount@description@implementedPlugins)
- Пример запроса: GET /"id"/info

3. plugins
    - Описание: Возвращает количество и список поддерживаемых плагинов фрагмента кода или список всех доступных плагинов вообще
    - Формат возвращаемых значений: json
    - Пример возвращаемого значения, см. листинг 3.6:

```
    {
      "plugins":["ExeStartCProcedure", "getSO"]
    }

```

Листинг 3.6 – Пример возвращаемого значения запроса plugins

- Пример запроса: GET /"id"/plugins или /plugins

4. getNeededContentFor

- Описание: Возвращает все нужные корневые файлы фрагмента кода в .tar архиве в зависимости от плагина
- Формат возвращаемых значений: .tar архив. Скачивается архив с именем ”id фрагмента кода”.tar в котором лежит папка src со всеми нужными корневыми файлами фрагмента кода
- Дополнительные параметры: ?plugin=”plugin_name”
- Пример запроса: GET /”id”/getNeededContentFor?plugin=ExeStartCProcedure

5. pluginProcedure
    - Описание: Возвращает результат работы выбранного плагина для выбранного фрагмента кода
    - Формат возвращаемых значений: Зависит от плагина
    - Пример запроса: GET /”id”/pluginProcedure?plugin=ExeStartCProcedure
6. add_package
    - Описание: Добавляет пакет в систему.
    - Формат входныых значений: packageName, MultipartFile (xml, .tar)
    - Пример возвращаемого значения:
    - Дополнительная информация: Post запрос. В xml должен быть description файл. В файле должен быть .tar файл в котором лежит папка src с корневыми файлами фрагмента кода
    - Пример запроса через curl: curl -X POST .../add_package -H ”Content-Type: multipart/form-dataF ”packageName=packageNameF ”xml=@description.xmlF ”file=@test_send.tar”
7. remove_package
    - Описание: Удаляет пакет из системы
    - Формат входных значений: параметр URL
    - Пример запроса: DELETE /remove_package?packageName=Convolution

Возможные коды ошибок (для всех запросов) включют в себя:

1. 500
    - Ошибка при получении доступа к файлам плагина
    - Ошибка при получении доступа к description файлу
    - Неверный формат description файла (internal file corruption или неправильно залили пакет)

- Не удалось получить доступ к папке с плагинами
- Не удалось создать .tar файл из корневых файлов
- Ошибка при получении списка фрагментов кода
- Ошибка при получении списка плагинов
- Не получилось записать значения в description файл при добавлении фрагмента кода
- Ошибка при получении доступа к месту хранения фрагментов кода
- Ошибка при создании папки для хранения фрагмента кода
- Ошибка при распаковке исходных файлов фрагмента кода
- Неправильный формат плагина
- Неизвестная ошибка. Смотерть само сообщение

2. 400
    - Запрошен неизвестный плагин
    - Запрошен неизвестный запрос
    - Запрошен неизвестный метод у фрагмента кода
    - Запрошен неизвестный фрагмент кода
    - В description файле нет запрошенного поля
3. 409
    - Предосталвенный фрагмент кода уже есть в базе. Измените имя или удалите старый фрагмент кода.
4. 200
    - Успех

Запрос на исполнение процедуры определенного плагина использует динамическую загрузку классов в java (рефлексию) для включения плагина в программу и исполнения его процедур без знаний о внутренней реализации плагина.

3.4 Обоснование и особенности реализации

От СУБФК не требуется быстродействия и эффективности по памяти, однако, для неё очень важная расширяемость. По этой причине был выбран язык Java.

Хранение фрагментов кода и плагинов реализовано в папках в системе для простоты будущего расширения и улучшения системы. СУБФК может ещё много раз переписываться изменятся и улучшатся, также могут быть созданны другие СУБФК, реализованные другими

инструментами. Использование папкок позволяет хранить фрагменты кода отчужденно, с возможностью несложного переноса хранилища в другую систему.

Микросервисная архитектура, в свою очередь, обеспечивает назависимость разработки компонентов системе активных знаний — на момент проведения испытаний версия системы активных знаний, использующая в своем составе СУБФК, ещё не была готова для использования потребителя, однако, из-за того что компоненты системы активных знаний предсталвляют собой микросервисы, у разработчиков есть возможность проводить интеграционное тестирование с выбранным числом компонентов. Более того, микросервисная архитектура добавляет возможность для почти неограниченного роста за счет распределенности. На данный момент реализация хранилища фрагментов кода реализована как папка в системе на которой запущена система управления, однако, расширение системы и перевод хранилища фрагментов кода и хранилища плагинов в удаленный формат не остановит работы системы, так как для остальных компонентов ничего не поменяется. Более того, сама система управления может быть установленна на удаленном сервере — таким образом мы снимаем необходимость наличия JVM на машине пользователя. Система активных знаний предполагает возможность активного использования большого количества фрагментов кода — количество используемых фрагментов кода не поместится на одной машине у рядового пользователя.

По этой причине сохранение новых фрагментов кода в систему реализовано через отправку .tar архива по сети. Это дольше и более ресурсозатратно, однако таким образом мы увеличиваем потенциал для расширения.

3.5 Испытания

Для оценки результата было выбрано интеграционное тестирование как основной показатель успешности выполненной работы — так как СУБФК работает как сервер (не посылает запросы другим компонентам, а только отвечает на них), значит если остальные компоненты системы активных знаний работают без ошибок, используя при этом в своей работе СУБФК, СУБФК можно считать удовлетворяющей потребностям системы. Помимо этого, нужно проверить основую функциональность системы:

- Добавление и удаление фрагментов кода
- Получение метаинформации о фрагменте кода
- Возможность добавления плагина, разработанного по алгоритму описанному в секции 3.2, и расширения доступного для добавления круга фрагментов кода.

Для нефункциональных требований достаточно субъективной оценки СУБФК на их выполнение. В частности нужно оценить способность системы к расширению, а также назависимость её работы от внутреннего наполнения фрагментов кода.

На момент проведения испытаний версия системы активных знаний, использующая в себе СУБФК ещё не готова к использованию пользователем, поэтому большая часть испытаний представляет собой симуляцию поведения во время работы системы. Несмотря на это — основной оценкой результата остается проверка на реальной системе, поэтому в первую очередь испытания включают в себя взаимодействие с другими компонентами системы. Начиная с зимы 2025 года СУБФК была запущена на одном из тестовых стендов кафедры паралелльных вычислений и использовалась вплоть до момента написания этого текста. После небольшого периода стыковки с другими компонентами система начала работать стабильно и требовала перезагрузки только после выключения самого тестового стенда. Также, опираясь на логирование производимое СУБФК в этом промежутке времени, можно сделать вывод о том, что количество оишбок со временем уменьшалось (во время тестирования исправлялись баги), а с марта 2025 года ошибки перестали появляться вообще.

Тем не менее, как описывалось ранее, некоторые компоненты системы ещё не внедрены на тестовый стенд. Это значит, что во время интеграции с другими, уже готовыми, компонентами, некоторые тестовые сценарии могли быть не пройдены. Поэтому также были проведены тесты на базовую функциональность системы. В частности была проведена проверка API системы на соответствие зафиксированному режму работы (были проверены все описанные запросы, форматы этмх запросов, а также соответствие возвращаемых данных ожидаемым), также был разработан один плагин, который был успешно внедрен в систему по алгоритму, описанному в секции 3.2, и была проверена правильность его работы с точки зрения других компонентов. Основной компонент, работающий с СУБФК, — это генератор. Этот компонент отвечает за генерацию итоговой паралельной программы которая отдается пользователю. На момент проведения испытаний этот компонент не был готов, поэтому испытаение заключалось в оценке возможности автоматически сгенерировать программу, которая бы выполяла задачу вычисления корреляционной свертки сейсмотрасс, используя СУБФК как источник кода. Испытание состоит из следующих шагов:

1. Использовать запрос code_fragments для того чтобы получить идентификатор нужного фрагмента кода
2. Использовать запрос pluginProcedure через plugin ExeStartCProcedure и getNeededContentFor для получения итоговой процедуры запуска и требуемых для запуска корневых файлов фрагмента кода
3. Из возвращаемого значения pluginProcedure составить итоговую процедуру C
4. Вставить эту процедуру, не меняя её, внутрь любой C программы
5. Вызвать эту процедуру в программе так, чтобы она была запущена в одной папке с разархивированным результатом getNeededContentFor
6. Запустить программу

7. Должно запуститься вычисление корреляционной свертки сейсмотрасс
8. В самой программе должна быть возможность получить доступ к результату выполнения корреляционной свертки

Испытание прошло успешно — метаинформации и фрагмента кода, которые были получены по API, хватает для того, чтобы автоматизированно сгенерировать программу, не имея информации о внутреннем устройстве фрагмента кода — задача вычисления корреляционной свертки сейсмотрасс успешно запускается, и внутри программы можно получить результат выполнения этой задачи.

В контексте данной работы такие результаты можно считать успешными. СУБФК, периодически взаимодействуя с другими компонентами, работает стабильно с течением времени. Различным компонентам системы активных знаний достаточно функциональности СУБФК. А также, используя СУБФК, теоретически можно автоматизированно генерировать программы без учета внутренней реализации конкретных фрагментов кода.

ЗАКЛЮЧЕНИЕ

В ходе представленной работы была спроектирована и разработана система управления базой фрагментов кода, также были разработаны необходимые для создания этой системы модели и форматы. Было проведено тестирование разработанной системы.

Практические испытания подтвердили, что предложенное решение представляет собой работающий прототип СУБФК, выполняющий требуемые системой активных знаний от СУБФК функции. С теоретической стороны была создана основа, на которой может продолжаться работа над данной задачей со стороны стандартизации и хранения фрагментов кода. Поставленные перед этой работой задачи были успешно выполнены, в частности:

- Были конкретизированы потребности системы активных знаний, а также специфицированы требования предъявляемые к СУБФК.
- Были проанализированы существующие решения, доступные для решения проблемы хранения и стандартизации переиспользования существующих фрагментов кода.
- Были разработаны теоретические основы и технические решения для создания СУБФК.

Однако разработанное решение имеет множество областей, которые нуждаются в улучшении. По большей части эти улучшения касаются технической части. В дальнейшие планы работы входит:

- Движение СУБФК в сторону распределенного хранения фрагментов кода.
- Расширение возможностей СУБФК путём добавления новых плагинов.
- Добавление поддержки системы пользователей для централизованного хранения фрагментов кода.
- Добавление в СУБФК функциональности валидации метаданных.

Итоговый разработанный прототип может быть использован в составе системы активных знаний для выполнения функции хранения и стандартизации переиспользования фрагментов кода. Теоретический результат, в свою очередь, может быть использован как основа для разработки собственного решения для СУБФК.

На защиту выносятся следующие положения:

- Разработана модель фрагмента кода.
- Разработан интерфейс взаимодействия с фрагментами кода.
- Разработана система управления базой фрагментов кода.
- Проведено тестирование разработанной системы управления базой фрагментов кода.

Выпускная квалификационная работа выполнена мной самостоятельно и с соблюдением правил профессиональной этики. Все использованные в работе материалы и заимствованные принципиальные положения (концепции) из опубликованной научной литературы и других

источников имеют ссылки на них. Я несу ответственность за приведенные данные и сделанные выводы.

Я ознакомлен с программой государственной итоговой аттестации, согласно которой обнаружение плагиата, фальсификации данных и ложного цитирования является основанием для не допуска к защите выпускной квалификационной работы и выставления оценки «неудовлетворительно».

_____ _____

ФИО студента Подпись студента

«____»_____20__г.

(заполняется от руки)

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. *Malyshkin V., Perepelkin V. A.* Построение баз активных знаний для автоматического конструирования решений прикладных задач на основе системы LuNA // Параллельные вычислительные технологии – XVIII Всероссийская научная конференция с международным участием, ПаВТ'2024, г. Челябинск, 2–4 апреля 2024 г. Короткие статьи и описания плакатов. — Челябинск: Издательский центр ЮУрГУ, 2024. — P. 196.
2. *Goncharov S., Ershov Y., Sviridenko D.* Semantic Programming. — Seminarbericht, Humboldt-Universität zu Berlin, Sektion Mathematik, 1986.
3. *Manna Z., Waldinger R.* Synthesis: Dreams → Programs. — Software Engineering, IEEE Transactions on, SE-5, 294-328, 1979.
4. *Malyshkin V.* Active Knowledge, LuNA and Literacy for Oncoming Centuries. — Lecture Notes in Computer Science, vol. 9465, Springer, Cham, 2015.
5. *Valkovsky V. A., Malyshkin V. E.* Синтез параллельных программ и систем на вычислительных моделях. — 1988.
6. *Malyshkin V. E., Perepelkin V. A.* LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem. — Parallel Computing Technologies. PaCT 2011. Lecture Notes in Computer Science, vol 6873, Springer, Berlin, Heidelberg, 2011.
7. *Gorodnichev M., Lebedev D.* Semantic tools for development of high-level interactive applications for supercomputers. — J Supercomput 77, 11866–11880, 2021.
8. *Puschel M. et al.* SPIRAL: Code Generation for DSP Transforms. — Proceedings of the IEEE, vol. 93, no. 2, pp. 232-275, Feb. 2005.
9. *GitHub*. GitHub Copilot · Your AI pair programmer. — https://github.com/features/copilot.
10. *Ziegler A., Kalliamvakou E., Li X. et al.* Productivity Assessment of Neural Code Completion. — ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), 2022.
11. *Musen M. A., Team Protégé*. The Protégé Project: A Look Back and a Look Forward. — AI Matters, 1(4):4-12, 2015.
12. *Berthold M. R. et al.* KNIME: The Konstanz Information Miner. — In: Preisach, C., Burkhardt, H., Schmidt-Thieme, L., Decker, R. (eds) Data Analysis, Machine Learning and Applications. Studies in Classification, Data Analysis, and Knowledge Organization. Springer, Berlin, Heidelberg. 2008.
13. *Afgan E., Baker D., Batut B. et al.* The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. — Nucleic Acids Res. 46(W1):W537-W544, 2018.

14. Малышкин В. Э., Перепелкин В. А. Мультиагентный подход к повышению эффективности исполнения фрагментированных программ в системе LuNA // “Проблемы информатики”, 2023, № 3, с.55-67. DOI: 10.24412/2073-0667-2023-3-55-67.
15. *Cholia S., Sun T.* The NEWT Platform: An Extensible Plugin Framework for Creating ReSTful HPC APIs. — 9th Gateway Computing Environments Workshop, New Orleans, LA, USA, 2014.
16. *McAffer Jeff, Lemieux Jean-Michel*. Eclipse Rich Client Platform: Designing, Coding, and Packaging Java Applications. — 2005.
17. *Azad B.* Notes on the Eclipse Plug-in Architecture. — Bolour Computing, 2003.

Приложение А

Система управления базой фрагментов кода
СУБФК

Руководство пользователя.

Новосибирск 2025

Аннотация

Настоящее руководство пользователя (далее – Руководство) описывает функциональные возможности системы управления базой фрагментов кода, разработанной для хранения и стандартизации переиспользования ранее разработанных программ с целью автоматизации программирования через концепцию активных знаний в составе системы активных знаний.

В разделе «Условия выполнения программы» указаны условия, необходимые для выполнения программы (минимальный состав аппаратных и программных средств и т.п.).

В разделе «Выполнение программы» указана последовательность действий пользователя, обеспечивающих загрузку и выполнение программы, приведено описание функций, формата ввода данных, а также ответы программы на эти команды.

Оформление программного документа «Руководство пользователя» произведено по требованиям ЕСПД (ГОСТ 19.101-77, ГОСТ 19.103-77, ГОСТ 19.104-78, ГОСТ 19.105-78, ГОСТ 19.106-78, ГОСТ 19.505-79, ГОСТ 19.604-78).

СОДЕРЖАНИЕ

1 Назначение ПРОГРАММЫ 1

1.1 ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ 1

1.2 ЦЕЛИ СИСТЕМЫ 1

2 Условия выполнения программы 2

2.1 МИНИМАЛЬНЫЙ СОСТАВ АППАРАТНЫХ СРЕДСТВ 2

2.2 МИНИМАЛЬНЫЙ СОСТАВ ПРОГРАММНЫХ СРЕДСТВ 2

2.3 ТРЕБОВАНИЯ К ПЕРСОНАЛУ 2

3 ВЫПОЛНЕНИЕ ПРОГРАММЫ 3

3.1 ВХОД В СИСТЕМУ 3

3.2 ИНТЕРФЕЙС СИСТЕМЫ 3

3.3 ПЕРЕЧЕНЬ ЗАПРОСОВ К СИСТЕМЕ УПРАВЛЕНИЯ 3

3.4 КОДЫ ОШИБОК 5

Перечень терминов 7

1 Назначение Системы

1.1 ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

Программа СУБФК предназначена для хранения и стандартизации переиспользования уже разработанных программ предметной области. Она позволяет автоматизировать усилия разработчиков по поиску нужных подпрограмм, разбору того, какие подпрограммы лучше, а какие хуже, а также написанию самих программ, которые используют как свою основу другие подпрограммы.

1.2 ЦЕЛИ СИСТЕМЫ

1) Хранение загруженных в систему фрагментов кода.
2) Обработка запросов на переиспользование добавленных в систему фрагментов кода.

2 Условия выполнения программы

2.1 МИНИМАЛЬНЫЙ СОСТАВ АППАРАТНЫХ СРЕДСТВ

Для работы с программой требуется:

1) АРМ пользователя со следующими характеристиками:
   a. Процессор с тактовой частотой 2 ГГц или выше;
   b. Оперативная память 2 Гб или выше;

2.2 МИНИМАЛЬНЫЙ СОСТАВ ПРОГРАММНЫХ СРЕДСТВ

Аппаратная конфигурация рабочих мест должна быть достаточной для установки ОС Windows 10 или выше, ОС Ubuntu.

2.3 ТРЕБОВАНИЯ К ПЕРСОНАЛУ

Пользователь программы (оператор) должен обладать практическими навыками работы с командной строкой выбранной операционной системы, навыками работы с папками и файлами.

3 ВЫПОЛНЕНИЕ ПРОГРАММЫ

3.1 ВХОД В СИСТЕМУ

Работа с Системой по большей части происходит автоматически, другими компонентами системы активных знаний, однако при ручном использовании СУБФК начинается с ее запуска, который осуществляется следующим образом:

1) Скачать на компьютер .jar-файл релиза программы
2) С помощью java запустить .jar файл командой “java -jar ModuleLibrary.jar”
3) Нажмите клавишу **«Enter»**.
4) В командной строке отобразятся сообщения о инициализации системы.

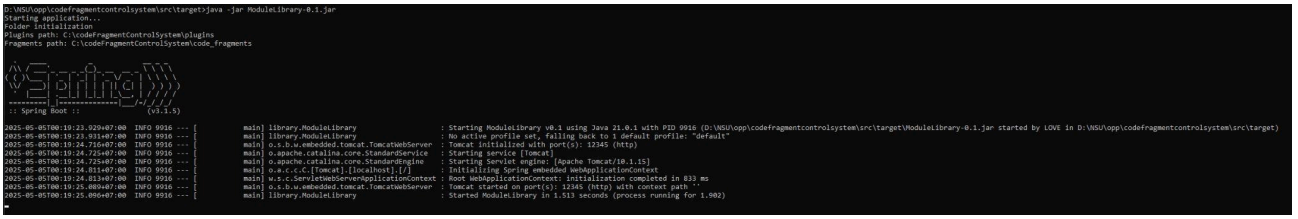

Рисунок 1. Успешный запуск программы

3.2 ИНТЕРФЕЙС СИСТЕМЫ

Интерфейс Системы отсутствует. Вся работа происходит путем отправки HTTP-запросов на localhost:12345

3.3. ПЕРЕЧЕНЬ ЗАПРОСОВ К СИСТЕМЕ УПРАВЛЕНИЯ

Для взаимодействия с системой нужно отправить HTTP-запрос по ссылке, соответствующей запросу. HTTP-ответ содержит ответ на запрос.

code_fragments
Назначение: возвращает список [id] фрагментов кода
Формат возвращаемых данных: JSON
Пример возвращаемого значения:

```
{
  "code_fragments":["b642e476-8d79-4783-b1da-a016b0d226aa"]
}
```

Пример запроса: /code_fragments

info
Назначение: возвращает метаинформацию о фрагменте кода
Формат возвращаемых данных: JSON
Пример возвращаемого значения:

```
{
  "package": "Convolution",
  "inputVarCount": 3,
```

”outputVarCount”: 2,
”description”: ”Correlation convolution of seismic traces. The report for the resulting correlograms includes the signal-to-noise ratio (SNR) and equivalent signal amplitude (A) in ADC units.”,
”name”: ”Correlation Convolution of Seismic Traces”,
”inputVarTypes”: [”any_path”, ”any_path”, ”int”],
”inputVarDescriptions”: [“Text file with a list of seismic traces in PC-A format”,Reference signal file in PC-A format”,Correlation interval in seconds”],
”outputVarTypes”: [”local_path”, ”local_path”],
”outputVarDescriptions”: [”Work report in .log format”,”Final convolved seismic trace file”],
”implementedPlugins”: [”ExeStartCProcedure”]
}

Дополнительные параметры: ?specific=name (вместо name поставить любой ключ из возвращаемого значения при запросе без specific, также можно комбинировать запросы specific: /info?specific=inputVarCount@outputVarCount@description@implementedPlugins)

Пример запроса: /b642e476-8d79-4783-b1da-a016b0d226aa/info

plugins

Назначение: возвращает список поддерживаемых плагинов

Формат возвращаемых данных: JSON

Пример возвращаемого значения:

{
"plugins":["ExeStartCProcedure", "getSO"]
}

Пример запроса: /"id"/plugins или /plugins

getNeededContentFor

Назначение: возвращает необходимые файлы фрагмента кода в зависимости от плагина

Формат возвращаемых данных: .tar архив. Скачивается архив с именем ”id фрагмента кода”.tar в котором лежит папка src со всеми нужными корневыми файлами фрагмента кода

Дополнительные параметры: ?plugin="plugin_name"

Пример запроса: /"id"/getNeededContentFor?plugin=ExeStartCProcedure

pluginProcedure

Назначение: выполняет указанный плагин для фрагмента кода

Формат возвращаемых данных: зависит от плагина

Пример запроса: /"id"/pluginProcedure?plugin=ExeStartCProcedure

add_package

Назначение: добавляет пакет в систему

Формат входных данных: packageName, MultipartFile (xml, .tar)

Тип запроса: POST

Дополнительная информация: В xml должен быть description файл. В файле должен быть .tar файл в котором лежит папка src с корневыми файлами фрагмента кода

Пример запроса:

curl -X POST .../add_package -H "Content-Type: multipart/form-data"
-F "packageName=packageName" -F "xml=@description.xml" -F "file=@test_send.tar"

remove_package

Назначение: удаляет пакет из системы

Формат входных данных: параметр URL
Пример запроса:
curl -X DELETE /remove_package?packageName=Convolution

Несколько примеров ответов на запросы:

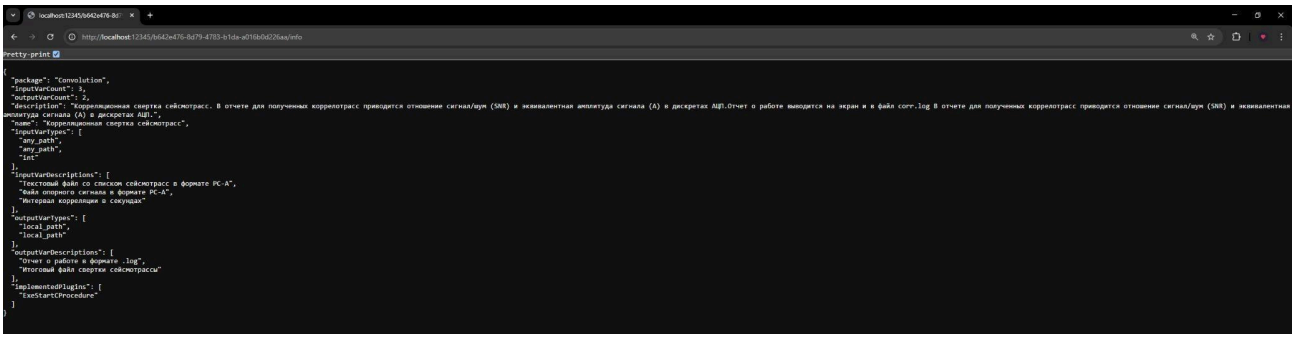

Рисунок 2. Запрос info

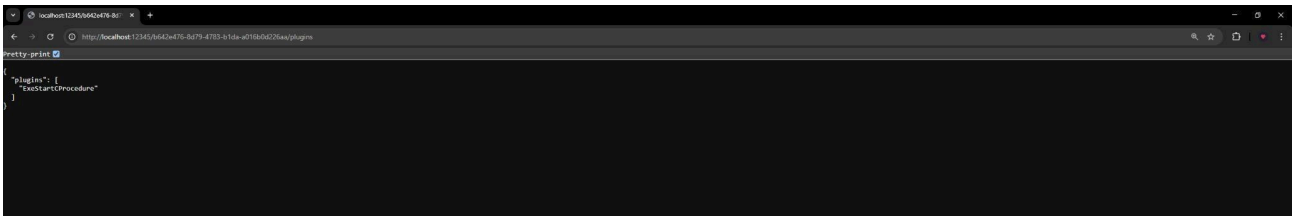

Рисунок 3. Запрос plugins

Рисунок 4. Запрос pluginProcedure

3.4. КОДЫ ОШИБОК

Код 500 (Внутренняя ошибка сервера)

Ошибка доступа к файлам плагина
Ошибка доступа к description файлу
Неверный формат description файла
Ошибка доступа к папке с плагинами
Ошибка создания .tar архива
Ошибка получения списка фрагментов кода
Ошибка получения списка плагинов
Ошибка записи в description файл
Ошибка доступа к хранилищу фрагментов кода
Ошибка создания папки для фрагмента кода
Ошибка распаковки исходных файлов
Неправильный формат плагина
Неизвестная ошибка

Код 400 (Неверный запрос)

Неизвестный плагин
Неизвестный запрос
Неизвестный метод фрагмента кода
Неизвестный фрагмент кода
Отсутствует запрошенное поле в description файле

Код 409 (Конфликт)

Фрагмент кода уже существует в базе

Код 200 (Успех)

Операция выполнена успешно

Перечень терминов

Термин	Определение
СУБФК	Система управления базой фрагментов кода
АРМ	Автоматизированное рабочее место

Приложение Б

Система управления базой фрагментов кода
СУБФК

Описание программы

Листов 10

Новосибирск 2025

Аннотация

В данном документе приведено описание системы управления базой фрагментов кода, разработанной для хранения и стандартизации переиспользования ранее разработанных программ с целью автоматизации программирования через концепцию активных знаний в составе системы активных знаний.

В документе описана общая структура системы. Оформление программного документа «Описание программы» произведено по требованиям ГОСТ 19.402-78 «ЕСПД. Описание программы» и ГОСТ 19.105-78 «Единая система программной документации (ЕСПД). Общие требования к программным документам (с Изменением № 1)».

СОДЕРЖАНИЕ

1 Назначение программы 4

2 Функциональные возможности 4

3 Условия эксплуатации 4

4 Входные и выходные данные 5

5 Вызов и загрузка 5

1. НАЗНАЧЕНИЕ ПРОГРАММЫ

Программа **СУБФК** предназначена для хранения и стандартизации переиспользования уже разработанных программ предметной области. Она позволяет автоматизировать усилия разработчиков по поиску нужных подпрограмм, разбору того, какие подпрограммы лучше, а какие хуже, а также написанию самих программ, которые используют как свою основу другие подпрограммы.вия выполнения программы

2. ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ

Программа **СУБФК** предназначена для хранения и стандартизации переиспользования уже разработанных программ предметной области. Она позволяет автоматизировать усилия разработчиков по поиску нужных подпрограмм, разбору того, какие подпрограммы лучше, а какие хуже, а также написанию самих программ, которые используют как свою основу другие подпрограммы.

3. УСЛОВИЯ ЭКСПЛУАТАЦИИ

Программа должна использоваться во время работы системы активных знаний, чтобы обеспечить ее конкретное функционирование (для хранения и стандартизации переиспользования фрагментов кода). Приложение ориентировано на внутреннее, закрытое использование, а также на опытных разработчиков, расширяющих количество фрагментов кода (интерфейс позволяет загружать новые фрагменты кода).

Программа **СУБФК** предназначена для использования на персональных компьютерах с операционной системой Linux. Заданные характеристики программы должны обеспечиваться при следующих климатических условиях эксплуатации:

1. Температура окружающего воздуха: от 0°C до +40°C;
2. Относительная влажность: до 80% при температуре не выше 25°C.

Обслуживание программы осуществляется посредством регулярных обновлений, которые выпускаются через сервис GitLab. Обновления включают исправления ошибок, улучшения производительности и новые функции. Пользователю рекомендуется устанавливать обновления для поддержания актуальности и надежности приложения.

Пользователи **СУБФК** делятся на два типа - инженеры знаний и рядовые пользователи. Для работы с программой **СУБФК** не требуется специальной квалификации, однако для доступа к некоторым функциям программы (например, добавление новых

форматов для фрагментов кода), пользователь должен обратиться к инженеру знаний, чтобы тот, в свою очередь включил в систему то, что не хватает пользователю (будь то фрагмент кода или плагин, поддерживающий определенные типы фрагментов кода). Пользователи должны обладать базовыми навыками работы с персональными компьютерами на платформе Linux. Обслуживание приложения (например, установка обновлений) не требует привлечения специализированного технического персонала. В случае возникновения сложностей рекомендуется обращаться в службу технической поддержки, предоставляемую разработчиком.

4. ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ

Входные данные программы **СУБФК** включают:

- фрагменты кода, сохраняемые пользователем (выбранные файлы и метаинформация) определенного формата
- HTTP-запросы, отправляемые пользователем для получения информации о фрагментах кода

Ввод данных должен производиться через HTTP-интерфейс.

5. ВЫЗОВ И ЗАГРУЗКА

Вызов и загрузку программы можно посмотреть в руководстве пользователя, также прилагаемом к программе.