|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ (ИУ5)\_\_\_\_\_

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ***

***НА ТЕМУ:***

***«******Модуль семантической организации файлов на основе метаграфового подхода»***

Студент ИУ5-83Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_В.А. Труфанов\_\_\_\_**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель ВКР **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_В.А. Михеев \_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Консультант **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_Ю.Е. Гапанюк \_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Нормоконтролер **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_Ю.Н. Кротов \_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

*2021 г.*

# РЕФЕРАТ

Расчетно-пояснительная записка 83 с., 33 рис., 31 табл., 40 источников, 3 приложения.

Объектом разработки является приложение, которое позволяет структурировать файлы и папки на основе добавляемой пользователем семантической информации.

Разработанная система позволяет создавать, редактировать и визуализировать семантические отношения, связывать семантические единицы с документами файловой системы, а также осуществлять по ним поиск.

Целью разработки является организация файлов, которая опирается на семантику вместо использования иерархии, диктуемой файловой системой и предоставляет средства отображения и поиска семантической информации на основе метаграфового подхода.

Расчетно-пояснительная записка состоит из трех частей.

Постановка задач разработки содержит описание предметной области, функциональных возможностей системы, процессов автоматизации, а также анализ аналогов и сравнение разрабатываемой системы с ними.

Исследовательская часть посвящена выбору способа хранения метаданных в разрабатываемой системе. Приведен подробный обзор существующих методов хранения дополнительной информации о файлах и их сравнение для определения наилучшего из них с учетом требований разработки. Также приводится обоснование применения метаграфового подхода для создания семантической модели проектируемой системы и описание ее возможностей.

Конструкторско-технологическая часть включает описание выбора технологий разработки, архитектуры системы, моделей хранения и отображения данных для хранения данных, а также интерфейса взаимодействия с пользователем.

СОДЕРЖАНИЕ

[РЕФЕРАТ 2](#_Toc73641672)

[СПИСОК ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ 5](#_Toc73641673)

[ВВЕДЕНИЕ 6](#_Toc73641674)

[1 ПОСТАНОВКА ЗАДАЧ РАЗРАБОТКИ 8](#_Toc73641675)

[1.1 Постановка задачи разработки 8](#_Toc73641676)

[1.2 Описание предметной области 9](#_Toc73641677)

[1.3 Перечень процессов, подлежащих автоматизации 11](#_Toc73641678)

[1.4 Функциональные возможности разрабатываемого модуля 11](#_Toc73641679)

[1.5 Выбор и обоснование критериев качества 13](#_Toc73641680)

[1.6 Анализ аналогов 14](#_Toc73641681)

[2 ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ 21](#_Toc73641682)

[2.1 Понятие семантической файловой системы 21](#_Toc73641683)

[2.2 Постановка задачи исследования 25](#_Toc73641684)

[2.3 Обзор способов хранения метаданных 25](#_Toc73641685)

[2.3.1 Хранение метаданных в файле средствами файловой системы 25](#_Toc73641686)

[2.3.2 Хранение метаданных в названии файла 31](#_Toc73641687)

[2.3.3 Хранение метаданных в дополнительном файле 32](#_Toc73641688)

[2.3.4 Хранение метаданных в базе данных 34](#_Toc73641689)

[2.4 Сравнение способов хранения метаданных 36](#_Toc73641690)

[2.4.1 Определение показателей важности локальных критериев 37](#_Toc73641691)

[2.4.2 Нормировка показателей и оценка Парето-оптимальности 39](#_Toc73641692)

[2.4.3 Выбор наилучшего варианта на основе интегральных критериев 42](#_Toc73641693)

[2.5 Обоснование метаграфового подхода 45](#_Toc73641694)

[3 КОНСТРУКТОРСКО-ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ 49](#_Toc73641695)

[3.1 Выбор технологий разработки программного изделия 49](#_Toc73641696)

[3.1.1 Electron 49](#_Toc73641697)

[3.1.2 TypeScript 51](#_Toc73641698)

[3.1.3 База данных 52](#_Toc73641699)

[3.1.4 Высокоуровневое средство работы с БД 56](#_Toc73641700)

[3.1.5 Cytoscape.js 62](#_Toc73641701)

[3.2 Архитектура системы 63](#_Toc73641702)

[3.3 Диаграмма последовательности взаимодействия с системой 64](#_Toc73641703)

[3.4 Проектирование моделей хранения данных 66](#_Toc73641704)

[3.4.1 Метаграфовая модель 67](#_Toc73641705)

[3.4.2 Инфологическая модель 67](#_Toc73641706)

[3.4.3 Даталогическая модель 68](#_Toc73641707)

[3.4.4 Модели ORM и визуализации 70](#_Toc73641708)

[3.5 Интерфейс взаимодействия с пользователем 72](#_Toc73641709)

[ЗАКЛЮЧЕНИЕ 79](#_Toc73641710)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 80](#_Toc73641711)

[ПРИЛОЖЕНИЕ A ГРАФИЧЕСКИЕ МАТЕРИАЛЫ 84](#_Toc73641712)

[ПРИЛОЖЕНИЕ B ТЕХНИЧЕСКОЕ ЗАДАНИЕ 95](#_Toc73641713)

[ПРИЛОЖЕНИЕ C ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ 102](#_Toc73641714)

# СПИСОК ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

1. Документ – объекты файл и папка файловой системы.
2. Тег (категория) – семантическая единица информации, связываемая с документами
3. Семантика (документа) - информация более высокого уровня, чем уровень физического хранения на диске.
4. ОС – операционная система
5. API - интерфейс, который определяет взаимодействие между несколькими программными средствами.
6. БД – база данных.
7. СУБД - система управления базами данных
8. SQL - это предметно-ориентированный язык, предназначенный для управления данными в СУБД.
9. ORM - технология программирования, связывающая БД с концепциями объектно-ориентированных языков программирования
10. Фреймворк - программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.
11. Кроссплатформенность - способность программного обеспечения работать с несколькими аппаратными платформами или ОС.

# ВВЕДЕНИЕ

Современный уровень развития информационных технологий позволяет людям оперировать невообразимым ранее объемом данных на персональных устройствах. В связи с этим возникает потребность в поиске новых способов организации и хранения данных, а также их представления пользователю в более наглядной форме.

Исторически самым популярным способом организации файлов на логическом уровне стала иерархическая файловая структура, которая позволяет сопоставить каждому файлу и каталогу путь в дереве, корнем которого является так называемый корневой каталог. Такой подход достаточно хорошо описывает фиксированные структуры (каталоги с исходным кодом, программными библиотеками) и однотипные файлы (журналы работы программ, общие статические ресурсы), поэтому он стал самым распространенным в большинстве операционных систем.

Несмотря на определенные достоинства иерархической организации, безусловно, очевидным недостатком является невозможность интуитивно (без использования символических ссылок и других специальных средств) структурировать информацию в более сложных ситуациях, когда требуется сохранить разнородные файлы, обладающими множеством атрибутов с точки зрения пользователя, и помещение файла в единственный узел дерева ограничивает имплицируемую семантику.

В связи с этим было предложено использовать более абстрактное понятие категорий или тегов, которое отражает семантическую единицу и служит для идентификации информации. Присвоение тегов помимо иерархических отношений один-ко-многим позволяет создавать отношения вида многие-ко-многим, когда один и тот же набор тегов может сопоставляться с произвольным числом документов.

Возможности тегирования в той или иной форме впоследствии были добавлены в большинство файловых систем в форме атрибутов (метаданных) файлов, а также был отчасти реализован функционал поиска по этим свойствам (например, с помощью расширенного синтаксиса запросов (AQS) в Windows). Тем не менее, унификация способов организации метаданных для всего множества файлов труднодостижимая задача, поэтому многие большие проекты в этой области так и не были реализованы. В связи с этим популярными стали разработки упрощенных моделей семантической системы, которые не реализуют весь функционал файловой системы, а работают поверх существующих способов организации файлов.

Такой принцип используется и в данной работе, целью которой является создание модуля семантической организации файлов. Помимо присвоения семантических единиц к объектам файловой системы требуются средства отображения семантической модели в удобном для пользователя виде. В связи с этим в работе используется метаграф, являющийся особым видом сложного графа, на основе которого, как будет продемонстрировано, возможно построение семантической модели.

# ПОСТАНОВКА ЗАДАЧ РАЗРАБОТКИ

# Постановка задачи разработки

Организация хранения и поиска данных являются одними из основных задач, которые решают информационные системы. Популярные способы организации данных существуют уже многие десятилетия, и современный объем информации и ее разнообразие выдвигают новые требования.

В современных операционных системах в подавляющем большинстве используется иерархические файловые системы, которые рассчитаны на организацию и поиск файлов в дереве каталогов. На данный момент у многих пользователей персональных компьютеров в файловой системе накапливаются сотни тысяч файлов, а для того, чтобы найти требуемый файл надо помнить его путь (или часть) или имя, что вызывает затруднения ввиду такого количества документов.

В связи с этим актуальными становятся средства поиска файлов на основе семантики, добавляемой пользователем или содержащейся в самом файле (метаданные или его содержимое), а также визуализации отношений между различными семантическими единицами для наглядного представления связанных документов. При этом, как правило, интерфейс хранения данных остается неизменным, и возможности семантической организации вводятся на базе существующих файловых систем. Для визуализации семантических отношений, которые зачастую неоднозначны и динамичны, требуются специальные инструменты на базе сложных графов, одним из которых является используемый в работе метаграф.

Таким, образом основной целью проектирования является разработка модуля, способного организовывать файлы семантически на основе метаграфового подхода.

Для достижения этой цели требуется решить следующий задачи проектирования:

1. Исследовать предметную область, определить функциональные задачи.
2. Сравнить существующие аналоги проектируемой системы
3. Проанализировать способы хранения метаданных
4. Обосновать применение метаграфового подхода
5. Разработать архитектуру программного обеспечения.
6. Разработать структуру базы данных
7. Разработать метаграфовую модель
8. Реализовать приложение с интегрированной метаграфовой моделью и базой данных.
9. Провести тестирование информационно-программного продукта.
10. Провести отладку программного продукта.
11. Оформить техническую документацию.

# Описание предметной области

Предметной областью в разрабатываемом модуле является совокупность сущностей, объединенных процессом семантической организации файлов и визуализации семантических отношений. Для иллюстрации предметной области рассмотрим пример, в котором демонстрируются ограничения иерархического способа организации файлов.

На диске требуется расположить два файла с фотографиями: Фото-01.jpg и Фото-02.jpg. С первой фотографией требуется ассоциировать информацию о событии (концерт), месте (Москва), и годе (2021), со второй фотографией – только о месте и годе.

Для задания таких ассоциаций в иерархической модели приходится создать несколько папок для каждого элемента информации и создавать копии фотографий в этих папках, что представлено на рисунке 1. Так вторая фотография помещается дважды с путями: Фото/Года/2021/Фото-02.jpg и Фото/Места/Москва/Фото-02.jpg Очевидным недостатком, помимо дублирования файлов, является тот факт, что при поиске файла отображается несколько расположений и, не имея контекстной информации, нельзя понять, что один и тот же файл связан с двумя атрибутами.

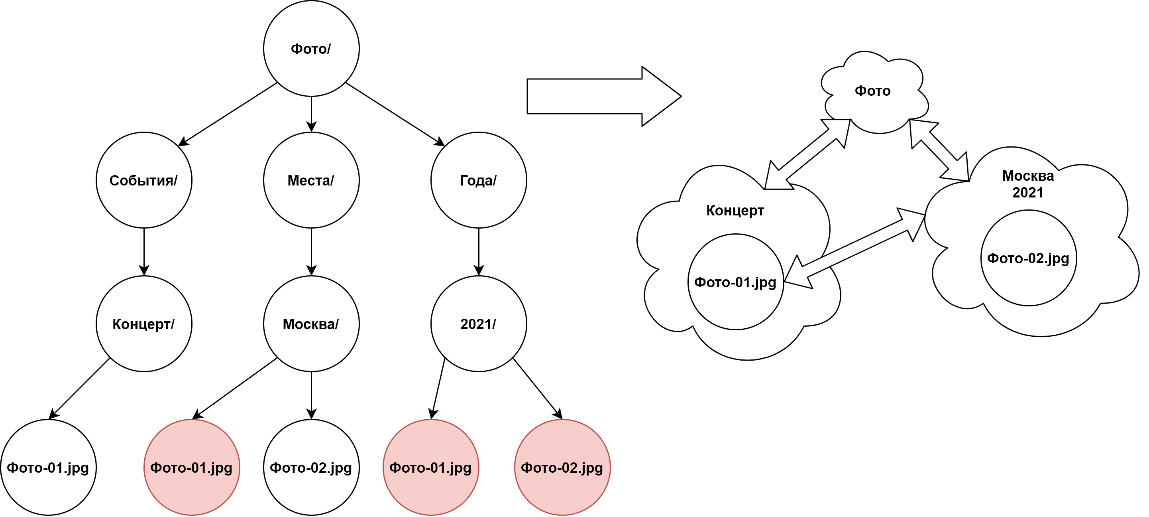


Рисунок 1 – Иллюстрация ограничений иерархической модели

В семантической модели, представленной на рисунке 1 справа, файлы ассоциируются с атрибутами или группой атрибутов, которые в свою очередь могут быть связаны с другими атрибутами. При этом файл хранится в файловой системе в единственном экземпляре и его возможно найти по обращению к какому-либо из связанных с ним семантической информации.

В общем виде предметная область представлена на рисунке А.1.

Пользователи проектируемого модуля – пользователи ПК, которые хотят изменить способ работы с файлами на своем устройстве.

Рассмотрим процесс организации файлов с их стороны. Для пользователей необходимо иметь возможность добавлять теги к папкам и файлам, строить между тегами отношения на визуализации и осуществлять по ним поиск, а также возможность сохранить полученную семантическую модель в проекте.

В связи с этим можно выделить следующие основные сущности предметной области:

1. Проект – сущность, отражающая информацию об одной выстраиваемой семантической модели
2. Файл – сущность, отражающая информацию о файле
3. Каталог – сущность, отражающая информацию о каталоге
4. Сущности, отражающие семантическую модель – описываются в разделе Метаграфовая модель

# Перечень процессов, подлежащих автоматизации

В процессе разработки следует автоматизировать следующие процессы внутри системы:

1. Создание и редактирование семантической модели в реальном времени
2. Отображение выстраиваемой модели на визуализации
3. Поиск информации по семантической модели
4. Сохранение семантических моделей и связанных с ней данных
5. Отображение и редактирование списка проектов
6. Привязка документов к элементам семантической модели
7. Проверка доступности открытия документов
8. Коммуникация между частями приложения

# Функциональные возможности разрабатываемого модуля

Функциональные возможности разрабатываемого модуля описаны в Техническом Задании. Ниже приведен список функционала приложения с пояснениями.

1. Создание проекта, содержащего рабочий метаграф – создание специальной сущности для хранения семантической модели
2. Редактирование названия проекта
3. Создание категорий – создание семантических единиц, к которым прикрепляются документы
4. Создание одной связи и множества связей – создание связей между категориями
5. Объединение категорий в мета-категорию – создание обобщенной категории (целого), включающего множество категорий (частное)
6. Включение категорий в существующую мета-категорию
7. Удаление мета-категории без вложенных объектов – удаление обобщенной категории без удаления включенных элементов
8. Удаление категорий и мета-категорий
9. Удаление связей
10. Редактирование названий созданных элементов метаграфа
11. Отображение категорий, подкатегорий и связей в виде метаграфа – визуализация построенных семантических отношений в наглядном виде
12. Привязка файлов и папок к созданным элементам – соединение документов и ранее созданных семантических единиц
13. Возможность заново привязать документ при его переименовании или перемещении – восстановление связей в случае изменений файлов вне приложения
14. Открепление файлов и папок от созданных элементов
15. Дублирование элементов метаграфа – создание копи части семантической модели с прикрепленными файлами
16. Поиск по файлам и папкам, добавленным в проект, по именам с отображением прикрепленных категорий и мета-категорий
17. Поиск элементов метаграфа, добавленных в проект, по присвоенным именам с отображением прикрепленных документов
18. Удаление проекта, добавленного в приложение

# Выбор и обоснование критериев качества

Для разрабатываемого программного изделия приоритетными являются следующие критерии качества:

1. Платформа
2. Поддерживаемые типы файлов
3. Тегируемые объекты
4. Отношения тегов
5. Возможности построения процессов
6. Отслеживание перемещений

Поясним значения выбранных критериев.

**Платформа** – поддерживаемые приложением распространенные операционные системы (семейств ОС).

**Поддерживаемые типы файлов –** совокупность форматов файлов, к которым можно привязать теги.

**Тегируемые объекты** – объекты файловой системы, к которым можно применять теги (файлы или папки)

**Отношения тегов** – возможности задания отношений между тегами, является одной из основных характеристик мощности семантической модели.

**Возможности построения процессов –** возможности применяемой семантической модели описать процессы и его этапы для тегирования документов, что делает ее более совершенной.

**Отслеживание перемещений** – возможности приложения следить за изменениями в расположении и именовании файлов на уровне иерархической файловой системы, чтобы синхронизировать их с метаданными.

Оценим коэффициенты важности критериев при помощи методы балльной оценки [1] (таблица 1).

Таблица 1 – Метод балльной оценки

|  |  |  |  |
| --- | --- | --- | --- |
| Код фактора | Название критерия | Балл | Коэффициент важности локального критерия по методу бальной оценки ( |
| X1 | Платформа | 80 | 0,184 |
| X2 | Поддерживаемые типы файлов | 40 | 0,092 |
| X3 | Тегируемые объекты | 50 | 0,115 |
| X4 | Отношения тегов | 100 | 0,230 |
| X5 | Возможности построения процессов | 90 | 0,207 |
| X6 | Отслеживание перемещений | 75 | 0,172 |

Коэффициент важности локального критерия по методу бальной оценки ( рассчитывается по формуле 1:

где – балл критерия,

– количество критериев.

# Анализ аналогов

По функционалу наиболее близким к разрабатываемой системе продуктом является теговый менеджер. Были найдены 10 теговых менеджеров, которые можно считать аналогами проектируемого модуля:

1. Tabbles (B1)
2. SetTags (B2)
3. Tagging for Windows (B3)
4. XnView MP (B4)
5. Stagsi (B5)
6. TagSpaces (B6)
7. Elyse (B7)
8. Ritt (B8)
9. Hydrus (B9)
10. FenrirFS (B10)

Краткое описание каждого из этих продуктов изложено ниже.

**Tabbles** – большой программный продукт для Windows, реализованный по модели файловый менеджер с базой данных. Среди основных особенностей – возможность использовать облачный сервис и синхронизацию тегов между устройствами пользователя, групповое редактирование тегов, тесная интеграция с проводником и наличие отдельного компонента (теггера) для отслеживания перемещения/удаления файлов вне программы. Также поддерживается автотегирование на основе правил.

**SetTags** – теговый менеджер для Windows, для хранения тегов используются ярлыки, которые, в отличие от символических ссылок, могут отслеживать перемещение и переименования файлов с помощью средств ОС, для прикрепления тегов к почтовым сообщениям, контактам и URL используется база данных. Теги добавляются в имя файла, существует возможность тегирования папок. Присутствует предпросмотр некоторых типов файлов, а также автоматическое добавление тегов на основе пути и имени файла.

**Tagging for Windows** – бесплатный программный продукт для Windows, реализованный в виде расширения для проводника, поэтому основной функционал доступен напрямую через интерфейс проводника. Теги добавляются как к файлам, так и папкам, присутствует возможность создания иерархия тегов. Помимо расширения доступно отдельное приложение для поиска и тегирования файлов. Этот продукт использует внутреннюю базу данных, но благодаря интеграции с проводником, теги сохраняются при перемещении или переименовании файлов.

**XnView MP** – кроссплатформенный продукт, рассчитанный, в первую очередь, на работу с изображений, который также поддерживает и другие типы файлов. Т.к. основное внимание уделено работе с изображениями, в приложение реализован быстрый предпросмотр эскизов изображений, а также предусмотрено автоматическое создание тегов (категорий) на основе метаданных в формате IPTC или XMP Subject. Поддерживается группировка тегов, также возможно присвоить теги папкам. В продукте используется стандартный подход хранения тегов в базе данных, теги при перемещении или переименовании сохраняются частично.

**Stagsi** – условно-бесплатный продукт для Windows, реализованной по стандартной модели с базой данных. Из особенностей присутствует доска тегов, содержимой которой можно связать с сочетаниями клавиш для быстрого применения определенного набора тегов. Поддерживается группировка тегов, предпросмотр изображений, а также других файлов при ручном импорте эскизов. Переименование и перемещение файлов не отслеживается, но приложение вычисляет контрольные суммы для файлов, что делает возможным заново связать файл с тегами при новом импорте.

**TagSpaces** – Кроссплатформенный продукт с открытым исходным кодом, который не использует распространенный подход с базой данных. Теги, хранятся в имени файла и разделяются пробелами, что вводит ограничения на возможные символы и длины тегов. За счет такой реализации теги сохраняются при перемещении или удалении файлов, а также доступны из других программ. Присутствует возможность группировки тегов и присвоения тегов папкам. Также доступен предпросмотр файлов и поиск с применением логических операций.

**Elyse** – кроссплатформенный продукт, использующий базу данных. В приложении присутствует специальное окно для группировки тегов и составления иерархий. Поддерживается предпросмотр отдельных файлов, поиск с логическими операциями, а также добавление тегов к папкам и папки-представления, которые содержат результаты поиска с логическим и операциями и автоматически обновляются при добавлении новых файлов в базу данных. Перемещение и переименование файлов не отслеживается, требуется новый импорт.

**Ritt** – современное Windows-приложение, использующее базу данных на основе JSON. Папки рассматриваются как источники данных, после добавления которых в программе становится доступным тегирование (как файлов, так и папок). Теги организуются в деревья, при этом применение тега нижнего уровня не добавляет родительские теги. Присутствует предпросмотр для некоторых типов файлов. Перемещение и переименование файлов обрабатывается только из самой программы.

**Hydrus** – кросс-платформенный продукт, испытавший влияние имиджбордов, в первую очередь, представляет собой средство архивирования данных. После добавления файла в программу, он сохраняется во внутренней базе данных, имя изменяется на хеш, поэтому перемещение или переименование не предусмотрено. Благодаря такому подходу реализована возможность скачивать публичные репозитории из сети и популярных онлайн-галерей. Поддерживаются основные форматы мультимедиа, для которых предусмотрены эскизы, и некоторые другие распространенные типы. Для группировки тегов используются именные пространства, также поддерживается создание иерархий. Присутствует полностью настраиваемая система рейтингов.

**FenrirFS** – бесплатное Windows-приложение, рассчитанное на работу с большими папками сложной структуры. Приложение автоматически присваивает цветные метки (теги) на основе пути файлов, а также формирует «умные» папки (например, документы). Самостоятельно такие папки возможно создать с помощью SQL-скриптов. Возможность тегирования доступна только для файлов. Приложение отслеживает перемещения и переименования файлов, возможно привязать теги заново самостоятельно.

Проведем сравнение аналогов с системой (В11), разрабатываемой в рамках этой работы, с использованием метода взвешенной суммы нормированных показателей сравнения [1]. Исходные данные для сравнения приведены в таблице А.1.

Проведем перевод вербальных показателей в числовые для каждого критерия (таблицы 2-7).

Таблица 2 – Вербально-числовая шкала для X1

|  |  |  |
| --- | --- | --- |
| Windows,Mac, Linux | Windows, Mac | Windows |
| 3 | 2 | 1 |

Таблица 3 – Вербально-числовая шкала для X2

|  |  |  |
| --- | --- | --- |
| Любые | Преимущественно мультимедиа | Преимущественно изображения |
| 4 | 2 | 1 |

Таблица 4 – Вербально-числовая шкала для X3

|  |  |  |
| --- | --- | --- |
| Файлы и папки | Файлы, папки(платно) | Файлы |
| 2 | 1,5 | 1 |

Таблица 5 – Вербально-числовая шкала для X4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Группировка, вложенные теги, иерархии, связи | Группировка и вложенные теги | Группировка, иерархия | Вложенные теги, цвет | Группировка, цвет | Группировка |
| 5 | 3 | 3 | 2 | 2 | 1 |

Таблица 6 – Вербально-числовая шкала для X5

|  |  |
| --- | --- |
| Да | Нет |
| 1 | 0 |

Таблица 7 – Вербально-числовая шкала для X6

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Есть (умное отслеживание) | Есть | Есть (Windows-ярлыки) | Частично (хеши) | Частично |
| 5 | 4 | 3,5 | 2 | 1 |

Рассчитаем коэффициенты нормализации для показателей критериев “чем больше, тем лучше” по формуле 2:

где – максимальное значение -ого локального критерия среди сравниваемых вариантов решения

Результаты расчета представлена в таблице 8.

Таблица 8 – Нормированные значения сравниваемых вариантов и коэффициенты важности

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Код факт  ора | Весовой коэффициент | Нормированные значения показателей | | | | | | | | | | |
| В1 | В2 | В3 | В4 | В5 | В6 | В7 | В8 | В9 | В10 | В11 |
| Х1 | 0,184 | 0,333 | 0,333 | 0,333 | 1 | 0,333 | 1 | 0,667 | 0,333 | 1 | 0,333 | 1 |
| Х2 | 0,092 | 1 | 1 | 1 | 0,25 | 1 | 1 | 1 | 1 | 0,5 | 1 | 1 |
| Х3 | 0,115 | 1 | 1 | 1 | 1 | 0,5 | 0,75 | 1 | 1 | 0,5 | 0,5 | 1 |
| Х4 | 0,230 | 0,4 | 0,2 | 0,6 | 0,2 | 0,6 | 0,4 | 0,6 | 0,6 | 0,6 | 0,2 | 1 |
| Х5 | 0,207 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| X6 | 0,172 | 1 | 0,7 | 0,8 | 0,2 | 0,4 | 0,8 | 0,2 | 0,2 | 0,8 | 0,8 | 0,2 |

Проверим рассматриваемые варианты на Парето-оптимальность (таблица 9).

Таблица 9 – Сравнение вариантов на Парето-оптимальность

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | В1 | В2 | В3 | В4 | В5 | В6 | В7 | В8 | В9 | В10 | В11 |
| В1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| В2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| В3 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| В4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| В5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| В6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| В7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| В8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| В9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| В10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| В11 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| Результат сравнения | 0 | 2 | 0 | 1 | 1 | 0 | 1 | 3 | 0 | 3 | 0 |
| Парето-оптимальность варианта | да | нет | да | нет | нет | да | нет | нет | да | нет | да |

Варианты В1, В3, В6, В9, В11 являются Парето-оптимальными и требуется их дальнейшее сравнение. Рассчитаем взвешенную сумму для этих вариантов (таблица 10).

Таблица 10 – Взвешенная сумма локальных критериев

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Код фактора | Весовой коэффициент | Нормированные значения показателей | | | | |
| В1 | В3 | В6 | В9 | В11 |
| X1 | 0,184 | 0,333 | 0,333 | 1 | 1 | 1 |
| X2 | 0,092 | 1 | 1 | 1 | 0,5 | 1 |
| X3 | 0,115 | 1 | 1 | 0,75 | 0,5 | 1 |
| X4 | 0,230 | 0,4 | 0,6 | 0,4 | 0,6 | 1 |
| X5 | 0,207 | 0 | 0 | 0 | 0 | 1 |
| X6 | 0,172 | 1 | 0,8 | 0,8 | 0,8 | 0,2 |
|  | | 0,533 | 0,544 | 0,592 | 0,563 | 0,862 |

Произведем ранжирование вариантов по предпочтительности:

В результате применения методы взвешенной суммы нормированных показателей сравнения можно сделать вывод, что проектируемая система является лучшей среди рассмотренных вариантов.

Не было найдено ни одного аналога разрабатываемой системы, в котором реализованы возможности создания процессов с привязкой документов, что повышает значимость этой работы.

## **ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ**

# **Понятие семантической файловой системы**

**Семантика файла** в общем случае может быть заключена в любой информации более высокого уровня, чем уровень физического хранения. Например, расширение файла или магическое число, определяющее тип файла, ассоциативная информация (ключевые слова в файле, которые характеризуют содержимое), структурная информация (логическая организация данных, включая отношения между файлами), поведенческая информация (семантика просмотра и модификации, управление изменениями), информация об окружении (создатель, история изменений) и т.д.

**Семантическая файловая система** — это система хранения информации, обеспечивающая гибкий ассоциативный доступ к содержимому системы путем автоматического извлечения атрибутов из файлов с помощью преобразователей для конкретных типов файлов [2]. Преобразователь должен знать о специфике типа файла и его семантически значимых атрибутов.

Архитектурно все семантические файловые системы можно разделить на 2 вида: интегрированные и расширенные. Интегрированные подходы включают расширенные семантические функции непосредственно в файловую систему. Расширенные подходы предоставляют эти функции эволюционно, дополняя традиционный интерфейс файловой системы, что позволяет традиционным интерфейсам манипулирования файлами оставаться неизменными [3].

Интегрированные файловые системы предоставляют пользователю новую улучшенную файловую систему, которая встроена в существующие файловые системы. Эти системы обеспечивают интегрированную модель данных, в которой данные файла и метаданные файла представлены в одной модели данных, и эти две модели неявно синхронизированы. Такие системы могут предоставлять язык определения типов, на подобие языка описания интерфейсов (IDL), который используется для определения модели данных файловой системы для общего хранения содержимого файла и метаданных. Если структурная информация переносится из приложения в файловую систему, необходимо определить процедуры доступа, специфичные для каждого типа файла. Затем эти процедуры должны быть доступны для приложений.

Преимущества интегрированного подхода включают:

1. Возможность модели данных соответствовать модели, используемой в приложении, оптимизация хранения модели на диске на уровне файловой системы.
2. Доступность метаданных - пользователи и приложения могут иметь доступ к метаданным, которые обычно скрыты в файлах.

Недостатками такого подхода являются:

1. Необходимость обновления операционной системы или переход на совершенно новую.
2. Необходимость адаптировать приложения для использования новой модели данных на уровне файловой системы.

Расширенные семантические файловые системы оставляют без изменений традиционные интерфейсы файловой системы, обеспечивая дополнительную абстракцию содержимого файла. В таком случае инструменты запросов и управления информацией о файлах могут быть более интеллектуальными, чем в интегрированном подходе [3].

В большинстве реализаций введенные абстракции используются либо для доставки запросов, либо для доставки индексов. Доставка запросов направляет независимый от хранилища пользовательский запрос к соответствующим файлам. Доставка индексов извлекает содержимое файлов и делает их доступными в виде метаданных (индексов) для системы запросов на уровне пользователя.

В архитектуре расширенных файловых систем файлы представляются элементами логических коллекций или доменов. Каждый домен управляется менеджером домена. Домены можно сравнить с представлениями в базах данных в том смысле, что они являются подмножеством полуструктурированного информационного пространства. Спецификации доменов могут быть основаны на традиционных понятиях ОС, таких как каталоги и регулярные выражения в именах файлов. Менеджеры доменов обеспечивают объектно-ориентированное представление содержимого файлов в своих доменах (выделение значимых атрибутов) и предоставляют возможность выполнять запросы в своих доменах по неиндексированным атрибутам файла. Обычно менеджеры доменов собирают семантику на основе свойств и содержимого типов файлов. Для определенных или новых типов файлов только разработчики приложений требуется определить логику менеджера доменов. Вся собранная семантика файлов в итоге сохраняется в семантическом хранилище в структуре, которая удобна для семантической поисковой системы, чтобы предоставлять пользователям семантический поиск через интерфейс [4].

Расширенные файловые системы обычно предоставляют собственный язык обмена содержимым файла для различных элементов архитектуры. Посредством этого языка в архитектуре с доставкой индексов индексы и другая метаинформация доставляется от менеджеров домена к вышестоящим брокерам знаний. Это позволяет брокерам знаний напрямую обрабатывать запросы приложений. В архитектуре доставки запросов запросы и результаты запросов обмениваются между архитектурными элементами.

Брокеры знаний обладают знаниями, необходимыми для обработки запроса. Они осведомлены о требуемой декомпозиции запроса, переводе запроса и сопоставлении результатов [3].

В обобщенном виде архитектура расширенных семантических файловых систем представлена на рисунке 2.

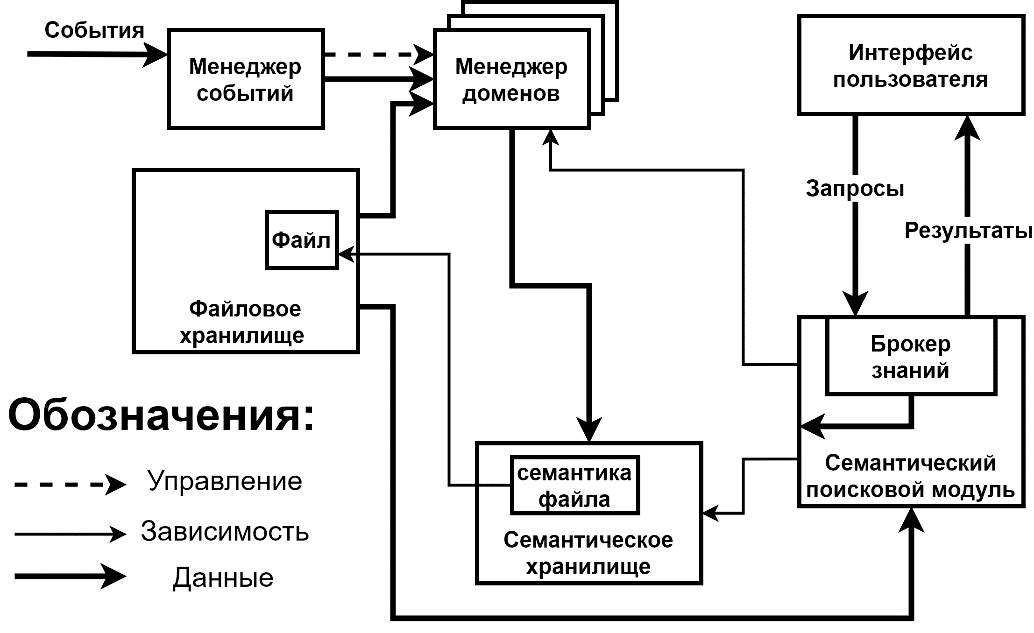


Рисунок – Обобщенная архитектура расширенной семантической файловой системы

Достоинства расширенного подхода:

1. Не требуется модифицировать существующие файловые системы, интерфейсы работы с файлами остаются прежними
2. Гибкая модульная архитектура, позволяющая заменять отдельные компоненты усовершенствованными вариантами

Недостатки расширенного подхода:

1. Необходимость создания инструментов слежения за изменением содержимого файлов и их местоположением
2. Необходимость поддержания синхронизации между извлеченными в хранилище метаданными (например, в базу данных) и самими файлами.

С течением времени расширенные подходы стали более распространенными, чем интегрированные, поскольку они более гибко адаптируются к системам, используемым конечными пользователями.

На сегодняшний день не существует широко используемых реализаций семантических файловых систем общего назначения. Сложность разработки таких систем связана с неоднозначностью выделения семантической информации из большого разнообразия файлов, а также проблемами предоставления удобного интерфейса, заменяющего стандартные, привычные пользователям средства доступа к файлам, поэтому некоторые большие проекты, такие как WinFS [5]**,** так и не были реализованы.

В связи с этим достаточно популярным стала реализация отдельных компонентов или упрощенной модели семантической системы.

# **Постановка задачи исследования**

Существуют различные подходы к организации хранения метаданных в семантическом хранилище. Наиболее распространенными вариантами хранения метаданных являются следующие варианты:

1. Хранение метаданных в файле средствами файловой системы
2. Хранение метаданных в названии файла
3. Хранение метаданных в дополнительном файле
4. Хранение метаданных в базе данных

Каждый из них обладает определенными преимуществами и недостатками, поэтому универсального способа хранения не существует. В связи с этим, одной из первостепенных задач, которую требуется решить при создании модуля семантической организации файлов, является выбор архитектуры семантического хранилища.

# **Обзор способов хранения метаданных**

# **Хранение метаданных в файле средствами файловой системы**

Первым подходом, который естественно предложить для хранения метаданных является помещение их в файл на уровне файловой системы, так как такой подход не требует проектирования дополнительного хранилища. Существует два варианта этого подхода, реализации которых в ходе развития операционных и файловых систем стали достаточно близки.

**Расширенные файловые атрибуты**

Первый вариант – хранение метаданных в расширенных файловых атрибутах. Расширенные атрибуты файлов — это свойства файловой системы, которые позволяют пользователям связывать компьютерные файлы с метаданными, не интерпретируемыми файловой системой, тогда как стандартные атрибуты имеют цель, строго определенную файловой системой (например, разрешения или записи времени создания и модификации) [1]. Такие атрибуты обычно ограничены по размеру до значения, значительно меньшего, чем максимальный размер файла (например, размер может составлять блок файловой системы).

Доступность расширенных атрибутов файлов, как и обычных зависит от поддержки базовой файловой системой семейством операционных систем.

**Расширенные файловые атрибуты в Linux**

В Linux файловые системы (ext2, ext3, ext4, и другие) поддерживают расширенные атрибуты и работу с ними с помощью утилиты xattr, если они включены в конфигурация ядра. Любой обычный файл или каталог может иметь расширенные атрибуты, состоящие из имени и связанных данных. Имя должно быть нуль-терминированной строка, с префиксом идентификатора пространства имен и точкой. В настоящее время существует четыре пространства имен: пользовательское, доверенное, безопасность и система. Пространство имен пользователя не имеет ограничений в отношении именования или содержимого. Системное пространство имен в основном используется ядром для списков управления доступом к файлу [6].

Ядро Linux и файловая система могут устанавливать ограничения на максимальное количество и размер расширенных атрибутов, которые могут быть связаны с файлом, в частности имя атрибута ограничено 255 байтами, а значение атрибута ограничено 64 кБ [7].

Некоторые файловые системы, такие как ReiserFS (и исторически ext2 и ext3), требуют, чтобы файловая система была смонтирована с опцией монтирования user\_xattr, чтобы можно было использовать расширенные атрибуты пользователя.

В текущих реализациях файловых систем ext2, ext3 и ext4 общее количество байтов, используемых именами и значениями всех расширенных атрибутов файла, должно соответствовать одному блоку файловой системы (1024, 2048 или 4096 байтов) [7].

Доступ к расширенным атрибутам осуществляется как к атомарным объектам. Чтение (getxattr) извлекает все значение атрибута и сохраняет его в буфере. Запись (setxattr) заменяет любое предыдущее значение новым значением. Пример работы с API xattr представлен на рисунке 3:

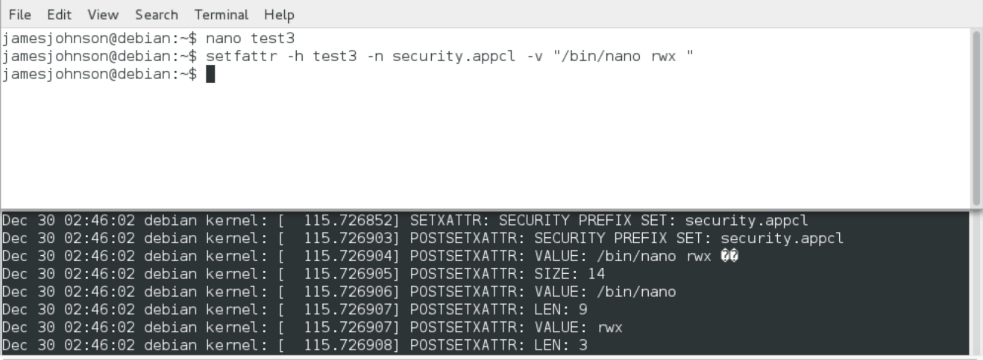


Рисунок – Установка расширенных атрибутов с xattr

**Расширенные файловые атрибуты в macOS**

macOS поддерживают расширенные атрибуты, используя B\*-дерево атрибутов файла в файловой системе HFS+, которая позволяет использовать именованные форки (этот вариант хранения атрибутов будет рассмотрен ниже). Хотя именованные форки в HFS + поддерживают произвольно большие объемы данных, поддержка расширенных атрибутов реализована через встроенные атрибуты, ограничивая их размер до размера, который может поместиться в одном узле B\*-дерева. Однако HFS+ поддерживает произвольное количество таких форков, и точного ограничения на количество расширенных атрибутов нет [6].

macOS поддерживает работу с расширенными атрибутами через Linux-подобное API на базе xattr, при этом нет никаких ограничений на пространства имен (как в xattr). В самой системе эти атрибуты используются для установки флага карантина, который указывает на то, что приложение было загружено из Интернета и требует полных проверок безопасности, а также для прикрепления сведений о веб-сайте, с которого был загружен файл [8].

**Расширенные файловые атрибуты в Windows**

В Windows NT расширенные атрибуты ограниченной длины поддерживаются файловыми системами FAT, HPFS и NTFS, использующиеся для реализации Unix-подобных разрешений. Подсистема Windows для Linux использует их для аналогичных целей, сохраняя режим файла Linux, владельца, идентификатор устройства и время файлов в расширенных атрибутах [6].

В файловой системе NTFS каждая единица информации, связанная с файлом реализована как атрибут файла. В частности, предусмотрен атрибут $EA, который служит для расширения индекса атрибутов [9]. Этот атрибут также используется как расширенный, например, браузер хранит в нем информацию о статусе файла (идентификация загрузки из интернета).

Кроме того, NTFS может хранить расширенные атрибуты произвольной длины в форме альтернативных потоков данных (ADS), что будет рассмотрено далее.

Можно выделить следующие достоинства такого варианта хранения атрибутов:

1. Помещение атрибутов на уровне файловой системы гарантирует их эффективное и безопасное хранение
2. Данные атрибутов останутся доступными даже после удаления используемого семантического программного обеспечения
3. Атрибуты доступны для других методов доступа, помимо специализированного программного обеспечения. Например, система полнотекстового поиска может проиндексировать их содержимое.
4. Атрибуты связаны с файлом независимо от переименования и его местоположения – файловая система обеспечивает связь метаданных с файлом на протяжении всего его существования.

Недостатками расширенных атрибутов являются:

1. Отсутствие унифицированного подхода к хранению атрибутов в различных файловых системах
2. Возможные ограничения на права доступа к файлу, влияющие на доступность атрибутов (в частности, права доступа к расширенным атрибутам в Linux напрямую связаны с правами доступа к самому файлу, т.е. невозможно изменить атрибуты у файла, доступного только для чтения) [10]
3. Отсутствие универсальной обработки атрибутов в утилитах по работе с файлами – не все средства архивирования, резервного копирования и клонирования учитывают расширенные атрибуты и поэтому они могут быть удалены после завершения операций
4. Необходимость создания специального средства поиска для эффективного доступа к атрибутам – требуется построение индекса и использование языка запросов для поисков по нему вместо линейного обхода дерева.

**Форки и альтернативные потоки данных**

Вторым вариантом, близким по смыслу с расширенными файловыми атрибутах является хранение атрибутов в форках. Форк — это набор данных, связанных с объектом файловой системы. Файловые системы без форков допускают только один набор данных для содержимого файла, в то время как файловые системы с форками допускают несколько таких наборов [11]. Каждый непустой файл должен иметь как минимум один форк, часто типа по умолчанию, и в зависимости от файловой системы файл может иметь одну или несколько других связанных форков, которые, в свою очередь, могут содержать как данные, являющиеся неотъемлемой частью файла, так и метаданные.

В отличие от расширенных атрибутов, которые обычно имеют фиксированный размер, форки могут иметь переменный размер, возможно, даже больший, чем основной форк данных файла [11]. При этом размер файла является суммой размеров каждого форка.

**Форки в macOS**

Форки в macOS используются в иерархической файловой системе (HFS). HFS позволяла объекту файловой системы иметь два типа форков: форк данных и форк ресурсов.

Форк ресурсов был разработан для хранения некомпилируемых данных, которые будут использоваться графическим интерфейсом системы, таких как локализуемые текстовые строки, значок файла, или меню и диалоговые окна, связанные с приложением. Однако эта функция была очень гибкой, поэтому были найдены дополнительные применения, такие как разделение документа текстового редактора на контент и представление с последующим сохранением каждой части в отдельных ресурсах.

Впоследствии в обновленной файловой системе HFS+ была добавлена возможность связать с файлом произвольное количество пользовательских именованных форков в дополнение к традиционным форкам данных и ресурсов [11]. На их основе была реализована поддержка расширенных файловых атрибутов, рассмотренных до этого.

**Форки в Windows**

В Windows вместо форков используется понятие альтернативных потоков данных (ADS). Файл в файловой системе NTFS имеет один (или несколько) связанных с ним потоков данных. Поток данных по умолчанию названия не имеет (например, dirpath\filename::$DATA), но альтернативные потоки данных имеют имена, например: dirpath\filename:streamname:$DATA [12].

Альтернативные потоки NTFS были введены в Windows NT, чтобы позволить службам для Macintosh (SFM) хранить форки ресурсов. Также альтернативные потоки используется системой в браузере, чтобы отмечать файлы, загруженные с внешних сайтов, как потенциально небезопасные для запуска [13].

Альтернативные потоки не отображаются в проводнике Windows, и их размер не входит в размер файла. Когда файл копируется или перемещается в другую файловую систему без поддержки ADS, альтернативные потоки данных не сохраняются [13]. Управление альтернативными потоками реализуется с помощью сценариев командной оболочки PowerShell (рис. 4).

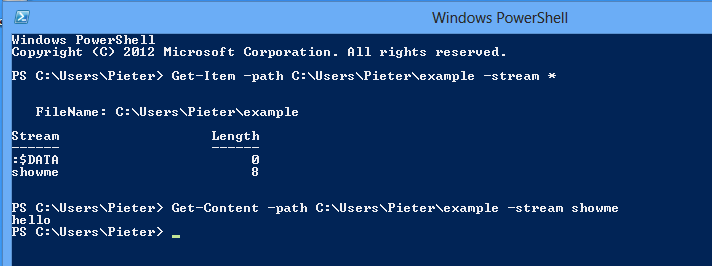


Рисунок – Запись информации в ADS с помощью PowerShell

Достоинства и недостатки форков и потоков данных в целом такие же, что и у расширенных атрибутов ввиду близости этих вариантов, за исключением того, что фактических ограничений на количество данных в форках и потоках не предусмотрено.

# **Хранение метаданных в названии файла**

Метаданные возможно хранить напрямую в названии файла. Для этой цели применяется особая схема именования файлов, в которой теги отделяются от фактического названия файла специальными символами, например, img [tag1 tag2].jpg – название тегов заключено в квадратные скобки [14]. Такой вариант хранения, безусловно, достаточно спорный – фактически стирается граница между семантикой названия файла и его атрибутами, так как они содержатся логически в одном поле данных. Также возникает определенные ограничения - максимальная длина пути к файлу в ОС, возможный набор символов в названии папок и файлов. Однако, будучи встроенным в имя файла, тег прикрепляется к нему и может быть удален только путем переименования файла. Это решает проблему согласованности метаданных и соответствующего файла при его перемещении. Теги, встроенные в имя файла, просто синхронизируются со облачными хранилищами и, безусловно, доступны для поиска и чтения любым другим программным обеспечением для работы с файлами.

Таким образом, можно выделить следующие недостатки данного варианта хранения метаданных:

1. Ограничения на длины путей и имен файлов в системах (в частности, 255 символов для названий в большинстве файловых систем [15])
2. Ограничения набора символов в названии файлов и возможной кодировки (разрешенные символы варьируются в различных файловых системах [15])
3. Изменение семантики названия объекта файловой системы
4. Редактирование названия файла сторонней программой, не осведомленной о его специальном формате, влечет за собой потерю метаданных

К достоинствам помещения метаданных в название можно отнести:

1. Сохранение связи метаданных с файлом при его перемещении – фактически метаданные становятся составляющей названия файла
2. Доступ сторонних средств поиска к атрибутам, так как фактически требуется поиск по названию объектов файловой системы, что дает возможность их использования в качестве инструмента поиска в проектируемой программе

# **Хранение метаданных в дополнительном файле**

Дополнительные (sidecar) файлы представляют собой файлы, в которых хранятся данные, которые не поддерживаются форматом исходного файла или содержат дополнительные данные, в том числе метаданные [16].

В большинстве случаев связь между исходным файлом и дополнительным файлом основана на имени – используется то же базовое имя, что и в исходном файле, но другое расширение. Возможно также помещение дополнительных файлов во вложенную скрытую папку, для того чтобы сохранить консистентный вид исходной файловой структуры [14]. Пример такого расположения дополнительных файлов c расширением json представлен на рисунке 5:

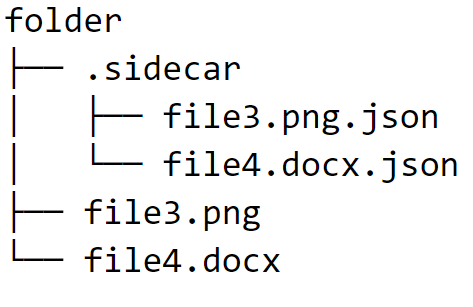


Рисунок – Размещение дополнительных json-файлов в скрытой папке

Универсальной схемы расположения атрибутов для дополнительных файлов нет. Например, возможно использование собственной схемы на базе формата обмена данными json: атрибуты, представленные парами тип-значение, помещаются в одномерный массив [17]. Тем не менее, существует стандарт хранения метаданных, использующий гибридный вариант на основе дополнительных файлов и встраивания атрибутов в сам файл - Extensible Metadata Platform (XMP), расширяемая платформа метаданных.

XMP стандартизирует модель данных для определения и обработки расширяемых метаданных. Стандарт также содержит рекомендации по встраиванию информации XMP в популярные форматы медиа-файлов и документов без нарушения их читабельности приложениями, не поддерживающими XMP [18]. Метаданные в качестве альтернативы или в случае неподдерживаемого формата файла хранятся в дополнительном файле.

Заданная модель данных XMP может использоваться для хранения любого набора свойств метаданных. Это могут быть простые пары имя-значение, структурированные значения или списки значений. Стандарт XMP определяет пространства имен для наборов основных свойств (например, пространство имен для набора элементов метаданных Dublin Core) [18]. Пользовательские пространства имен можно использовать для расширения модели данных. Для работы с XMP моделями разработан ряд специальных программ, позволяющих читать и изменять метаданные в большом количестве форматов файлов.

Итак, подход с дополнительным файлом обладает следующими преимуществами:

1. Отсутствие ограничений на размеры и типы атрибутов
2. Возможность применения произвольной, а также стандартизированной (для некоторого количества типов файлов) схемы хранения атрибутов

Недостатками данного подхода можно признать:

1. Отсутствие согласованности метаданных с файлами при их перемещении
2. Необходимость создания средств поиска в случае с произвольной схемой или использование сторонних программ при работе с XMP

# **Хранение метаданных в базе данных**

Наконец, рассмотрим подход хранения атрибутов файлов в базе данных. Внедрение инструментов СУБД поверх файловой системой – достаточно известная схема, существует такое понятие как файловые системы баз данных — это файловая система на основе базы данных, которая вместо или в дополнение к иерархическому структурированному управлению файлы идентифицирует их по характеристикам, таким как тип файла, тема, автор или аналогичные метаданные [19].

Благодаря использованию средств БД возможно решить не только задачу хранения метаданных, но также создания модели данных (схемы), построения отношений между различными моделями (на основе реляционной алгебры) и реализации языка запросов к ним (на основе инструкций SQL), т.е. целое множество задач, возникающих при построении семантической файловой системы.

База данных также обладает рядом свойств, обеспечивающих надежное хранение метаданных: ACID-транзакции, средства резервного копирования и восстановления, средства обеспечения безопасности [20].

Более того, некоторые СУБД предлагают встроенные инструменты для хранения файлов в файловой системе с реализацией к ним через БД. Так, в СУБД MS SQL Server существует технология FILESTREAM — хранилище данных, позволяющее хранить данные на NTFS диске и управлять ими, используя движок базы данных. СУБД создает каталог для хранения журнала и каталоги для хранения данных, имеющие имена в виде GUID [21]. В упрощенном виде эта технология представлена на рисунке 6.

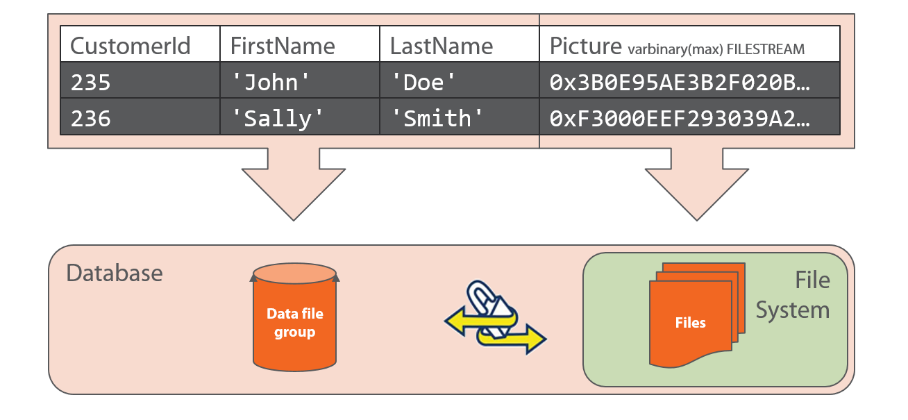


Рисунок – Упрощенная схема FILESTREAM

Такой подход использовался в расширенной семантической файловой системе WinFS. Метаданные в этой системе хранились в СУБД MS SQL Server с включенной функцией FILESTREAM. На уровне схемы были разработаны базовые модели элемента и отношений, от которых наследовались все схемы для конкретных типов файлов, образующих коллекцию элементов (домен) [22]. На основе SQL, а также специального программного API становилось возможным осуществлять расширенные семантические запросы (например, «номера телефонов всех людей, которые живут в Москве, каждый из которых более 100 раз появлялся в коллекции фотографий и с которыми я переписывался по электронной почте в течение последнего месяца»), а также дополнять существующие схемы собственными типами.

Некоторые аспекты файловой системы базы данных используются в многих системах управления веб-контентом для эффективного поиска документов по метаданным, а также в больших распределенных файловых системы, таких, как Google File System [19].

Таким образом, применение подхода с хранением метаданных в базе данных дает следующие преимущества:

1. Отсутствие ограничений на количество атрибутов и схему их размещения
2. Отсутствие необходимости использования стороннего языка запросов и реализации быстрого поиска атрибутов
3. Универсальность хранения метаданных – большинство современных БД имеют коннекторы и драйверы для всех основных операционных систем

Основным недостатком этого подхода является необходимость в реализации специального программного модуля для контроля согласованности атрибутов и файлов при их перемещении, т.к. метаданные изолированы от файловой системы. Также централизованное хранение метаданных означает потерю атрибутов для всех файлов в случае повреждения базы данных.

# **Сравнение способов хранения метаданных**

После рассмотрения всех вариантов хранения метаданных произведем сравнение рассмотренных на основе архитектурных особенностей и эксплуатационных показателей.

В качестве показателей сравнения выбраны следующие критерии:

1. **Независимость от платформы и файловой системы** (X1)–универсальность способа хранения метаданных
2. **Максимальный размер атрибутов** (X2) – ограничения размеров метаданных
3. **Возможности построения схем данных** (X3) – ограничения возможных модели метаданных
4. **Возможности поиска метаданных** (X4) – существующие механизмы извлечения семантики
5. **Встроенный язык запросов** (X5) – существующие способы запроса семантики
6. **Встроенная синхронизация между файлами и метаданными** (X6) – обеспечение целостности метаданных и связанных файлов

В качестве вариантов рассматриваются следующие способа хранения, описанные раннее:

1. Расширенные атрибуты (В1)
2. Форки / ADS (В2)
3. Название файла (В3)
4. Дополнительный файл (В4)
5. БД (В5)

Сравним эти варианты, выполнив следующие задачи:

1. Определение показателей важности локальных критериев, используя метод базового критерия и метод балльной оценки, на основе среднего значения полученных показателей;
2. Перевод вербальных показателей в числовые и их нормировка, оценка Парето-оптимальности;
3. Выбор наилучшего варианта на основе следующих интегральных критериев: взвешенная сумма локальных критериев, близость к идеалу, метод Борда, сравнение полученных интегральных оценок методом Борда.

# **Определение показателей важности локальных критериев**

Для расчета показателей важности методом базового критерия разбиваем показатели на группы важности:

Первая группа (менее значимые) – Х2, Х5

Вторая группа (более значимы, чем первая группа в 2 раза) – Х3, Х4

Третья группа (более значимы, чем первая группа в 4 раза) – Х1, Х6

Получаем:

– количество групп показателей сравнения способов хранения

– количество показателей, которые соответственно входят в состав первой, второй и третьей группы

– коэффициенты, которые показывают степень превосходства 2-ой, и 3-ей группы над критериями 1-ой группы

Получим,

Таким образом, получены значения весовых коэффициентов, приведенные для всех критериев в таблице 11.

Таблица 11 – Метод базового критерия

|  |  |  |
| --- | --- | --- |
| Код фактора | Название критерия | Коэффициент важности локального критерия ( |
| X1 | Независимость от платформы и файловой системы | 0,286 |
| X2 | Максимальный размер атрибутов | 0,071 |
| X3 | Возможности построения схем данных | 0,143 |
| X4 | Возможности поиска метаданных | 0,143 |
| X5 | Встроенный язык запросов | 0,071 |
| X6 | Встроенная синхронизация между файлами и метаданными | 0,286 |

Теперь оценим коэффициенты важности критериев при помощи методы балльной оценки, используя формулу 1 (таблица 12).

Таблица 12 – Метод балльной оценки

| Код фактора | Название критерия | Балл | Коэффициент важности локального критерия ( |
| --- | --- | --- | --- |
| X1 | Независимость от платформы и файловой системы | 100 | 0,230 |
| X2 | Максимальный размер атрибутов | 40 | 0,092 |
| X3 | Возможности построения схем данных | 80 | 0,184 |
| X4 | Возможности поиска метаданных | 70 | 0,161 |
| X5 | Встроенный язык запросов | 60 | 0,138 |
| X6 | Встроенная синхронизация между файлами и метаданными | 90 | 0,207 |

Наконец, оценим средние значения весовых коэффициентов критериев в таблице 14 по формуле 3 (таблица 13).

где ,

– количество критериев.

Таблица 13 – Средние значения весовых коэффициентов

|  |  |  |  |
| --- | --- | --- | --- |
| Код фактора |  |  |  |
| X1 | 0,286 | 0,230 | 0,258 |
| X2 | 0,071 | 0,092 | 0,082 |
| X3 | 0,143 | 0,184 | 0,163 |
| X4 | 0,143 | 0,161 | 0,152 |
| X5 | 0,071 | 0,138 | 0,105 |
| X6 | 0,286 | 0,207 | 0,246 |

# **Нормировка показателей и оценка Парето-оптимальности**

Значения критериев качества для всех способов хранения метаданных, а также полученные показатели важности критериев представлены в таблице 14.

Таблица 14 – Исходные значения критериев качества способов хранения метаданных

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Код фактора | Весовой коэффициент | В1 | В2 | В3 | В4 | В5 |
| Х1 | 0,258 | Нет | Нет | Есть (совпадающие разрешенные символы) | Есть | Есть |
| Х2 | 0,082 | Блок файловой системы | Неограничен | До 255 символов | Неограничен | Неограничен |
| Х3 | 0,163 | Ограничены (пары ключ-значение) | Неограничен (поток данных) | Ограничен (список в строке) | Неограничен (файл, имеется стандарт) | Неограничен |
| Х4 | 0,152 | Ограничены (сторонние средства и индекс) | Ограничены (сторонние средства и индекс) | Есть (стандартные средства) | Ограничены (сторонние средства и индекс) | Расширенные |
| Х5 | 0,105 | Нет (сторонние средства) | Нет (сторонние средства) | Нет (сторонние средства) | Есть (сторонние средства стандарта) | Есть |
| Х6 | 0,246 | Полная (с осведомленными инструментами) | Полная (с осведомленными инструментами) | Полная | Нет (дополнительный модуль отслеживания) | Нет (дополнительный модуль отслеживания) |

Проведем перевод вербальных показателей в числовые для каждого критерия (таблицы 15-20).

Таблица 15 – Вербально-числовая шкала для X1

|  |  |  |
| --- | --- | --- |
| Есть | Есть (совпадающие разрешенные символы) | Нет |
| 1 | 0,5 | 0 |

Таблица 16 – Вербально-числовая шкала для X2

|  |  |  |
| --- | --- | --- |
| Неограничен | Блок файловой системы | До 255 символов |
| 4 | 2 | 1 |

Таблица 17 – Вербально-числовая шкала для X3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Неограничен | Неограничен (файл, имеется стандарт) | Неограничен (поток данных) | Ограничены (пары ключ-значение) | Ограничен (список в строке) |
| 5 | 4,5 | 4 | 2 | 1 |

Таблица 18 – Вербально-числовая шкала для X4

|  |  |  |
| --- | --- | --- |
| Расширенные | Есть (стандартные средства) | Ограничены (сторонние средства и индекс) |
| 4 | 2 | 1 |

Таблица 19 – Вербально-числовая шкала для X5

|  |  |  |
| --- | --- | --- |
| Есть | Есть (сторонние средства стандарта) | Нет (сторонние средства) |
| 2 | 1 | 0 |

Таблица 20 – Вербально-числовая шкала для X6

|  |  |  |
| --- | --- | --- |
| Полная | Полная (с осведомленными инструментами) | Нет (дополнительный модуль отслеживания) |
| 2 | 1,5 | 0 |

Рассчитаем коэффициенты нормализации для показателей критериев на основе формулы 2 (таблица 21):

Таблица 21 – Нормированные значения сравниваемых вариантов и коэффициенты важности

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Код фактора | Весовой коэффициент | Нормированные значения показателей | | | | |
| В1 | В2 | В3 | В4 | В5 |
| Х1 | 0,258 | 0 | 0 | 0,5 | 1 | 1 |
| Х2 | 0,082 | 0,5 | 1 | 0,25 | 1 | 1 |
| Х3 | 0,163 | 0,4 | 0,8 | 0,2 | 0,9 | 1 |
| Х4 | 0,152 | 0,25 | 0,25 | 0,5 | 0,25 | 1 |
| Х5 | 0,105 | 0 | 0 | 0 | 0,5 | 1 |
| Х6 | 0,246 | 0,75 | 0,75 | 1 | 0 | 0 |

Проверим рассматриваемые варианты на Парето-оптимальность (таблица 22).

Таблица 22 – Сравнение вариантов на Парето-оптимальность

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | В1 | В2 | В3 | В4 | В5 |
| В1 | 0 | 0 | 0 | 0 | 0 |
| В2 | 1 | 0 | 0 | 0 | 0 |
| В3 | 0 | 0 | 0 | 0 | 0 |
| В4 | 0 | 0 | 0 | 0 | 0 |
| В5 | 0 | 0 | 0 | 1 | 0 |
| Результат сравнения | 1 | 0 | 0 | 1 | 0 |
| Парето-оптимальность варианта | нет | да | да | нет | да |

Варианты В2, В3, В5 являются Парето-оптимальными и требуется их дальнейшее сравнение.

# **Выбор наилучшего варианта на основе интегральных критериев**

Рассчитаем следующие интегральные критерии: взвешенная сумма локальных критериев и близость к идеалу (таблица 23).

Таблица 23 – Взвешенная сумма локальных критериев и близость к идеалу

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Код фактора | Весовой коэффициент | Нормированные значения показателей | | |
| В2 | В3 | В5 |
| X1 | 0,258 | 0 | 0,5 | 1 |
| X2 | 0,082 | 1 | 0,25 | 1 |
| X3 | 0,163 | 0,8 | 0,2 | 1 |
| X4 | 0,152 | 0,25 | 0,5 | 1 |
| X5 | 0,105 | 0 | 0 | 1 |
| X6 | 0,246 | 0,75 | 1 | 0 |
|  | | 0,435 | 0,504 | 0,759 |
|  | | 0,685 | 0,598 | 0,496 |

Далее проведем ранжирование Парето-оптимальных вариантов по методу Борда на основе формулы 4 (таблица 24).

где ,

– количество критериев, – количество вариантов,

– ранг показателя сравнения среди всех вариантов

Таблица 24 – Ранжирование по методу Борда

| Код фактора | В2 | В3 | В5 |
| --- | --- | --- | --- |
| Х1 | 1 | 2 | 3 |
| Х2 | 1,5 | 3 | 1,5 |
| Х3 | 2 | 3 | 1 |
| Х4 | 3 | 2 | 1 |
| Х5 | 2,5 | 2,5 | 1 |
| Х6 | 2 | 1 | 3 |
|  | 12 | 13,5 | 10,5 |

Все полученные интегральные оценки приведены в таблице 25.

Таблица 25 – Интегральные оценки по 3 критериям

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Интегральный критерий | В2 | В3 | В5 | Больше / меньше лучше |
| Взвешенная сумма | 1 | 2 | 3 | Больше |
| Близость к идеалу | 1,5 | 3 | 1,5 | Меньше |
| Метод Борда | 2 | 3 | 1 | Меньше |

Наконец выберем вариант на основе сравнения интегральных показателей с помощью метода Борда (таблица 26).

Таблица 26 – Сравнение по интегральным критериям с методом Борда

|  |  |  |  |
| --- | --- | --- | --- |
| Интегральный критерий | В2 | В3 | В5 |
| Взвешенная сумма | 1 | 2 | 3 |
| Близость к идеалу | 3 | 2 | 1 |
| Метод Борда | 2 | 3 | 1 |
|  | 6 | 7 | 5 |

Произведем ранжирование вариантов по предпочтительности:

Таким образом, на основе результатов сравнения вариантов хранения метаданных по интегральным критериям для разрабатываемого модуля выбран подход помещения атрибутов в базу данных.

# **Обоснование метаграфового подхода**

В рамках проектируемого модуля семантической организации файлов также возникает задача выбора схемы для хранения модели представления метаданных. Для упрощения реализации было решено не разделять файлы на домены и не создавать универсальный набор атрибутов, а уделить внимание отношениям между метаданными. Таким образом, семантика добавляемых метаданных сводится к присвоению тега объекту файловой системы.

Выбор модели хранения был произведен с опором на исследование [23], в котором проводилось сравнение между удобством в организации электронных писем по папками и с применением тегов пользователями почтовых сервисов. Респонденты в течение пяти дней организовывали поступающие по выбранной теме письма в соответствии со своими предпочтениями. После этого им было предложено изобразить на бумаге тот подход, которые они считают оптимальным, а также описать с каким трудностями они столкнулись при организации файлов в используемых сервисах. В результате анализа полученной информации, были сделаны следующие выводы:

1. Отношения части/целого или общего/конкретного можно легко выразить с помощью иерархии папок. Система тегов, которая позволяет связать подмножества тегов в иерархии, может сделать эти отношения более выразимыми.
2. Подмножества тегов могут быть связаны друг с другом во взаимоисключающие отношения или их полную группу для достижения разделения информации, которое происходит фактически, когда информация организована в папках.
3. Люди думают об информации способами, выходящими далеко за рамки репрезентативной способности папок, тегов или их сочетания - некоторые респонденты располагали информацию относительно времени и шагов рабочего процесса. При этом расположение документов меняется со временем в процессе добавления новой информации, подлежащей категоризации.

В качестве примера приведем два варианта организации предметной области «публикация книги» с использованием тегов, который респондент изобразил как желаемый результат (рис. 7 и 8).

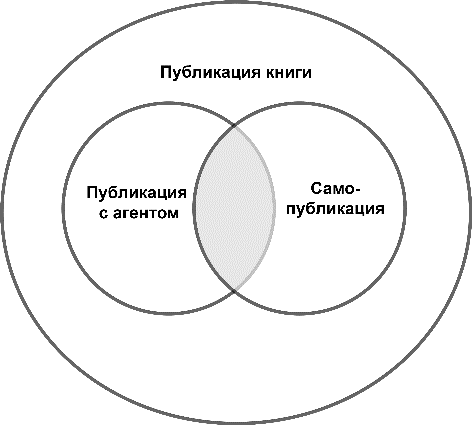


Рисунок – Набросок «Публикация книги» в виде диаграммы Венна

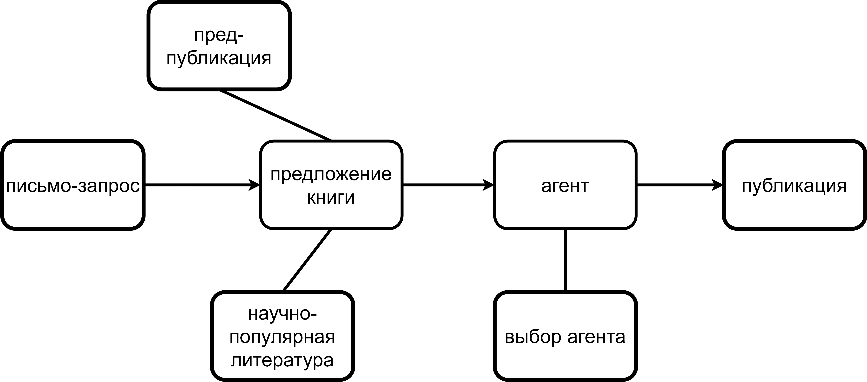


Рисунок – Набросок «Публикация книги», похожий на рабочий процесс

Исходя из результатов исследования, становится понятно, что следует использовать в качестве модели для метаданных сложный граф, элементы которого обладают возможностью включения подграфа для возможности создания иерархии тегов, а также множества вершин с направленными связями для создания рабочего процесса и его этапов.

Таким сложным графом является метаграф, формальное определение которого приведено в работе [24].

Метаграф – это один из видов сложных графов, который помимо множества вершин (V) и связывающих их ребер (E), содержит множества метавершин (MV) и метаребер (ME), т.е метаграф:

Фрагмент метаграфа (MG) – часть метаграфа, содержащая подмножество вершин (метавершин) и ребер (метаребер) метаграфа. Метавершины, вершины, метаребра и ребра характеризуется множеством атрибутов. Метавершина отличается от обычной вершины тем, что включает фрагмент метаграфа MG. Метаребро отличается от обычного ребра возможностью включения фрагмента метаграфа MG. Ребра и метаребра также характеризуются исходной вершиной (метавершиной) и конечной вершиной (метавершиной). Также как обычные ребра, метаребра могут быть направленные и ненаправленные.

Исходя из формального описания метаграфа возможно предложить следующие варианты построения организации тегов для представленных выше набросков. Отношения части/целого возможно выразить посредством вложения вершин в метавершину, что соответствует созданию иерархии папок. Расположение информации относительно шагов рабочего процесса возможно выразить с помощью метаребра, которое служит средством описания процессов. Примеры моделей метаграфа, соответствующих рисункам 7-8 приведены на рисунках 9-10. На рисунках А.2-А.3 представлены пары модель-набросок.

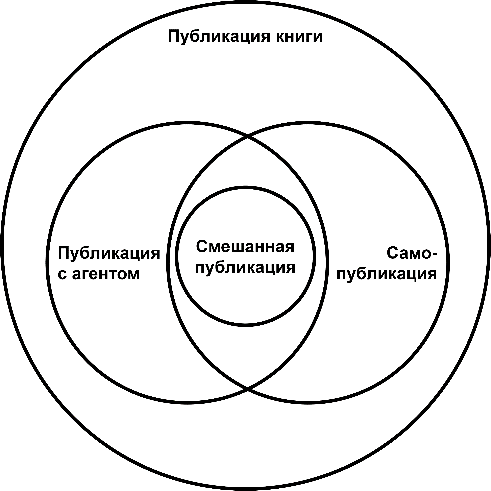


Рисунок – Метаграф-иерархия «Публикация книги», соответствующий рис. 7

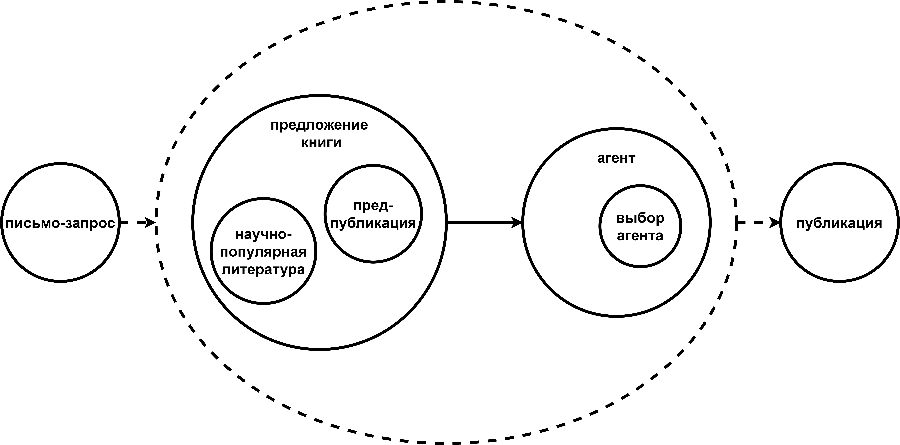


Рисунок – Метаграф-процесс «Публикация книги», соответствующий рис. 8

Тем не менее, ввиду ограничений используемой в системе метаграфовой модели описание процессов представляется с помощью метавершины и вложенных в нее этапов процесса, которым соответствуют вершины.

# КОНСТРУКТОРСКО-ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ

# **Выбор технологий разработки программного изделия**

Разработку модуля было решено проводить на базе веб-инструментов для того, чтобы существовала возможность достаточно простого дополнения приложение сетевым функционалом в будущем. Более того, для веб-разработки существует большое число крупных и поддерживаемых библиотек с открытым исходным кодом, что облегчает проектирование программного изделия.

Для создания и функционирования модуля необходимы следующие компоненты:

1. Фреймворк для построения кроссплатформенных приложений
2. Язык программирования для реализации основной логики модуля
3. База данных, которая подходит для кроссплатформенных клиентских приложений
4. Высокоуровневое средство для работы с БД из кода модуля
5. Библиотека визуализации для отображения метаграфа и изменения его топологии

# **Electron**

Electron — это программный фреймворк с открытым исходным кодом, разработанный и поддерживаемый GitHub. Он позволяет разрабатывать настольные приложения с графическим интерфейсом пользователя на базе веб-технологий: сочетает в себе механизм рендеринга Chromium и среду выполнения Node.js [25]. Electron лежит в основе многих популярных проектов, включая Visual Studio Code, Discord, WhatsApp, Skype и т.д.

Chromium — это версия веб-браузера Google Chrome с открытым исходным кодом. Он имеет в основном один и тот же код и функции, с небольшими отличиями и разным лицензированием. Content Module — это основной код, который позволяет Chromium отображать веб-страницы в независимых процессах и использовать ускорение графического процессора. Он включает движок рендеринга Blink и движок JavaScript V8 [26].

Проект Node.js изначально был выпущен как кроссплатформенная среда выполнения с открытым исходным кодом для разработки серверных приложений с использованием JavaScript. Он использовал движок Google V8 для интерпретации JavaScript и добавил API для доступа к файловой системе, создания серверов и загрузки кода из внешних модулей [26]. Node поставляется в комплекте с менеджером пакетов под названием npm, который позволяет легко использовать более миллиона библиотек, доступных в его реестре.

Можно выделить следующие достоинствами этого фреймворка [27]:

1. Использование стандартных веб-технологий позволяет быстро создавать десктопные версии веб-приложения с тем же внешним видом и функционалом. Для этого не требуются навыки программирования для десктоп-платформ.
2. Единая база исходного кода – не требуется разработки под каждую поддерживаемую ОС отдельно, упрощение поддержки и обновления кода
3. API-интерфейсы Node.js позволяет иметь низкоуровневый доступ к аппаратным средствам и компонентам операционной системы, например доступ к файловой системе.
4. Поддержка многих функций, специфичных для каждой ОС: добавление меню в окно приложения, отправка уведомлений с помощью системы уведомлений, добавление элементов в список недавних элементов, и т.д.
5. Инструменты для сборки разработанного приложения и создания установщиков под различные операционные системы
6. Возможности автоматического обновления и сбора отчетов о сбоях

Несмотря на многочисленные достоинства у фреймворка имеется ряд проблем [28]:

1. Большой размер для установки и работы установленной программы –для корректной работы требуется упаковать целиком Chromium и Node.js
2. Потребление ресурсов в процессе работы – приложения на Electron требуют больше процессорного времени и количества оперативной памяти, чем нативные, разработанные с помощью ОС-специфичных инструментов
3. Недостаточно высокая безопасность по умолчанию – для повышения уровня защищенности как исходного кода, так и защиты от атак в процессе выполнения требуется применять много дополнительных мер и использовать особую архитектуру приложения
4. Необходимость учитывать особенности операционных систем в некоторых частях логики программы, а также самостоятельно добавлять бинарные зависимости для некоторых платформ.

Несмотря на имеющиеся недостатки можно сделать вывод, что этот фреймворк полностью подходит для разработки кроссплатформенного прототипа программного изделия, в котором не требуется расширенная интеграция с ОС и первоклассная производительность.

# **TypeScript**

В качестве языка программирования принято решение использовать TypeScript, основанный на основном языка веба — JavaScript.

TypeScript — это язык программирования, разработанный и поддерживаемый Microsoft. Он является строгим синтаксическим надмножеством JavaScript и добавляет в язык необязательную статическую типизацию. TypeScript предназначен для разработки больших приложений и транскомпилируется в JavaScript. Поскольку TypeScript является надмножеством JavaScript, существующие программы JavaScript также являются допустимыми программами TypeScript [29].

Достоинствами TypeScript являются:

1. Строгая типизация - уменьшение количество потенциальных ошибок, которые могли бы возникнуть при разработке на JavaScript.
2. Реализация многих концепции из объектно-ориентированных языков — наследование, полиморфизм, инкапсуляция и модификаторы доступа и т.д.
3. Упрощение написания больших сложных программ – более легко поддерживать, развивать, масштабировать и тестировать приложения, чем на стандартном JavaScript.
4. Открытый доступный исходный код и кроссплатформенная работа, поддержка большинства современных браузеров и стандартов
5. Полная совместимость с JavaScript – любые специфичные конструкции языка в конечном счете преобразуются в JS.

# **База данных**

**Анализ существующих кроссплатформенных БД**

Одним из основных средств организованного хранения данных в информационных системах является база данных (БД). Доступ пользователя и приложений к самой базе данных, как правило происходит через систему управления базами данных (СУБД).

Выбор типа базы данных был произведен на основе сравнения, представленного в [30]. Учитывая, что проектируемый модуль является десктопным приложением без доступа в интернет и предназначен для хранения данных локальных пользователей машины, решено выбрать реляционную СУБД.

Для выбора конкретной СУБД рассмотрим 3 наиболее популярных реляционных СУБД: MySQL, PostgreSQL и SQLite.

MySQL – это одна из ведущих СУБД с открытым исходным кодом, доступная на различных платформах. Она была разработан Oracle Corporation и написан на C и C ++ с огромным объемом поддержки и документации. MySQL используется крупными техническими гигантами, такими как Facebook, Google, Twitter, и т.д. MySQL использует архитектуру сервер / клиент, состоящую из многопоточного SQL-сервера. Это позволяет повысить производительность, поскольку потоки ядра могут легко использовать несколько процессоров [31].

Достоинствами MySQL являются:

1. Простота использования
2. Высокая производительность для больших баз данных
3. Хорошая масштабируемость и несколько механизмов хранения данных
4. Кроссплатформенная работа
5. Высокая безопасность

Недостатками MySQL являются снижение производительности при массовых операциях вставки. Также СУБД имеет проблемы с длительными операциями выборки и лучше всего подходит для небольших SELECT [32].

MySQL наиболее подходит для веб-сайтов с большим объемом данных, пакетного программного обеспечения и бизнес-систем.

PostgreSQL — это мощная, продвинутая объектно-реляционная СУБД с открытым исходным кодом, обладающая высокой надежностью, функциональностью и производительностью. Она написана на C и была разработана PostgreSQL Global Development Group и работает в большинстве современных ОС. Многие технологические гиганты, такие как Apple, Cisco, Uber, используют эту СУБД [31].

Достоинствами PostgreSQL являются:

1. Высокая масштабируемость благодаря сложному механизму блокировки.
2. Большой набор пользовательских интерфейсов и предопределенных функций.
3. Поддержка JSON, который используется для связи с другими хранилищами данных
4. Зрелая функционал на стороне сервера
5. Поддержка управлением параллелизмом на нескольких уровнях

Недостатками MySQL являются снижение производительности при массовых операциях вставки. Также СУБД имеет проблемы с длительными операциями выборки и лучше всего подходит для небольших SELECT.

По сравнению с MySQL PostgreSQL более энергоемкий. Поскольку PostgreSQL разветвляет процесс для новых клиентских подключений, это может занимать много памяти при одновременном подключении клиентов по сравнению с моделью MySQL «поток на подключение». Также недостаток можно увидеть во время частых обновлений данных из-за отсутствия поддержки кластеризованных индексов [32].

PostgreSQL наиболее полезна в банковской сфере и больших бизнес-проектов.

SQLite — это библиотека на языке C, которая реализует легкий, высоконадежный и полнофункциональный механизм реляционной СУБД. Это кроссплатформенный, транзакционный движок базы данных SQL с нулевой конфигурацией, который является одной из наиболее широко распространенных БД в мире и может бесплатно использоваться для любых целей, коммерческих или частных. В отличие от большинства других баз данных SQLite не имеет отдельного серверного процесса, то есть является встраиваемой СУБД [31].

К преимуществам SQLite можно отнести:

1. Легкость и простота в использовании,
2. Отсутствие требований установки или настройки,
3. Отсутствие необходимости в выделенных серверах для хранения базы данных, все сохраняется на жестком диске,
4. Очень портативный благодаря файловой системе,
5. Высокая совместимость практически со всеми языками программирования и платформами

Одним из основных недостатков системы SQLite является отсутствие многопользовательских возможностей, которые можно найти в MySQL и PostgreSQL. Это приводит к отсутствию детального контроля доступа, системы управления пользователями и возможностей безопасности, помимо шифрования самого файла базы данных. Еще одним большим недостатком SQLite является обработка сериализуемых операций записи. Это может быть серьезным узким местом для приложений, требующих параллелизма. Поскольку SQLite — это файловая СУБД, это может вызвать проблемы с производительностью с большими наборами данных из-за ограничений файловой системы. В этих случаях было бы целесообразно выбрать клиент-серверные базы данных [32].

SQLite лучше всего подходит для тестирования и разработки, для которых не требуется большое количество функций.

Итоговое сравнение этих 3 СУБД приведено в таблице 27.

Таблица 27 – Сравнение СУБД SQLite, MySQL и PostgreSQL

|  |  |  |  |
| --- | --- | --- | --- |
| **СУБД** | **SQLite** | **MySQL** | **PostgreSQL** |
| Архитектура | Файловая (встроенная) | Клиент / Сервер | Клиент / Сервер |
| ОС сервера | Нет (без сервера) | FreeBSD, Linux, OS X, Solaris, Windows | FreeBSD, HP-UX, Linux, NetBSD, OpenBSD, OS X, Solaris, Unix, Windows |
| Согласованность транзакций | ACID | ACID | ACID |
| Язык программирования (базовый код) | C | C, C ++ | C |
| Поддерживаемые языки программирования | ActionScript, Ada, Basic, C, C #, C ++, D, Delphi, Forth, Fortran, Haskell, Java, JavaScript, Lisp, Lua, MATLAB, PHP, PL / SQL | C, C ++, Delphi, Perl, Java, Lua, .NET, Node.js | NET, C, C ++, Delphi, Java, Perl, PHP, Python, Tcl |
| Популярные варианты использования | Сайты с низким и средним трафиком, встроенные устройства, тестирование и разработка | Веб-сайты, веб-приложения, стек LAMP, приложения на основе OLTP | Аналитика, интеллектуальный анализ данных, хранилище данных, бизнес-аналитика, Hadoop |

**Выбор БД для кроссплатформенного приложения**

Исходя из результатов сравнения и особенностей проектируемой системы решено использовать SQLite – она наиболее подходящая СУБД для разработки кроссплатформенного десктопного приложения-прототипа, обеспечивающего хранение метаданных о локальной файловой системе. SQLite не требует отдельной установки и сервера, и существует большое количество инструментов для работы с этой СУБД на используемом в модуле языке программирования – TypeScript.

# **Высокоуровневое средство работы с БД**

**Анализ высокоуровневых средств работы с БД**

Работа напрямую с базами данных из приложения вызывает трудности, так как существуют различия в способах представления структур данных и способах выражения отношений между различными объектами. Для решения этой проблемы было создано множество различных инструментов, которые служат в качестве интерфейса между основным приложением и уровнем данных.

Для выбора средства работы с БД рассмотрим тремя распространенных подхода: стандартный SQL, построители запросов и ORM (объектно-реляционные преобразователи).

Некоторые приложения напрямую взаимодействуют с базой данных, записывая и выполняя запросы на родном языке, поддерживаемом ядром базы данных. Часто драйвер базы данных — единственное, что нужно для подключения, аутентификации и связи с экземпляром базы данных. Для многих реляционных баз данных предпочтительным языком запросов является SQL [33].

Достоинства управление данными с помощью SQL:

1. Отсутствие абстракции - явные запросы к базе данных и предсказуемые результаты их выполнения
2. Производительность написанных в «сыром» виде запросов - сложные уровни абстракции могут сгенерировать неэффективные и длинные SQL-запросы, которые повышают время отладки.
3. Гибкость в построении запросов - более высокие уровни абстракции пытаются преодолеть разрыв между двумя разными парадигмами, что может ограничивать типы операций, которые они могут выражать.

Недостатками такого подхода являются:

1. Необходимость хорошо разбираться в структуре данных БД, переводе между типами данных и структурами, используемыми в приложении
2. Необходимость контроля безопасности ввода данных – пользователи могут осуществить SQL-инъекцию (ввести SQL запрос вместо ожидаемых данных), что может повлиять на состояние базы данных.
3. Необходимость работы со строками для построения запросов - операции с базой данных могут быть охвачены множеством уровней манипуляций со строками, что может привести к случайному искажению данных.

Построитель SQL-запросов добавляет уровень абстракции над языками запросов для баз данных путем формализации шаблонов запросов и предоставления методов или функций, что позволяет обезопасить ввод и упростить интеграцию с приложением [33]. При этом структуры и действия, поддерживаемые уровнем базы данных, по-прежнему хорошо узнаваемы при использовании подобных построителей.

Преимущества построителей SQL-запросов:

1. Упрощение работы с БД - построители запросов используют те же конструкции, что и приложение, легко разбить запросы на логические блоки, которые обрабатывают определенные части запроса.
2. Прозрачность преобразований запросов в SQL – методы, предоставляемые построителями запросов понятно для разработчика, отображаются в итоговый SQL запрос из-за небольшого уровня абстракции.
3. Поддержка нескольких баз данных - построители SQL-запросов абстрагируют некоторые различия в различных реляционных базах данных и позволяют изменить используемую БД в будущем.

Недостатки построителей SQL-запросов:

1. Требуется хорошо разбираться в SQL в дополнение к специфическому синтаксису и возможностям самого построителя запросов.
2. Отсутствие автоматической синхронизации между объектами, используемыми в приложении и объектами в базе данных.
3. Не все запросы реализуемы с использованием предоставленных построителем методов, поэтому иногда приходится обращаться к БД напрямую.

Объектно-реляционные преобразователи (ORM) — это программное обеспечение, предназначенные для преобразования между представлениями данных в реляционных базах данных и представлением в памяти, используемым в объектно-ориентированном программировании (ООП) [34]. ORM предоставляет объектно-ориентированный интерфейс для данных в базе данных, пытаясь использовать знакомые концепции программирования и уменьшить количество шаблонного кода, необходимого для ускорения разработки.

Существуют весомые преимущества ORM:

1. Помощь в абстрагировании модели данных на уровне БД до контекста приложения, что позволяет быстрее работать над основной бизнес-логикой вместо того, чтобы разбираться с хранилищем данных.
2. Удаление шаблонной логики, необходимой для взаимодействия с базами данных - ORM часто позволяет управлять изменениями схемы базы данных на основе изменений в коде приложения, поэтому не обязательно заранее проектировать оптимальную схему базы данных, а использовать решение, предоставляемое ORM.

Недостатками ORM являются:

1. Попытка скрыть детали базы данных, что может приводит к проблемам по мере роста сложности приложения.
2. Абстракция никогда не может быть полной, поэтому требуется понимания основного языка запросов или структуры базы данных.
3. Потеря объектно-реляционного соответствия - трудности перевода между объектно-ориентированным программированием и реляционной парадигмой, используемой в реляционных базах данных [35], что с каждым увеличением сложности приводит к дополнительным абстракциям.
4. ORM обычно медленнее, чем альтернативы из-за генерации сложных запросов для относительно простых операций с базой данных.

Итоговое сравнение трех рассмотренных высокоуровневых средства работы с БД по их достоинствам и недостаткам представлено в таблице 28:

Таблица 28 – Сравнение SQL, построителя запросов и ORM

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Подход** | **Упор на работу с БД / программирование** | **Сложность разработки** | **Уровень абстракции** | **Сложность API** |
| Стандартный SQL | БД | высокая | нет | низкий |
| Построитель запросов | смешанный | низкая | низкий | низкие |
| ORM | программирование | низкая | высокий | высокий |

Исходя из сравнения высокоуровневых средств принято решение использовать ORM – она предоставляет наибольший уровень абстракции и интеграции с кодом приложения. Также использование ORM сокращает время разработки схемы базы данных и внесения изменений в модель данных. Для разрабатываемого модуля полагается, что сгенерированные неоптимальные SQL запросы не будут являться узким местом в работе прототипа.

**Выбор наиболее удобного средства работы с БД**

Выбор ORM на языке TypeScript напрямую зависит от уровня привязки типов TypeScript для API и безопасности типов, который фактически обеспечивается. Можно выделить три основных критерия [36], по которым стоит осуществлять этот выбор:

1. Источник определения типов: Встроены ли определения типов в ORM или взяты из публичного репозитория типов DefinitelyTyped
2. Создание записи: являются ли модели типобезопасными и можно ли создавать записи типобезопасным способом
3. Выборка записей: являются ли объекты типобезопасными при извлечении данных даже для неполных моделей и отношений

В результате анализа существующих ORM было решено использовать библиотеку Prisma, которая имеет положительные показатели по всем трем критериям.

Prisma — это более новая ORM, ориентированная на задание моделей через схему, в отличие от типичных ORM, которые используют классы для определения моделей. В схеме Prisma определяется свой источник данных (БД) модели, которые отражают сущности, и отношения между ними.

Используя эту схему, Prisma генерирует типобезопасный клиент, который предоставляет API-интерфейс CRUD) который затем используется для запроса своей базы данных. Этот клиент Prisma работает как продвинутый конструктор запросов, который можно использовать в приложении Node.js для возврата простых объектов JavaScript, а не экземпляров класса модели [36].

Рассмотрим, каким образом, Prisma удовлетворяет критериям, описанным выше.

Определения типов клиента Prisma автоматически генерируются при создании клиента (Prisma Client). Модели, определенные в схеме Prisma (например, Vertex и Edge), автоматически экспортируются как типы в сгенерированный файл index.d.ts, что обеспечивает полную безопасность типов при запросе данных.

При создании новой записи с помощью Prisma попытка добавить свойства, не определенные в модели, приводит к ошибке типа. Свойства модели дополняются автоматически. Вложенные записи, вставляющие данные в несколько таблиц с помощью отношений, также безопасны по типу. Это означает, что при создании вершины и связанных ребер с использованием одного и того же вызова prisma.vertex.create() поля модели Edge также проверяются по типу и заполняются автоматически, что гарантирует, что вложенная запись также будет действительной.

При извлечении записей из базы данных возвращаемые объекты полностью типизированы даже для запросов с отношениями. Например, при выборке всех вершин из базы данных и включении отношения с вершинами для дополнительной выборки всех связанных с вершиной ребер итоговый тип становится (Vertex & {edges: Edge[];})[].

Чтобы дополнительно продемонстрировать уровень безопасности типов, рассмотрим частичный запрос, в котором запрашиваются только определенные свойства, изменяя тип возвращаемого объекта:

const vertexPartialEdges = await prisma.vertex.findMany({

include: {

edges: {

select: {

name: true,

},

},

},

});

В этом запросе возвращаются все вершины, но для модели отношения сообщений выбираются только названия ребер. Результат запроса vertexPartialEdges затем имеет следующий тип:

(Vertex & {

edges: {

name: string;

}[];

})[]

Это означает, что попытка получить доступ к другим полям ребра, которые не были выбраны, не удастся. Prisma - единственная ORM-подобная библиотека, которая может обеспечить такую степень детализации безопасности типов.

Таким образом, Prisma позволяет достичь самого высокого уровня безопасности типов среди ORM TypeScript для запросов вложенных отношений, а также частичных запросов, которые изменяют форму возвращаемых моделей.

# **Cytoscape.js**

Двухмерную визуализаци метаграфа было решено осуществить с помощью библиотеки Cytoscape.js. Cytoscape.js — это библиотека теории графов с открытым исходным кодом, написанная на JS, которая позволяет удобно отображать насыщенные интерактивные графы и управлять ими. Библиотека позволяет клиенту подключаться к пользовательским событиям и легко интегрируется в приложение [37]. Cytoscape.js включает в себя обработку множества жестов пользователя, включая масштабирование, выделение элементов, панорамирование и т. д. Стоит отметить, что данная библиотека одна из немногих, которая предоставляет возможность создания родительских вершин (compound node), что можно использовать для представления метаграфа с некоторыми ограничениями.

Библиотеку возможно автономно использовать как на стороне клиента в браузере, так и без визуализации на Node.js. Для TypeScript доступны типы из публичного репозитория DefinitelyTyped.

Достоинства данной библиотеки [37]:

1. Полнофункциональная библиотека графов с открытым исходным кодом, написанная на чистом JS
2. Используется в коммерческих проектах и проектах с открытым исходным кодом
3. Совместимость со всеми современными браузерами и системами модулей, полная поддержка JSON
4. Использует таблицы стилей для отделения презентации от данных
5. Поддерживает операции теории множеств и включает алгоритмы теории графов

# **Архитектура системы**

Архитектура проектируемого модуля опирается на архитектуре Electron-приложений, которую можно рассматривать как разновидность клиент-серверной архитектуры.

Electron-приложения состоят из двух типов процессов: основного (main) процесса и одного или нескольких рендеринг (renderer) процессов. Каждый процесс играет свою роль в приложении. Упрощенная архитектура Electron-приложения представлена на рисунке 11.

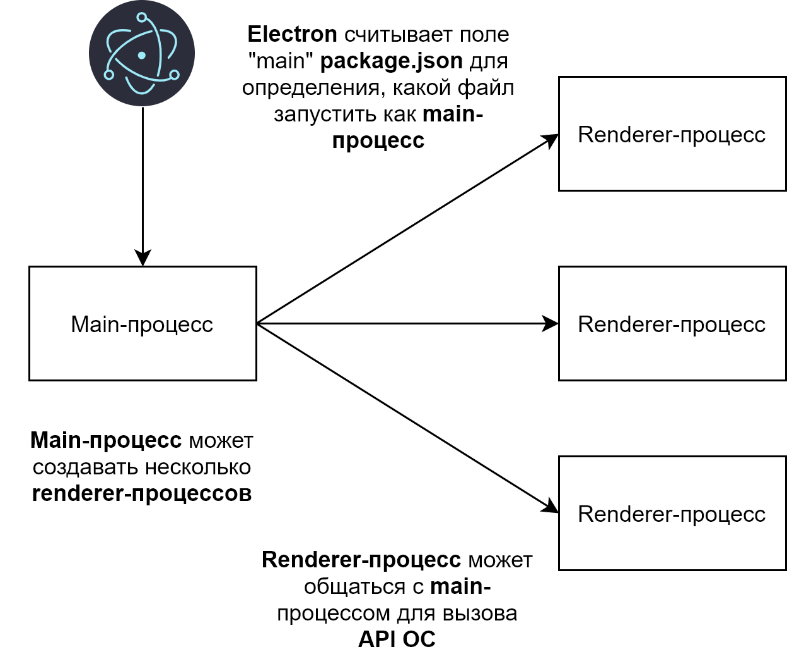


Рисунок – Упрощенная архитектура Electron-приложений

У основного процесса есть несколько важных обязанностей. Он может реагировать на события жизненного цикла приложения, такие как запуск, завершение, подготовка к выходу, переход в фоновый режим, переход на передний план и многое другое. Этот процесс также отвечает за связь с API-интерфейсами операционной системы. Когда Electron запускается, он обращается к полю запуска в манифесте package.json, чтобы определить точку входа приложения и запускает этот файл как основной процесс.

Основной процесс может создавать и уничтожать renderer-процессы с помощью модуля BrowserWindow, которые могут загружать веб-страницы для отображения графического интерфейса. Каждый процесс использует преимущества многопроцессорной архитектуры Chromium и работает в собственном потоке. Затем эти страницы могут загружаться в дополнительные файлы JavaScript и выполнять код в этом процессе.

Renderer-процессы изолированы друг от друга и не имеют доступа к API интеграции операционной системы. Electron включает возможность упрощения взаимодействия между процессами посредством модулей обеспечения межпроцессорного взаимодействия (IPC).

В проектируемом модуле renderer-процесс содержит логику взаимодействия с пользователем через веб-страницы, а также создает объект графа библиотеки Cytoscape.js и работает с ним. Main-процесс отвечает за работу с БД SQLite через ORM Prisma. С его помощью также осуществляются обращение к файловой системе и вывод диалоговых экранов для выбора файлов. Общение между renderer-процессами и main-процессом основано на модулях ipcRenderer и ipcMain соответственно. Запросы к БД и файловой системе и их результаты передаются через именованные каналы, которые будут рассмотрены далее. В общем виде архитектура системы приведена на рисунке А.4.

# **Диаграмма последовательности взаимодействия с системой**

Диаграмма последовательности UML-диаграмма, на которой для набора объектов на единой временной оси показан жизненный цикл объекта (сущности) и взаимодействие актеров (действующих лиц) информационной системы в рамках прецедента [38].

Основными элементами диаграммы последовательности являются обозначения объектов (прямоугольники с названиями объектов), вертикальные «линии жизни», отображающие течение времени, прямоугольники, отражающие деятельность объекта или исполнение им определенной функции (прямоугольники на пунктирной «линии жизни»), и стрелки, показывающие обмен сигналами или сообщениями между объектами.

Диаграмма используется для:

1. Моделирования логики сложной процедуры, функции или операции.
2. Отображение взаимодействия объектов и компонентов друг с другом для завершения процесса.
3. Исследование функциональных особенностей существующего или будущего сценария.

С помощью данной диаграммы, приведенной на рисунке А.5 продемонстрировано, как происходит межпроцессорное взаимодействие в системе, инициируемое действиями пользователя.

На диаграмме изображена следующая последовательность:

1. Main-процесс порождает renderer-процесс, содержащий окно браузера для отображения проекта с рабочим метаграфом
2. Браузерное окно сообщает main-процессу о том, что оно отрисовано и готово к показу пользователю
3. Main-процесс инициирует показ окна пользователю
4. Пользователь видит браузерное окно с содержимым проекта
5. Пользователь изменяет совершает действия с метаграфом посредством интерфейса
6. Renderer-процесс получает данные с помощью API Cytoscape.js и отправляет их в требуемом формате main-процессу с помощью модуля ipcRenderer
7. Main-процесс слушает канал, по которому ipcRenderer отправил сообщение, с помощью модуля ipcMain и выполняет требуемый запрос к БД с помощью сгенерированных методов Prisma Client
8. Prisma Client возвращает данные моделей в виде JavaScript объектов
9. Main-процесс возвращает данные от Prisma Client
10. Renderer-процесс обновляет метаграф с API Cytoscape.js и выводит требуемые результаты запросов
11. Пользователь инициирует закрытие проекта
12. Браузерное окно сообщает main-процессу о том, что оно собирается закрыться
13. Main-процесс запрашивает у окна несохраненные данные
14. Браузерное окно передает несохраненные данные с помощью ipcRenderer main-процессу
15. Main-процесс сохраняет данные в БД с помощью методов Prisma Client
16. Prisma Client возвращает сохраненные данные main-процессу
17. Main-процесс закрывает браузерное окно с проектом

# **Проектирование моделей хранения данных**

Как было сказано выше, в системе было решено использовать реляционную СУБД SQLite и ORM Prisma. В связи с этим была разработана инфологическая и даталогическая модели. Помимо этого, своя модель данных также имеется на уровне модуля визуализации Cytoscape.js. Таким образом, следует рассмотреть 5 моделей:

1. Метаграфовая
2. Инфологическая
3. Даталогическая
4. ORM-модель
5. Модель визуализации

Все эти модели описываются в следующих разделах.

# **Метаграфовая модель**

Основным вопросом при создании моделей данных стало представление метаграфа, так как она служит семантической моделью в разработанной системе. Было решено использовать упрощенную модель на основе двух сущностей, опирающуюся на описание из работы [39]. Метаграф представляется двумя сущностями – Вершина и Ребро и двумя связями между ними, обозначающими начальную и конечную вершиной. Для отождествления вложенности вершины в метавершину вводится специальный тип ребра и соответствующий атрибут-дискриминатор у сущности Ребро. Таким образом, модель дает возможность представить метавершины, вершины и обычные ребра. Метаребра, описанные в исследовательской части, не поддерживаются, также невозможно включение ребра в метавершину – полагается, что ребро находится в метавершине, если обе вершины состоят в метавершине.

Дополнительное ограничение связано с особенностями используемой библиотеки визуализации. Вершина может иметь только одну родительскую вершину [37], то есть вершина может быть вложена напрямую в единственную метавершину, таким образом множество метаграфов, которое могут описать используемые модели еще несколько сужается.

# **Инфологическая модель**

На этапе анализа предметной области были выделены основные объекты предметной области. Следующим шагом является решение вопроса, какая информация об объектах должна храниться в модели данных и как ее представить с помощью данных. Для описания инфологической схемы используются диаграммы «сущность – связь» (ER-диаграммы).

Данная модель дает семантическое описание предметной области, позволяет описать объекты системы и отношения между ними [40]. При разработке модели используются три конструктивных элемента системы: сущность, связь, атрибут.

Сущность – различимый объект, который может быть идентифицирован свойствами, отличающими его от другого объекта.

Связь – отношения, установленные между сущностями.

Атрибут - характеристика сущности. Атрибуты определяют, какая информация о сущности должна храниться в базе данных.

Составленная на основе сущностей, связей и атрибутов предметной области ER-диаграмма модели хранения данных представлена на рисунке А.6.

# **Даталогическая модель**

Построение даталогической модели основано на ранее разработанной инфологической модели. Даталогическая модель – создание схемы базы данных на основе конкретной модели данных, в частности, на реляционной.

В такой модели сущности соответствуют таблицами, атрибуты — полям таблиц. Определенные связи соответствуют аналогичным связям в инфологической модели. С помощью этой диаграммы можно представить информационную структуру базы данных в графическом виде в более наглядном виде. Даталогическая модель представлена на рисунке А.7.

Описание таблиц базы данных представлено в таблице 29, а описание связей представлено в таблице 30.

Таблица 29 – Описание таблиц базы данных

| Сущность | Атрибут | Тип данных | Краткое описание |
| --- | --- | --- | --- |
| Каталог | id | INTEGER | ID каталога |
| name | TEXT | Имя каталога |
| fullPath | TEXT | Путь до каталога без имени |
| valid | BOOLEAN | Флаг действительности каталога |
| Файл | id | INTEGER | ID файла |
| name | TEXT | Имя файла |
| fullPath | TEXT | Путь до файла без имени |
| valid | BOOLEAN | Флаг действительности файла |
| dirId | INTEGER | ID каталога, в котором находится файл |
| Ребро | id | TEXT | ID ребра |
| startID | TEXT | ID начальной вершины |
| endID | TEXT | ID конечной вершины |
| name | TEXT | Название ребра |
| inMeta | BOOLEAN | Флаг ребра вложенности |
| workspaceID | TEXT | ID связанного проекта |
| Вершина | id | TEXT | ID вершины |
| name | TEXT | Название вершины |
| meta | BOOLEAN | Индикатор метавершины |
| x | REAL | Координата x на визуализации |
| y | REAL | Координата y на визуализации |
| workspaceID | TEXT | ID связанного проекта |
| Проект | id | TEXT | ID проекта |
| name | TEXT | Название проекта |
| createdAt | DATETIME | Дата создания проекта |
| isTreeMode | BOOLEAN | Флаг режима проекта |
| Каталог\_Ребро | edgesId | TEXT | ID ребра |
| directoriesID | INTEGER | ID каталога |
| Каталог\_Вершина | verticesId | TEXT | ID вершины |
| directoriesID | INTEGER | ID каталога |
| Файл\_Ребро | edgesId | TEXT | ID ребра |
| filesID | INTEGER | ID файла |
| Файл \_ Вершина | edgesId | TEXT | ID вершины |
| filesID | INTEGER | ID файла |

Таблица 30 – Описание связей сущностей

|  |  |  |  |
| --- | --- | --- | --- |
| Связь | От какой сущности | К какой сущности | Тип связи |
| Ребро состоит в | Проект | Ребро | 1-М |
| Вершина состоит в | Проект | Вершина | 1-М |
| Является начальной | Вершина | Ребро | 1-М |
| Является конечной | Вершина | Ребро | 1-М |
| Связан с ребром | Каталог | Ребро | М-М |
| Связан с вершиной | Каталог | Вершина | М-М |
| Прикреплен к ребру | Файл | Ребро | М-М |
| Прикреплен к вершине | Файл | Вершина | М-М |
| Включает | Каталог | Файл | 0,1-М |

# **Модели ORM и визуализации**

Модель данных, используемая в программе основана на типах TypeScript, сгенерированных ORM Prisma по даталогической модели БД. Каждая таблица (сущность) представляется отдельным типом. При этом при запросах, затрагивающих несколько связанных сущностей, возвращаемый объект включает только запрашиваемые поля, а связанным сущностям соответствуют массивы, что было продемонстрировано в описании ORM Prisma выше. Пример сгенерированных типов моделей для вершины и ребра представлен на рисунке 12.

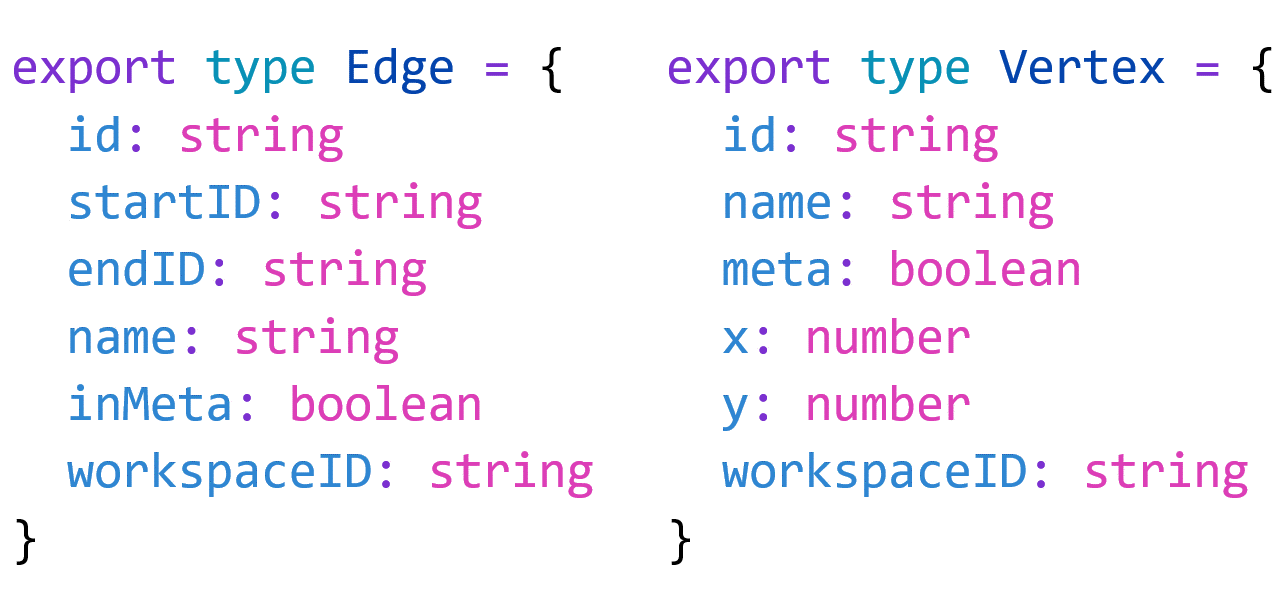


Рисунок – Типы Вершина и Ребро, сгенерированные Prisma

Модель данных визуализации основана на структуре данных библиотеки Cytoscape.js. Структура графа основана на формате JSON и блок, определяющий топологию графа, задается следующим образом (рисунок 13).

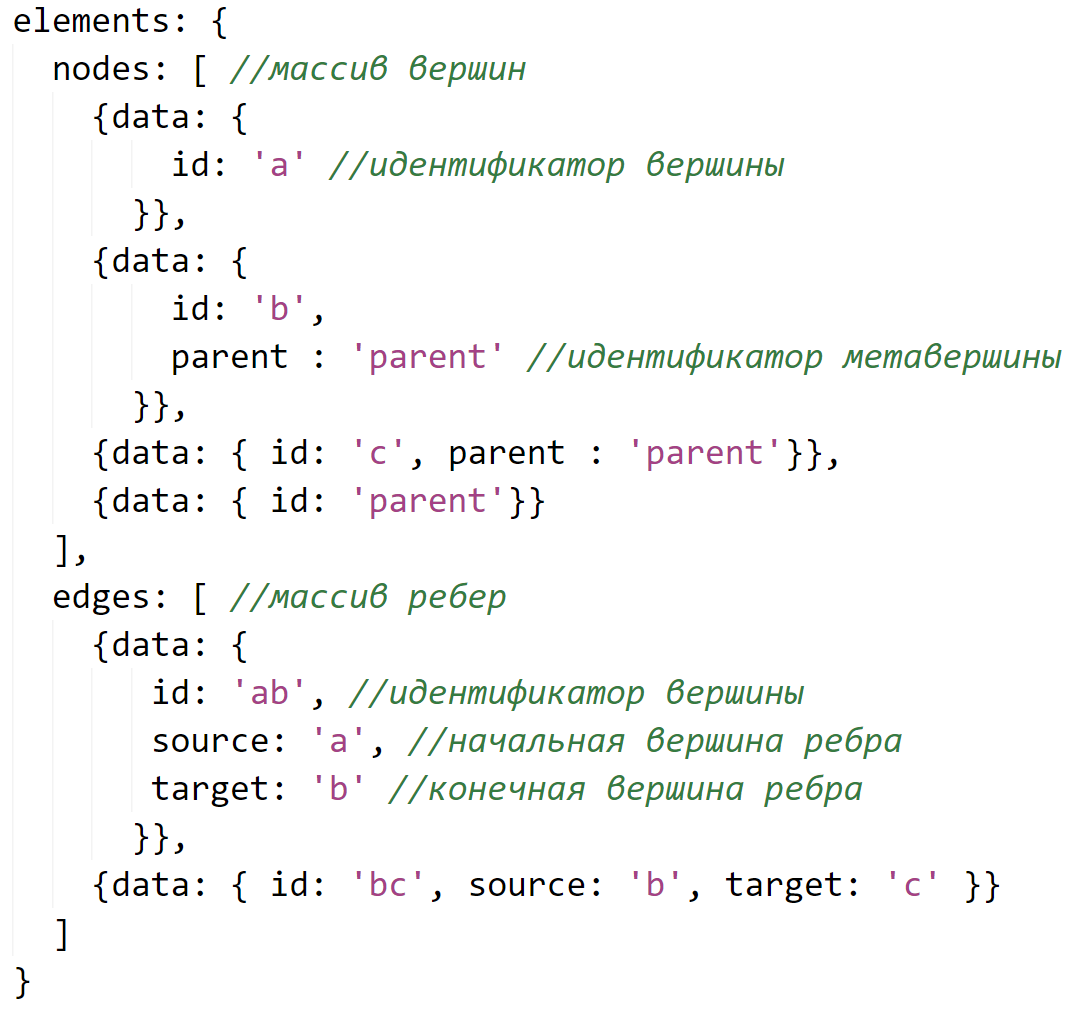


Рисунок – Описание структуры графа в Cytoscape.js

Граф, соответствующий представленной выше структуре приведен на рисунке 14.

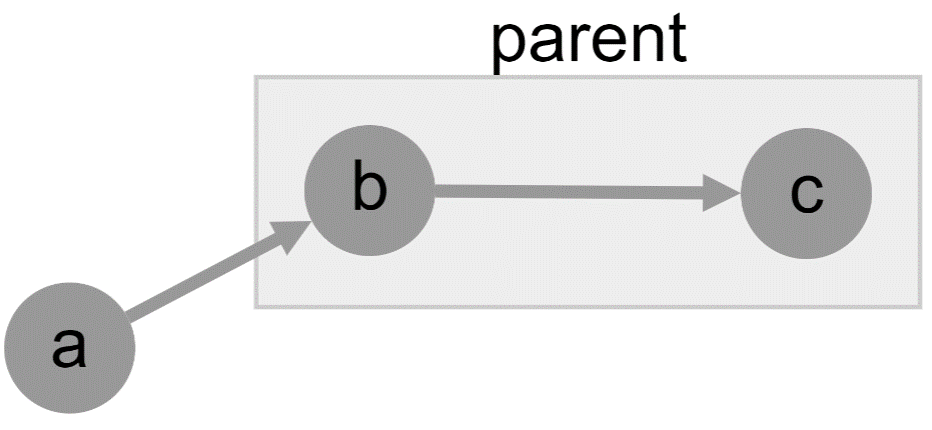


Рисунок – Визуализация графа, описанного на рис. 13

# **Интерфейс взаимодействия с пользователем**

Интерфейс взаимодействия с пользователем разбито на два основных экрана: экран выбора и создания проектов и основной экран с текущим проектом. Для ускорения разработки интерфейса было принято решение использовать фреймворк Bootstrap, которая дает доступ к большому числу шаблонов, стилей и иконок для разработки веб-приложений.

На экране со списком проектов представлен следующий функционал: добавление и удаление проектов, открытие проектов (рис. 15).

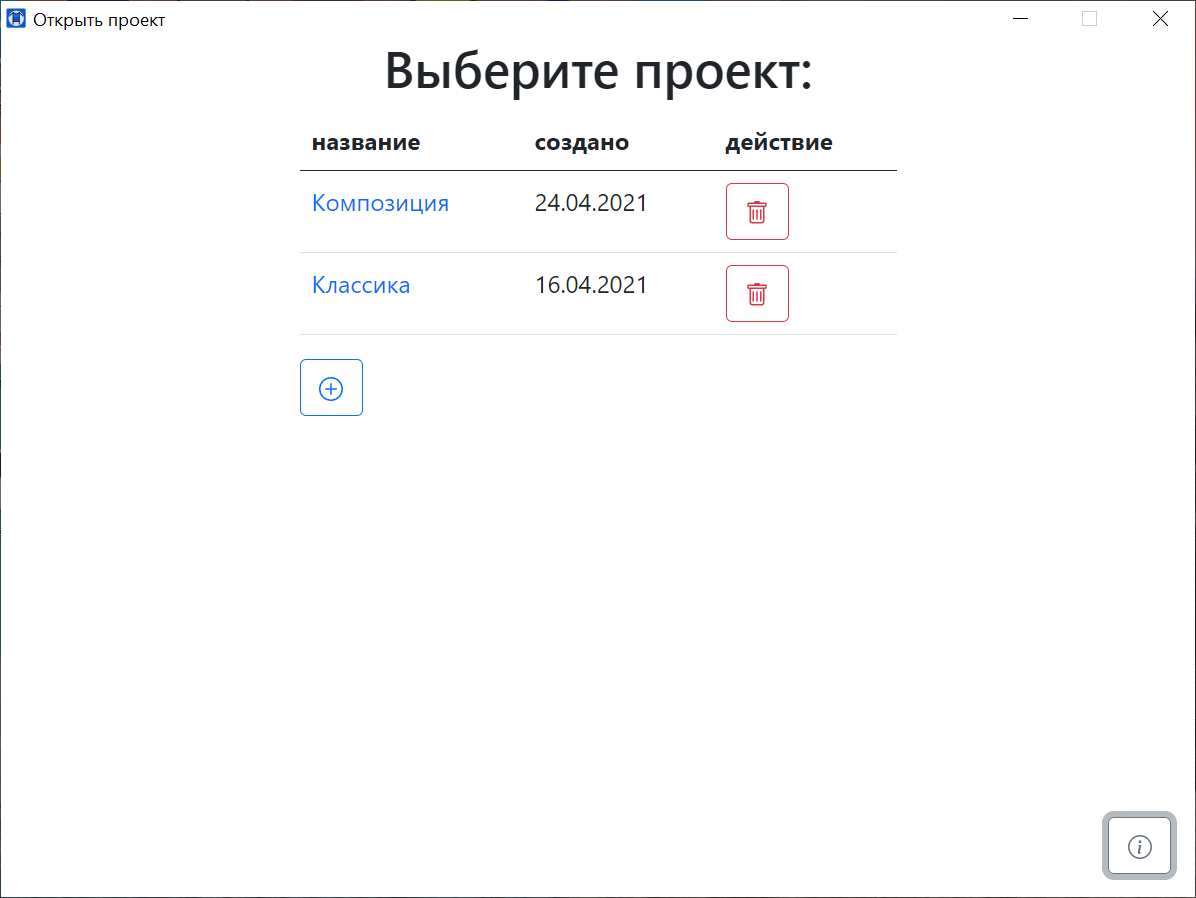


Рисунок – Начальный экран со списком проектов

На этом же экране имеется возможность просмотра информации об авторах проекта (рис. 16).

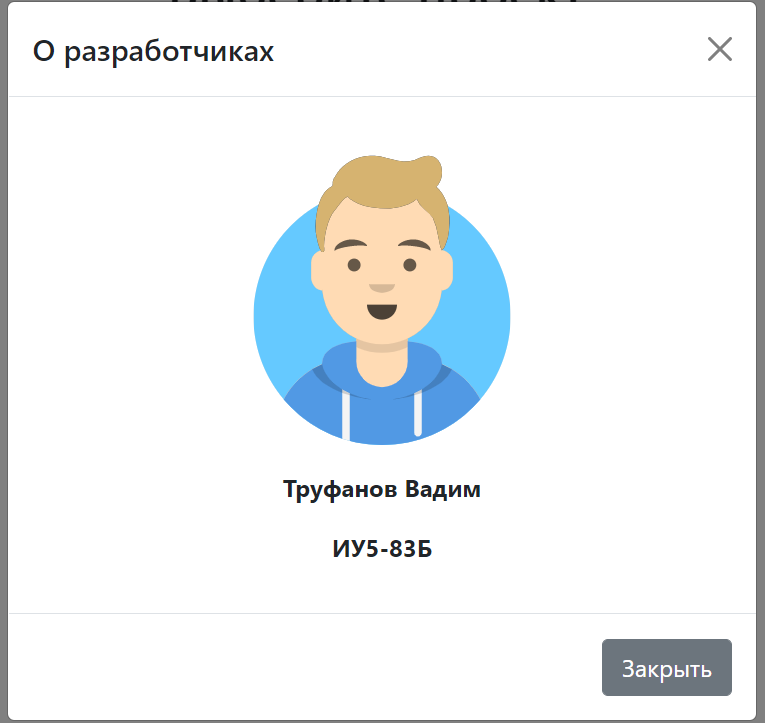


Рисунок – Экран об авторах проекта

При открытии проекта открывается новое окно, содержащее основной функционал приложения. В верхнем поле ввода возможно изменение названия проекта. Ниже поля ввода расположена панель инструментов для работы с визуализацией метаграфа (рис. 17).



Рисунок – Основной экран приложения

Раздел панели «Связи» содержит 3 функции:

1. Добавление связи
2. Добавление множества связей с общей начальной категорией
3. Удаление связи.

Раздел панели «Категории» содержит 6 функций:

1. Добавление категории
2. Объединение существующих категорий в новую мета-категорию
3. Включение категорий в существующую мета-категорию
4. Удаление категорий
5. Удаление мета-категории без вложенных в нее элементов
6. Дублирование элементов построенного метаграфа

Раздел панели «Инструменты» содержит 2 функции:

1. вписать все элементы метаграфа в размер рабочей области
2. открыть панель поиска

При выборе определенной функции выводятся подсказки, какие элементы следует выбрать для осуществления той или иной операции (рис. 18).

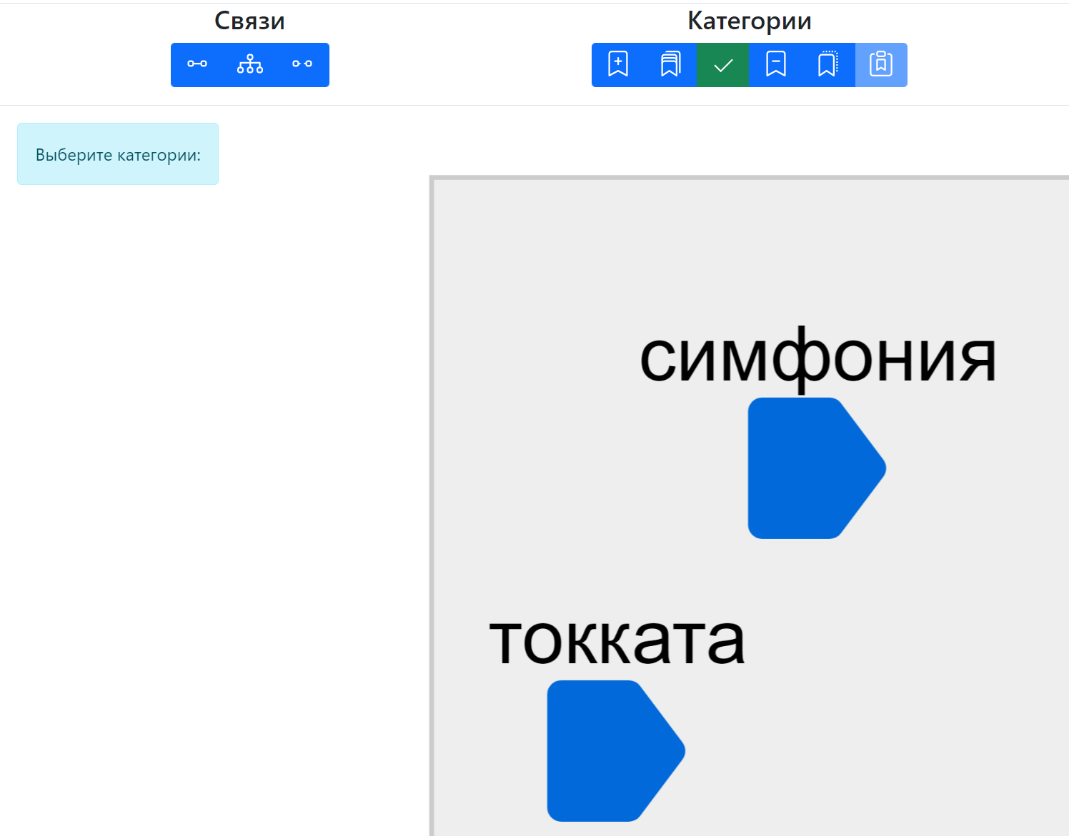


Рисунок – Отображение подсказок при работе с функциями

Правее панели инструментов расположен переключатель режима проекта: иерархия и процесс. В режиме иерархии связи становятся ненаправленными и у них скрываются имена. Становится недоступна функция дублирования элементов и активируется функция добавления множества связей. В режиме процессов связи становятся направленными и у них показываются имена. Становится недоступна функция добавления множества связей и активируется функция дублирования элементов. Оба режима представлены на рисунках А.8-А.9.

При выделении какого-либо элемента визуализации открывается боковая панель для возможности прикрепления документов к элементу или их удаления, а также изменения текущего имени элемента, выступающим в роли тега.

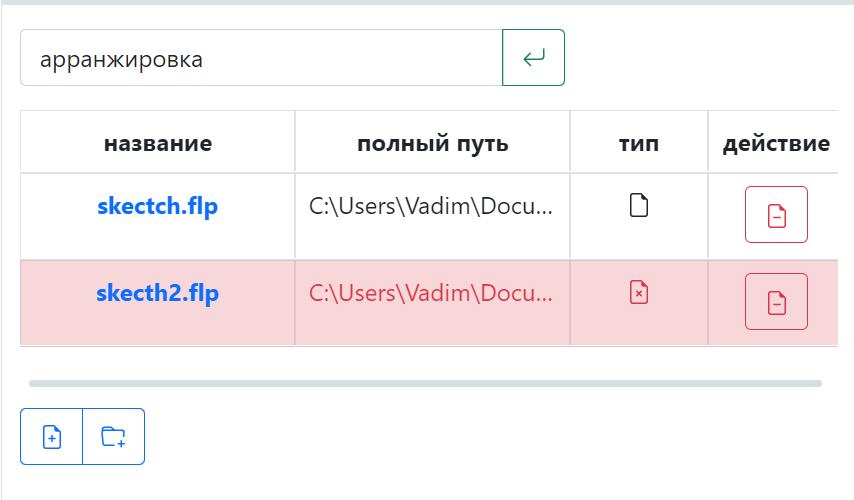


Рисунок – Панель с информацией об элементе метаграфа

В таблице выводится список документов, прикрепленных напрямую к элементу, представлена возможность открытия документов программой по умолчанию, удаления документов по отдельности и прикрепление папки или файла (рис. 20).

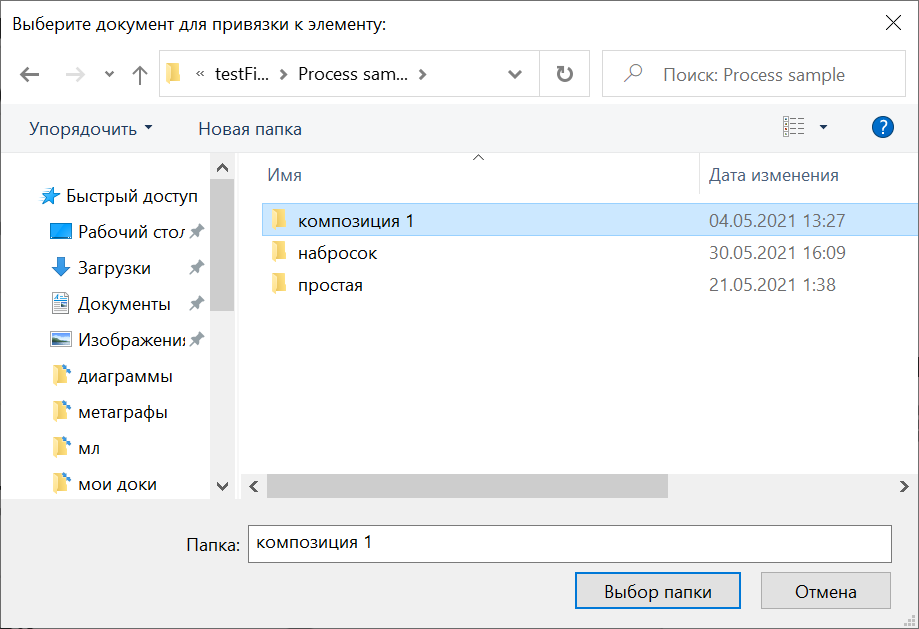


Рисунок – Диалоговое окно привязки папки

Элементы в таблице, выделенный красным цветом и особыми иконками, указывает на то, что документ не удается открыть. При нажатии на такой элемент предлагается выбрать его местоположение на диске заново (рис. 21).

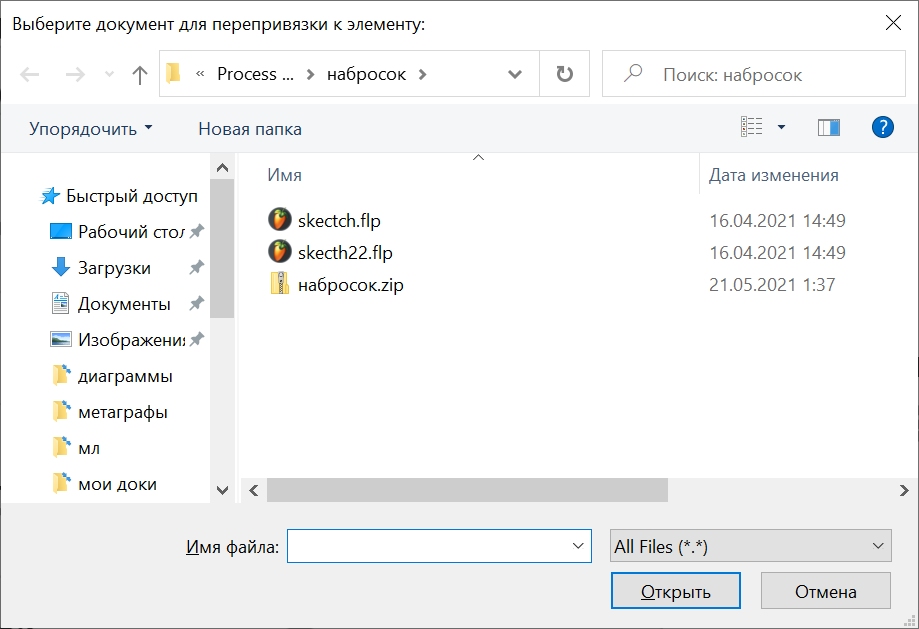


Рисунок – Диалоговое окно перепривязки файла

Поиск по проекту поддерживает два режима: поиск по документам и по элементам.

Поиск по документам основан на именах документов, при вводе специального символа «\*» выводится список всех документов проекта. В результате поиска выводится список документов с возможностью раскрытия элементов для открытия документа программой по умолчанию и просмотра прикрепленных к нему категорий. Список категорий включает все категории, непосредственно прикрепленные к документу, а также все мета-категории, в которые вложены непосредственно прикрепленные теги (рис. 22). При этом некоторые элементы списка отмечаются красным цветом и соответствующим набором иконок для обозначения того, что документ не может быть открыт (недействительный путь, нет прав доступа).

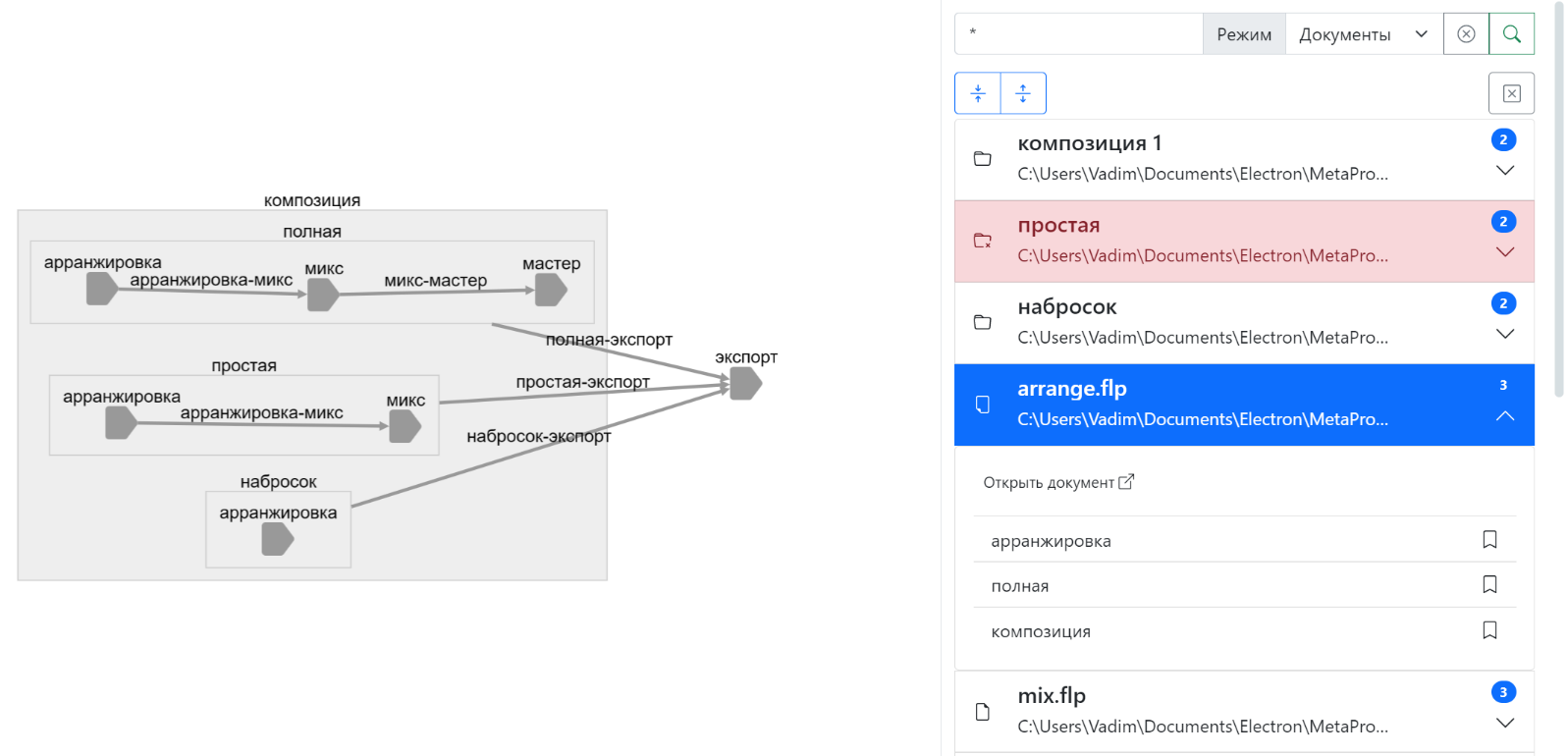


Рисунок – Поиск документов в метаграфе

Поиск по элементам работает аналогичным образом, за исключением того, что развернутый список включает множество документов, прикрепленных к элементу напрямую или к элементам, вложенным в мета-категорию. При этом документы также выделяются красным цветом если их не удается открыть (рис. 23).

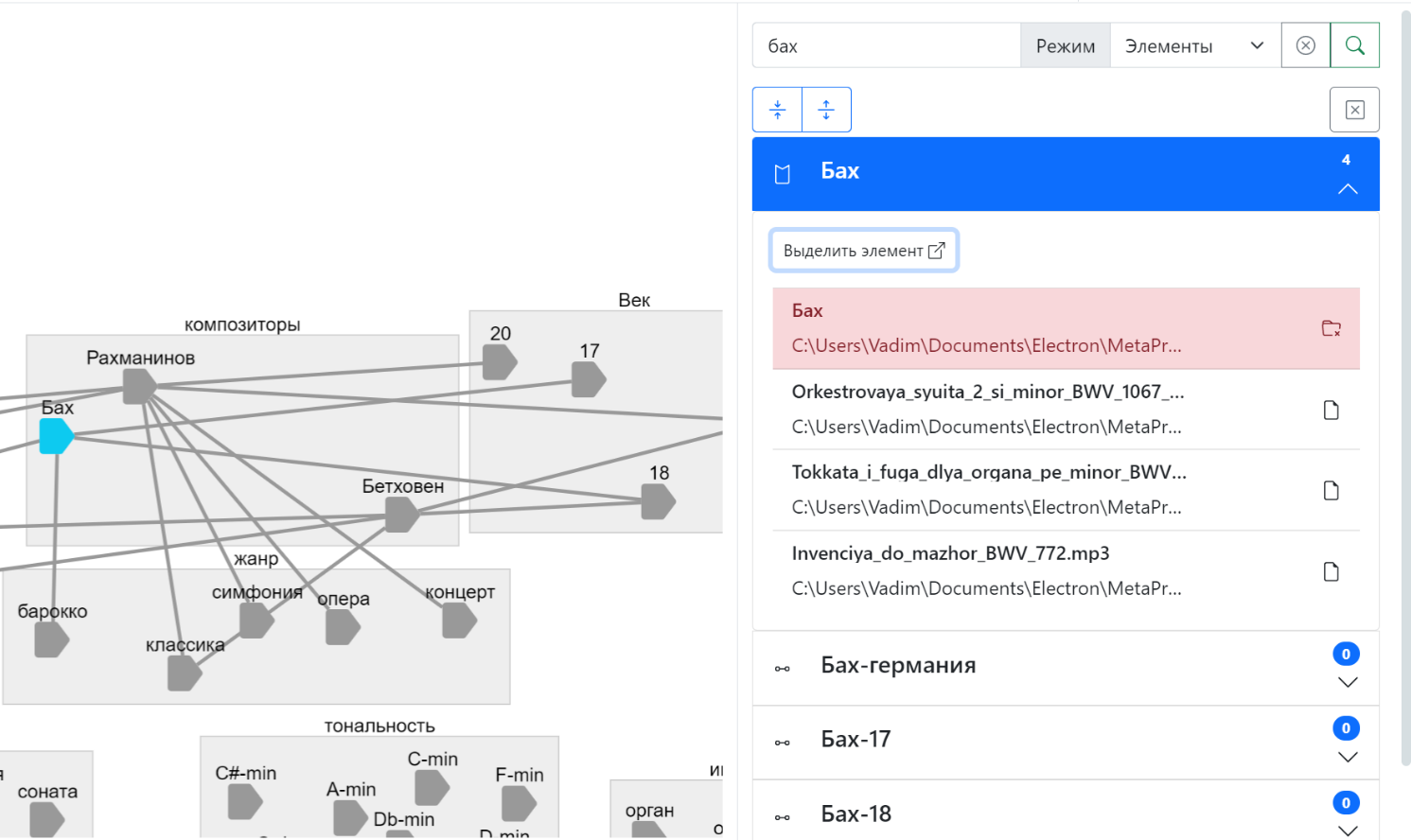


Рисунок – Поиск элементов в метаграфе

При вводе пустого запроса или отсутствии результатов поиска выводится соответствующая информация (рис. 24).

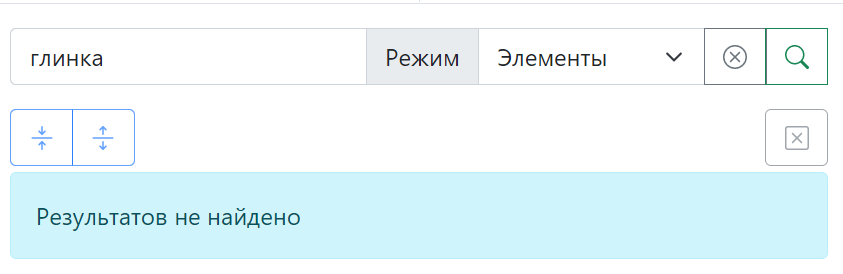


Рисунок – Отображение информации о пустом поиске

# ЗАКЛЮЧЕНИЕ

В результате выполнения Выпускной квалификационной работы бакалавра было разработано приложение, позволяющее организовать файлы на основе семантики вместо стандартных иерархических отношений.

В качестве модели хранения и отображения семантической информации была использована метаграфовая модель, предоставляющая широкие возможности описания отношений между семантическими единицами. Была разработана интерактивная визуализация этой модели для взаимодействия с пользователем.

В ходе разработки были рассмотрены способы хранения метаданных, проведено их сравнение. Кроме того, были проанализированы существующие популярные БД и средства работы с ними для выбора наиболее подходящих компонентов разработанной системы.

Использование кроссплатформенных технологий позволило создать приложение, которое на зависит от особенностей файловых и операционных систем и обеспечивает унифицированный пользовательский интерфейс на поддерживаемых платформах.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Основы эксплуатации автоматизированных систем обработки информации и управления. Краткий курс : учеб. пособие / В. М. Постников. — М.: Изд-во МГТУ им. Н. Э. Баумана, 2013. — 177 c.
2. David K. Gifford. Semantic file systems. / David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, and James W. O'Toole // SIGOPS Oper. Syst. Rev. 25. – 1991. – 5. – 6–25.
3. Semantic File Systems [Электронный ресурс] – URL: http://www.objs.com/survey/OFSExt.htm, (дата обращения: 23.05.2021)
4. Ba-Hung Ngo, Christian Bac, Frederique Silber-Chaussumier. Integrating Ontology into Semantic File Systems // Huitiemes Journees Doctorales en Informatique et Reseaux (JDIR'07) – Marne-la-Vallee, France, 2007. pp.139-142.
5. Why WinFS had to vanish [Электронный ресурс] – URL: https://www.theguardian.com/technology/2006/jun/29/insideit.guardianweeklytechnologysection, (дата обращения: 23.05.2021)
6. Extended file attributes [Электронный ресурс] – URL: https://en.wikipedia.org/wiki/Extended\_file\_attributes, (дата обращения: 27.05.2021)
7. xattr(7) [Электронный ресурс] – URL: https://man7.org/linux/man-pages/man7/xattr.7.html, (дата обращения: 24.05.2021)
8. An introduction to extended attributes, xattrs [Электронный ресурс] – URL: https://eclecticlight.co/2017/12/11/an-introduction-to-extended-attributes-xattrs/, (дата обращения: 24.05.2021)
9. NTFS Attribute Types [Электронный ресурс] – URL: https://docs.microsoft.com/en-us/openspecs/windows\_protocols/ms-fscc/a82e9105-2405-4e37-b2c3-28c773902d85?redirectedfrom=MSDN, (дата обращения: 24.05.2021)
10. Extended attributes and tag file systems [Электронный ресурс] – URL: https://www.lesbonscomptes.com/pages/tagfs.html, (дата обращения: 25.05.2021)
11. Fork (file system) [Электронный ресурс] – URL: https://en.wikipedia.org/wiki/Fork\_(file\_system), (дата обращения: 25.05.2021)
12. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. — СПб.: Питер, 2015. — 1120 с.
13. NTFS [Электронный ресурс] – URL: https://en.wikipedia.org/wiki/NTFS#Alternate\_data\_stream\_(ADS), (дата обращения: 25.05.2021)
14. Organizing files and folders with tags [Электронный ресурс] – URL: https://docs.tagspaces.org/tagging/, (дата обращения: 25.05.2021)
15. Filename [Электронный ресурс] – URL: https://en.wikipedia.org/wiki/Filename#Comparison\_of\_filename\_limitations, (дата обращения: 26.05.2021)
16. Sidecar file [Электронный ресурс] – URL: https://en.wikipedia.org/wiki/Sidecar\_file, (дата обращения: 26.05.2021)
17. Specification of the meta file formats [Электронный ресурс] – URL: https://docs.tagspaces.org/dev/metafileformats/, (дата обращения: 26.05.2021)
18. Extensible Metadata Platform [Электронный ресурс] – URL: https://en.wikipedia.org/wiki/Extensible\_Metadata\_Platform, (дата обращения: 26.05.2021)
19. File system [Электронный ресурс] – URL: https://en.wikipedia.org/wiki/File\_system#Database\_file\_systems, (дата обращения: 27.05.2021)
20. 14 Characteristics of Database Management System [Электронный ресурс] – URL: https://whatisdbms.com/characteristics-of-database-management-system/, (дата обращения: 27.05.2021)
21. FileStream [Электронный ресурс] – URL: https://ru.wikipedia.org/wiki/FileStream, (дата обращения: 27.05.2021)
22. WinFS Architecture [Электронный ресурс] – URL: http://ntfs.com/winfs\_arch.htm, (дата обращения: 27.05.2021)
23. Civan, A. Better to organize personal information by folders or by tags?: The devil is in the details. / Civan, A., Jones W., Klasnja, P. and Bruce, H. // Proc. Am. Soc. Info. Sci. Tech. – 2008. – 45. – 1-13.
24. Конспект лекций по спецкурсу «Гибридные интеллектуальные информационные системы на основе метаграфового подхода»: Учебно-методическое пособие / Гапанюк Ю.Е. – М.: Издательство «Спутник +», 2018. – 53с.
25. Electron (software framework) [Электронный ресурс] – URL: https://en.wikipedia.org/wiki/Electron\_(software\_framework), (дата обращения: 28.05.2021)
26. Electron in Action / Steve Kinney - Manning Publications, 2019. – 376 c.
27. Electron | Build cross-platform desktop apps with JavaScript, HTML, and CSS [Электронный ресурс] – URL: https://www.electronjs.org/, (дата обращения: 28.05.2021)
28. Electron | Pros And Cons. [Электронный ресурс] – URL: https://medium.com/@nalegaveshardul40/electron-pros-and-cons-8f58fd6313d5, (дата обращения: 28.05.2021)
29. TypeScript [Электронный ресурс] – URL: https://en.wikipedia.org/wiki/TypeScript, (дата обращения: 29.05.2021)
30. Relational vs. NoSQL data [Электронный ресурс] – URL: https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/relational-vs-nosql-data, (дата обращения: 29.05.2021).
31. Top 5 Free, Cross-Platform, and Open-Source Database System in 2020 [Электронный ресурс] – URL: https://www.geeksforgeeks.org/top-5-free-cross-platform-and-open-source-database-system-in-2020/, (дата обращения: 27.05.2021).
32. Sqlite vs. MySQL vs. PostgreSQL: A Comparison of Relational Databases [Электронный ресурс] – URL: https://logz.io/blog/relational-database-comparison/, (дата обращения: 29.05.2021).
33. SQL vs ORMs vs Query Builders | Compare: Prisma's Data Guide [Электронный ресурс] – URL: https://www.prisma.io/dataguide/types/relational/comparing-sql-query-builders-and-orms, (дата обращения: 30.05.2021).
34. Object–relational mapping [Электронный ресурс] – URL: https://en.wikipedia.org/wiki/Object%E2%80%93relational\_mapping, (дата обращения: 27.05.2021).
35. Object–relational impedance mismatch [Электронный ресурс] – URL: https://en.wikipedia.org/wiki/Object%E2%80%93relational\_impedance\_mismatch, (дата обращения: 27.05.2021).
36. Top 8 TypeScript ORMs, Query Builders, Libraries: Evaluate Type Safety [Электронный ресурс] – URL: https://www.prisma.io/dataguide/database-tools/evaluating-type-safety-in-the-top-8-typescript-orms, (дата обращения: 27.05.2021).
37. Cytoscape library [Электронный ресурс] – URL: https://js.cytoscape.org/, (дата обращения: 30.05.2021)
38. Диаграмма последовательности [Электронный ресурс] – URL: https://ru.wikipedia.org/?curid=741445&oldid=113572563, (дата обращения: 27.05.2021)
39. Gapanyuk, Y.E., Chernenkiy, V., Gapanyuk, Y.E., Kaganov, Y., Dunin, I., Lyaskovsky, M., Larionov, V. Storing Metagraph Model in Relational, Document-Oriented, and Graph Databases. // DAMDID/RCDL (Moscow, Russia) - Moscow, 2018 - pp.120-127.
40. Ревунков Г.И. Банки данных: Учеб. для вузов. / Григорьев Ю.А., Ревунков Г.И. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2002. – 320 с.

# 

# ПРИЛОЖЕНИЕ A ГРАФИЧЕСКИЕ МАТЕРИАЛЫ

1. Предметная область (Рисунок А.1)
2. Значения критериев качества для аналогов и системы (Таблица А.1)
3. Метаграфовая модель для иерархии (Рисунок А.2)
4. Метаграфовая модель для процесса (Рисунок А.3)
5. Архитектура системы (Рисунок А.4)
6. Диаграмма последовательности взаимодействия пользователя с проектом (Рисунок А.5)
7. Инфологическая модель (Рисунок А.6)
8. Даталогическая модель (Рисунок А.7)
9. Интерфейс взаимодействия (режим процессов) (Рисунок А.8)
10. Интерфейс взаимодействия (режим иерархий) (Рисунок А.9)

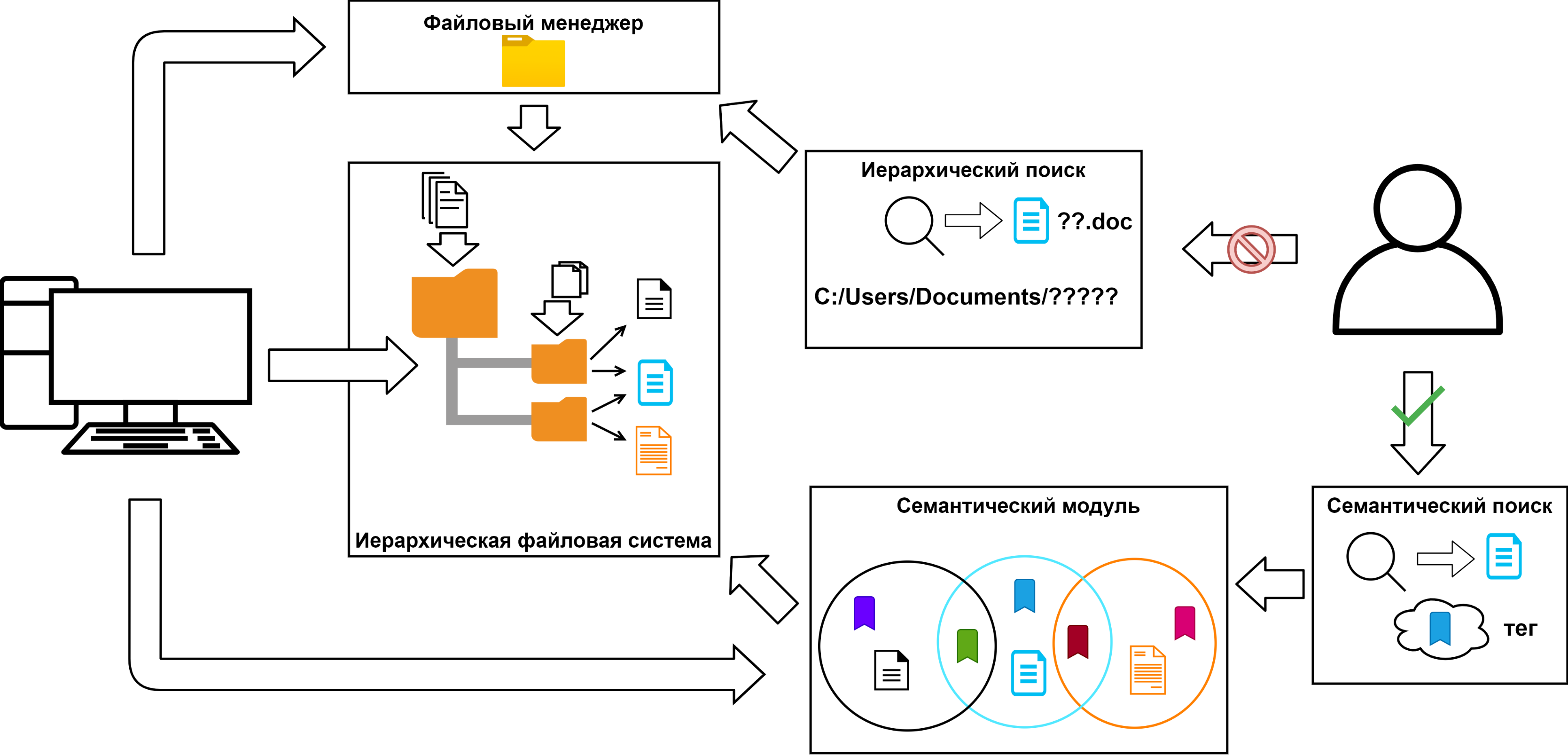


Рисунок A.1 – Предметная область

Таблица A.1 – Значения критериев качества для аналогов и системы

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Код фактора | В1 | В2 | В3 | В4 | В5 | В6 | В7 | В8 | В9 | В10 | В11 |
| Х1 | Windows | Windows | Windows | Windows,Mac, Linux | Windows | Windows,Mac, Linux | Windows, Mac | Windows | Windows,Mac, Linux | Windows | Windows,Mac, Linux |
| Х2 | Любые | Любые | Любые | Преимущественно изображения | Любые | Любые | Любые | Любые | Преимущественно мультимедиа | Любые | Любые |
| Х3 | Файлы и папки | Файлы и папки | Файлы и папки | Файлы и папки | Файлы | Файлы, папки(платно) | Файлы и папки | Файлы и папки | Файлы | Файлы | Файлы и папки |
| Х4 | Вложенные теги, цвет | Группировка | Группировка и вложенные теги | Группировка | Группировка, иерархия | Группировка, цвет | Группировка, иерархия | Группировка, вложенные теги | Группировка, иерархии | Группировка | Группировка, вложенные теги, иерархии, связи |
| Х5 | Нет | Нет | Нет | Нет | Нет | Нет | Нет | Нет | Нет | Нет | Есть |
| Х6 | Есть (умное отслеживание) | Есть (Windows-ярлыки) | Есть | Частично | Частично (хеши) | Есть | Частично | Частично | Есть | Есть | Частично |

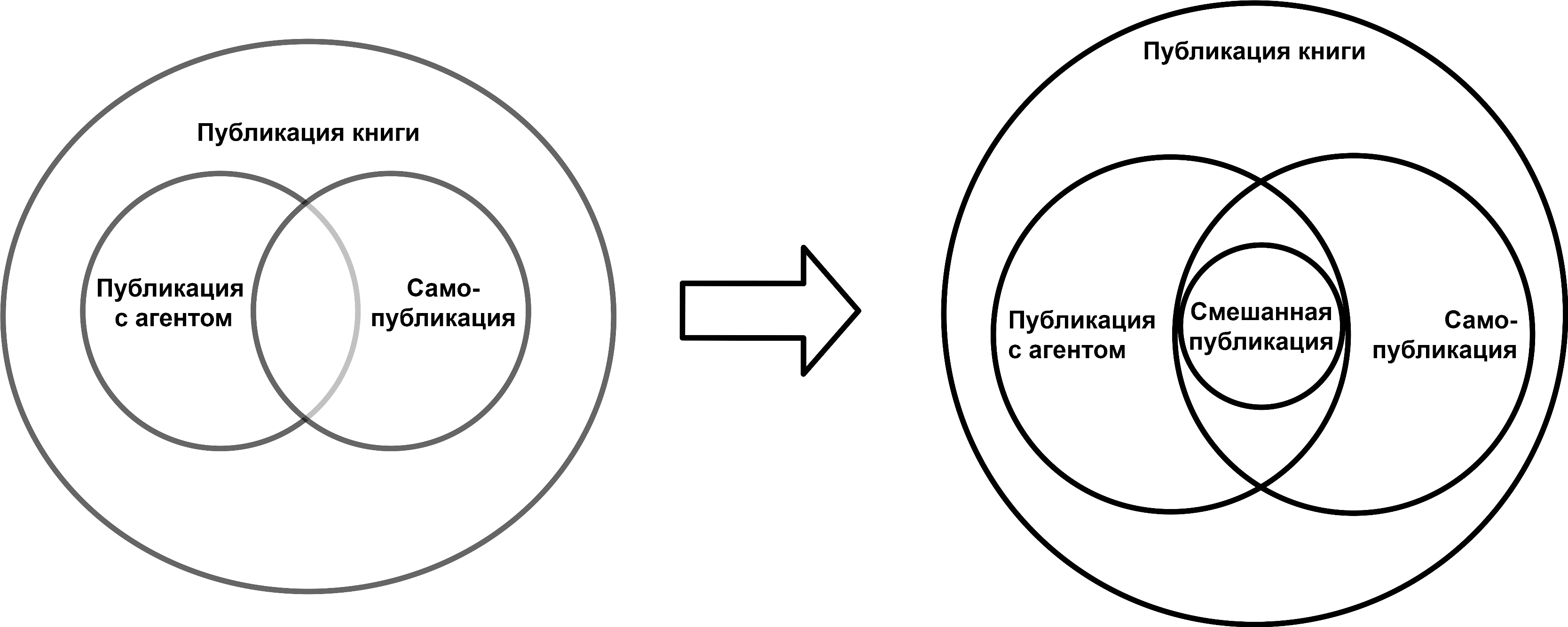


Рисунок A.2 – Метаграфовая модель для иерархии

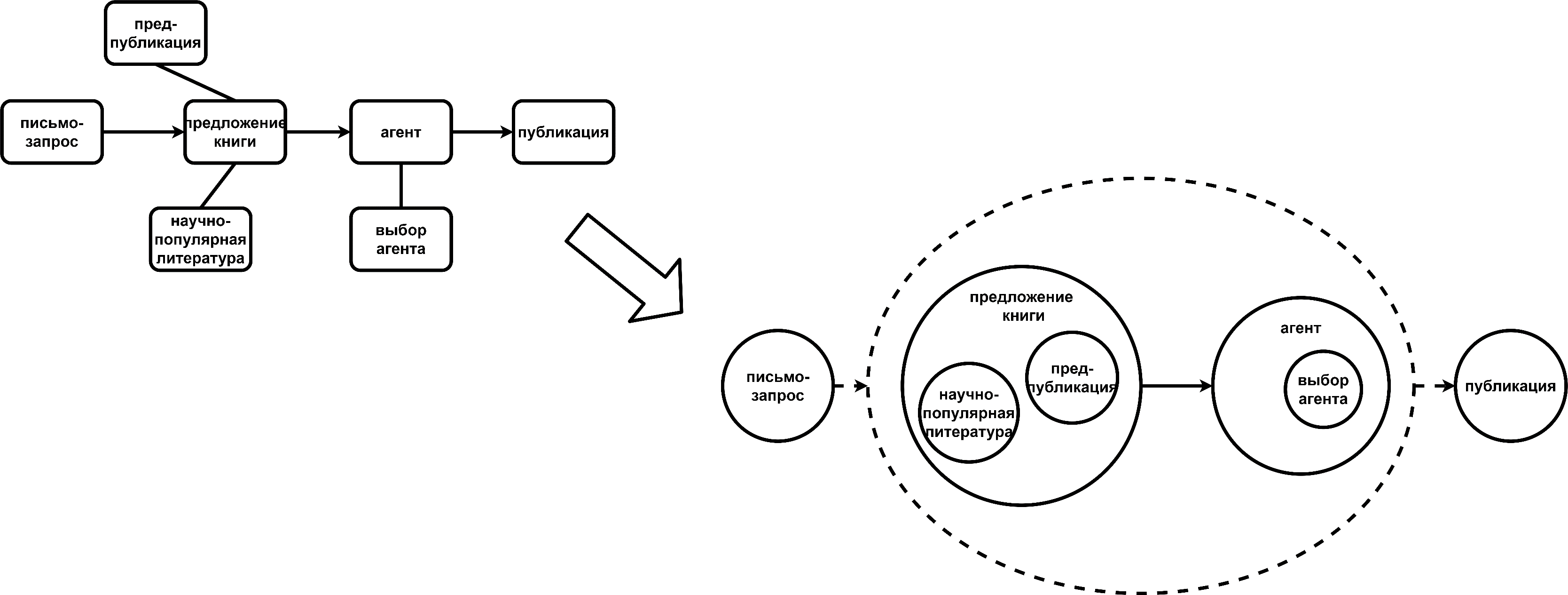


Рисунок А.3 – Метаграфовая модель для процесса

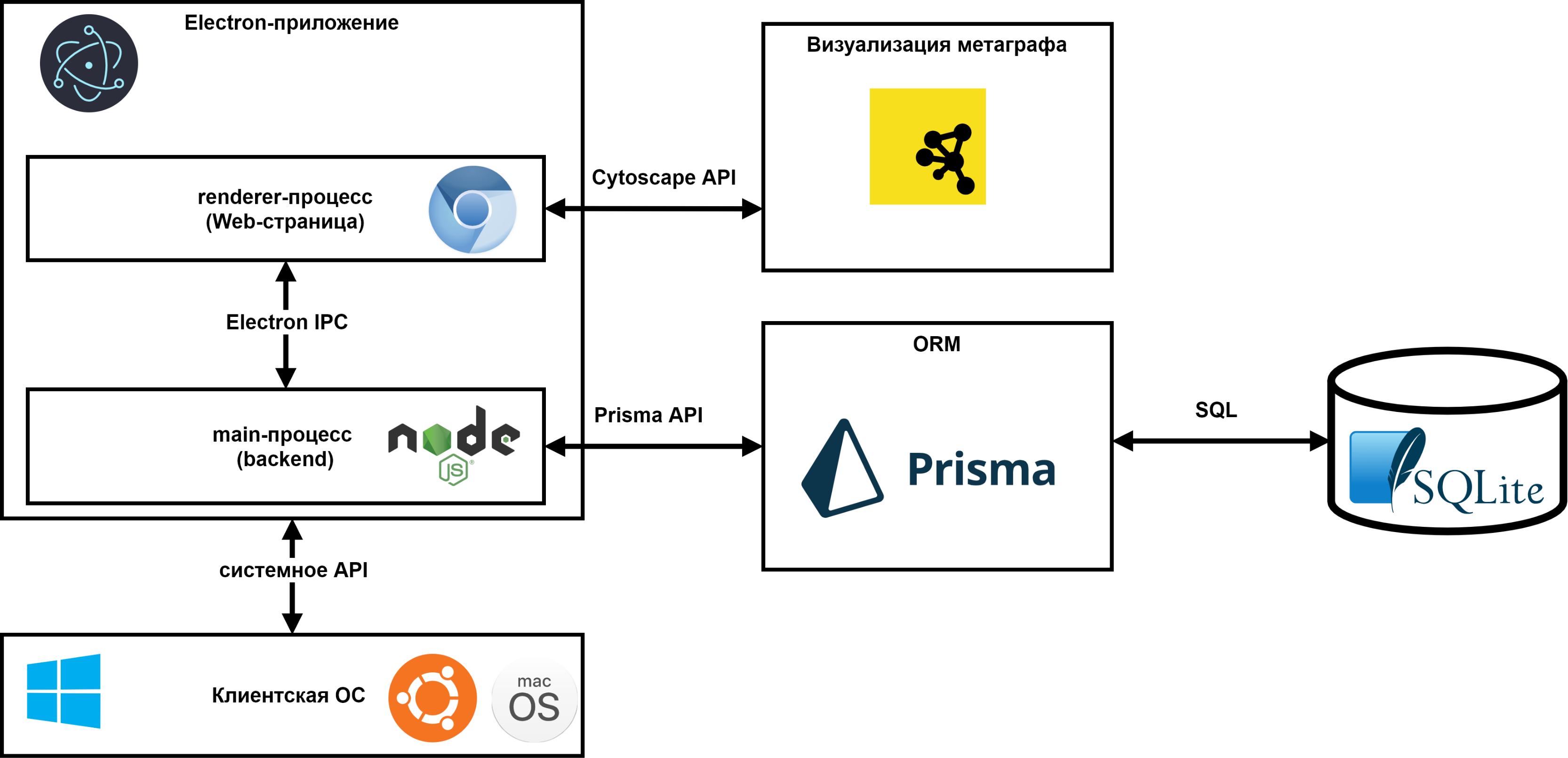


Рисунок А.4 – Архитектура системы

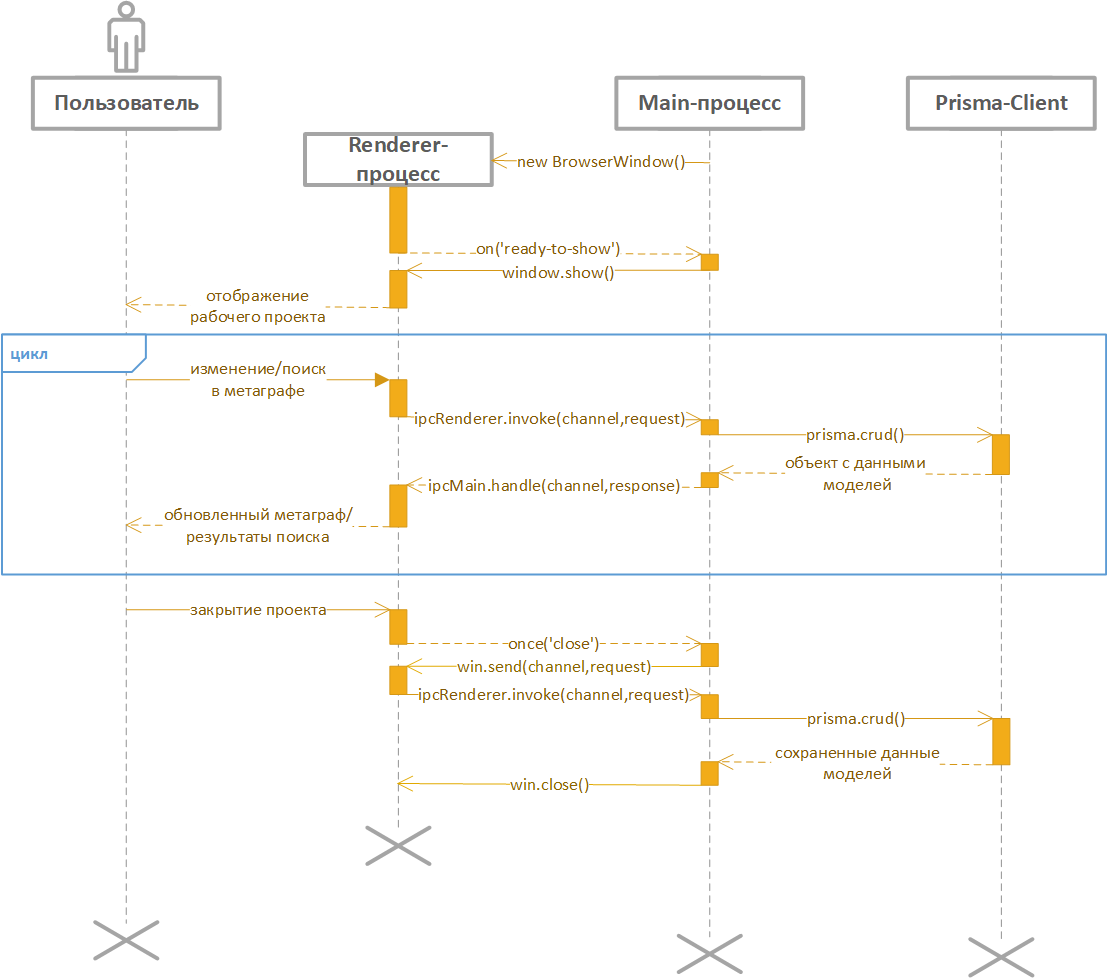


Рисунок А.5 – Диаграмма последовательности взаимодействия пользователя с проектом

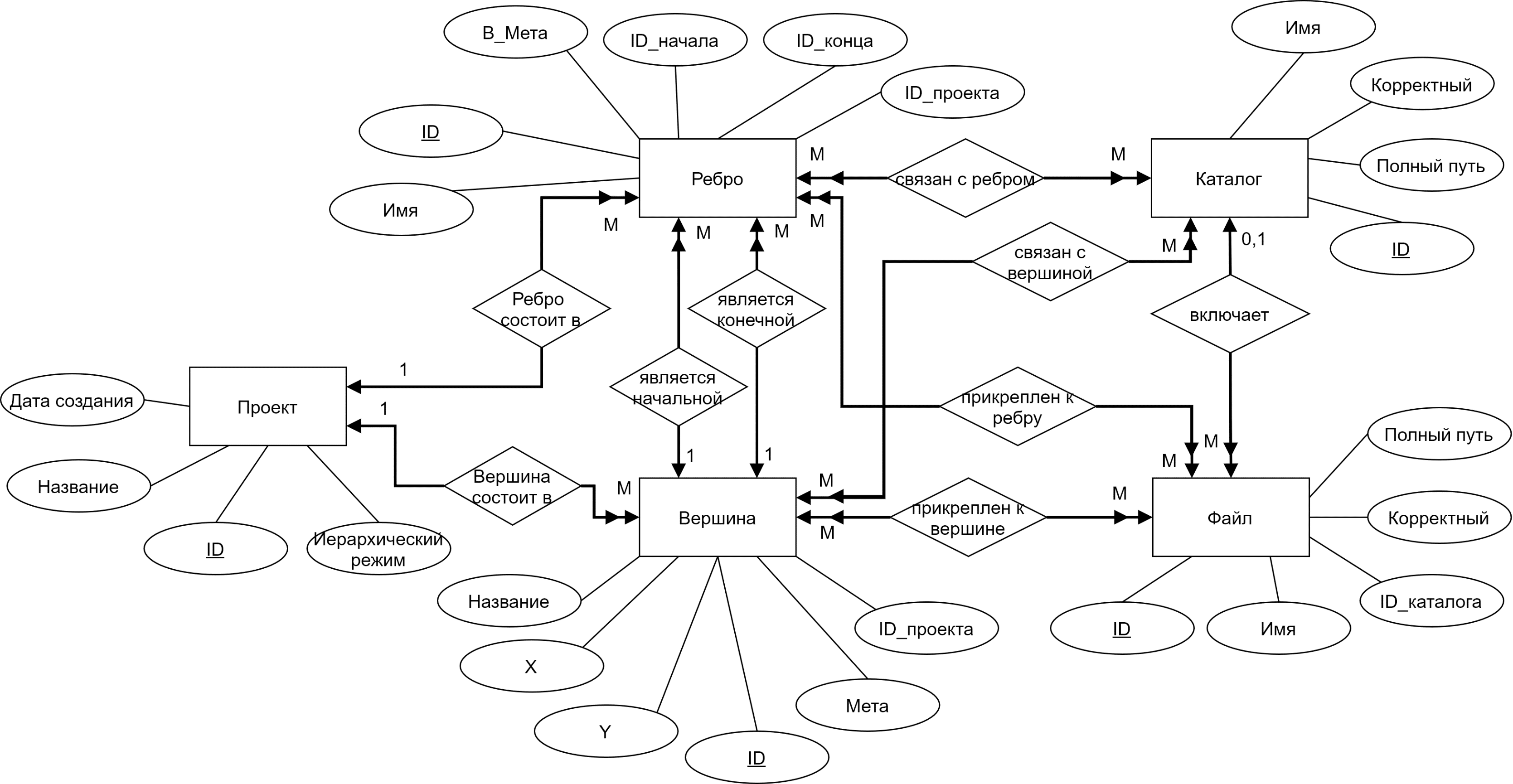


Рисунок А.6 - Инфологическая модель

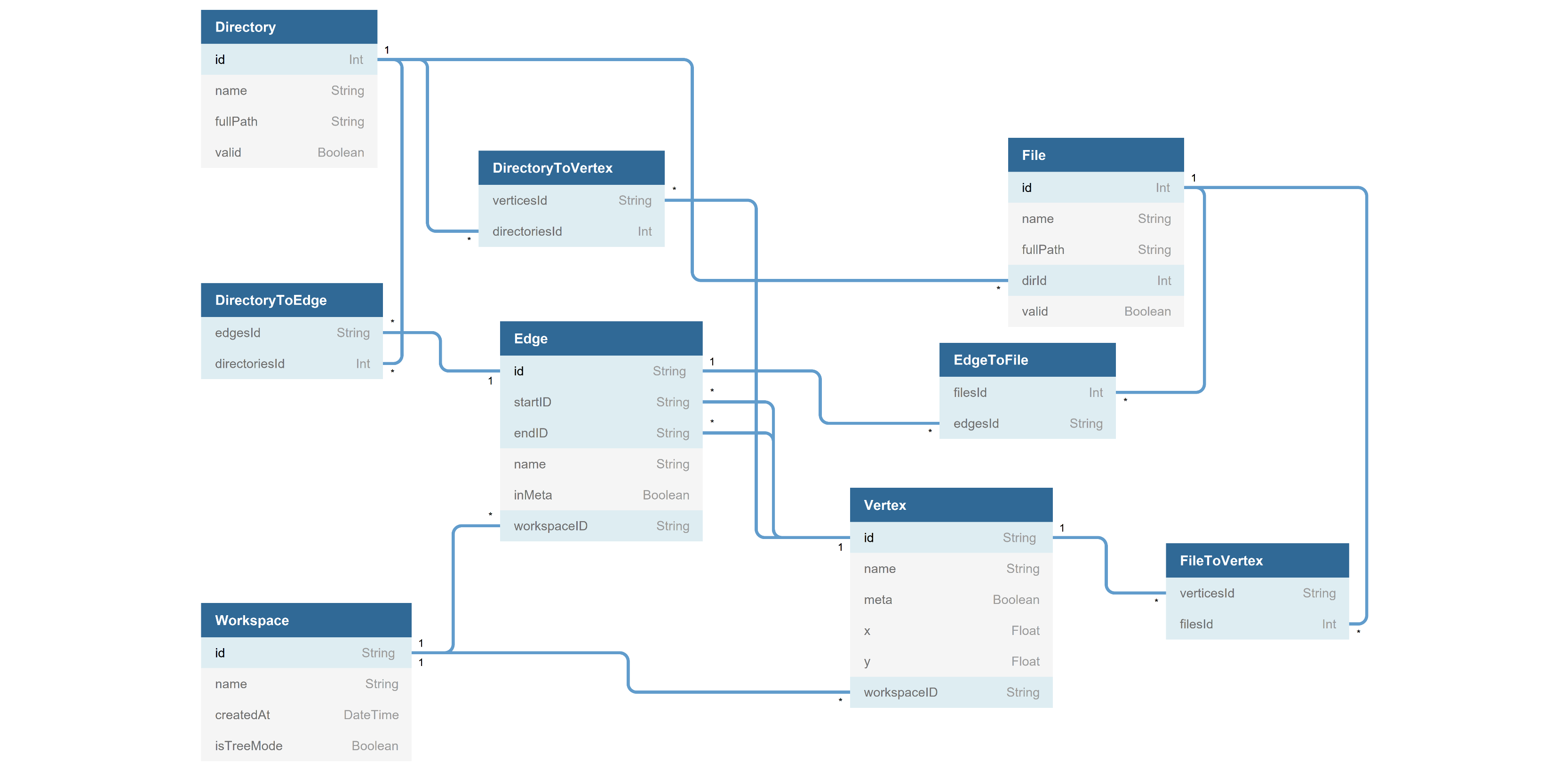


Рисунок А.7 – Даталогическая модель

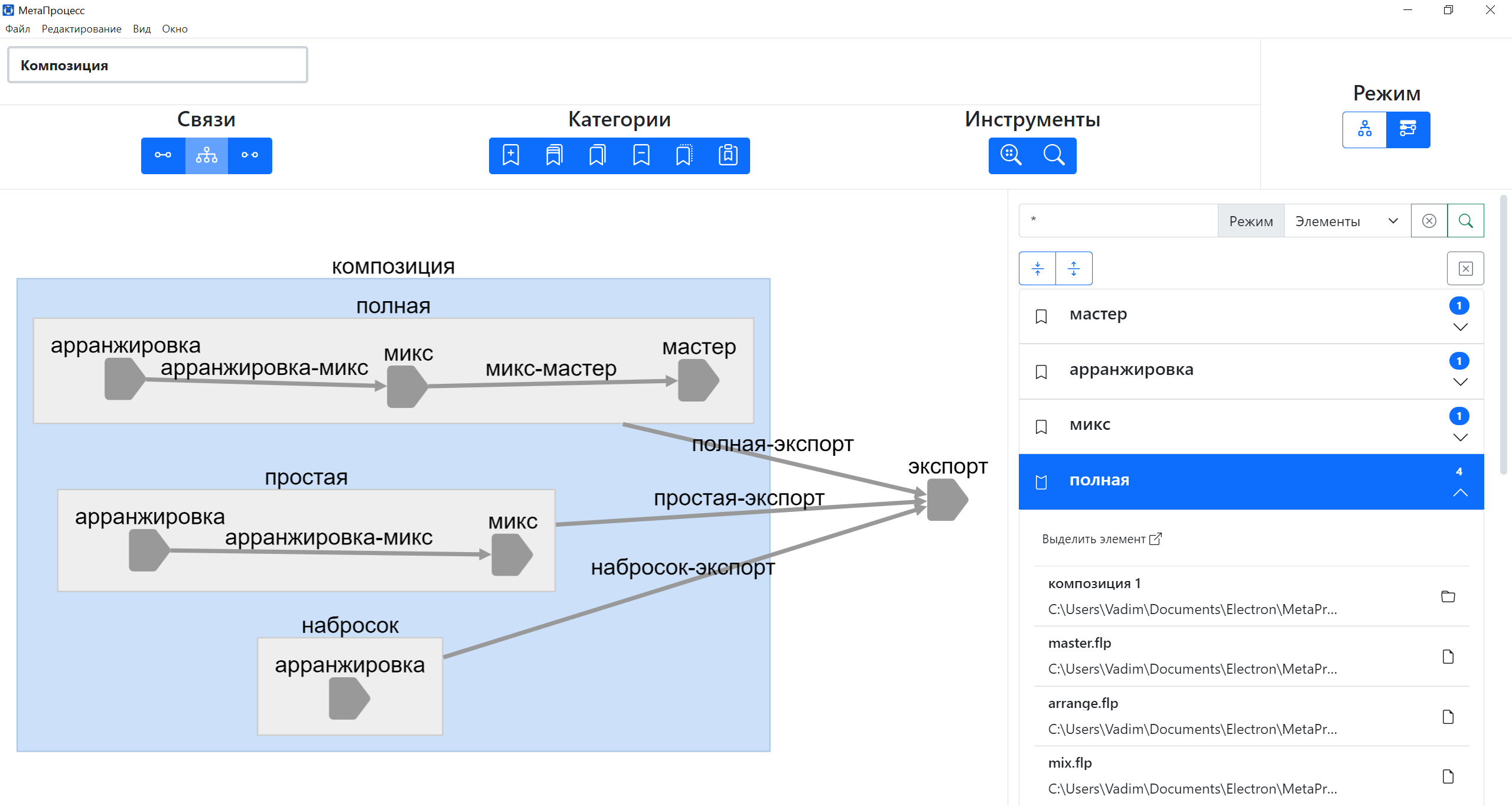


Рисунок А.8 – Интерфейс взаимодействия (режим процессов)

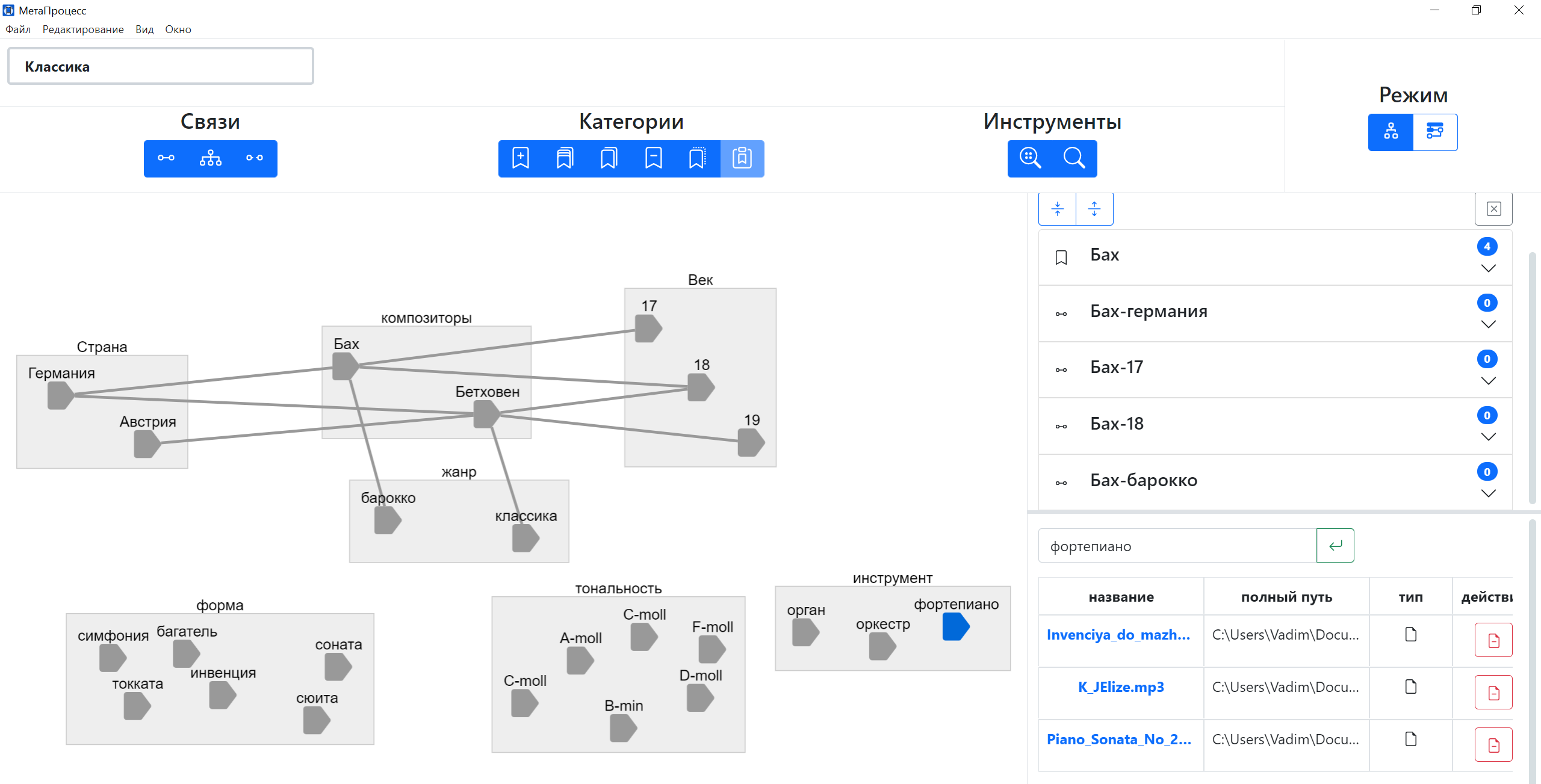


Рисунок А.9 – Интерфейс взаимодействия (режим иерархий)

# ПРИЛОЖЕНИЕ B ТЕХНИЧЕСКОЕ ЗАДАНИЕ

**Министерство образования и науки Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего профессионального образования**

**«Московский государственный технический университет имени Н.Э. Баумана»**

**(МГТУ им. Н.Э.Баумана)**

Утверждаю Согласовано

\_\_\_\_\_\_\_\_\_\_\_ Михеев В.А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_»\_\_\_\_\_\_\_\_\_\_2021 г. «\_\_\_»\_\_\_\_\_\_\_\_\_\_2021 г.

**«Модуль семантической организации файлов на основе метаграфового подхода»**

Техническое задание

(вид документа)

писчая бумага

(вид носителя)

6

(количество листов)

Исполнитель:

студент группы ИУ5-83Б

Труфанов В.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_»\_\_\_\_\_\_\_\_\_\_2021 г.

Москва, 2021 г.

**Наименование**

Модуль семантической организации файлов на основе метаграфового подхода.

1. **Основание для разработки**

Основанием для разработки является задание на выпускную работу, подписанное руководителем выпускной работы и утверждённое заведующим кафедрой. Задание утверждено кафедрой ИУ5 МГТУ им. Н.Э. Баумана.

1. **Исполнитель**

Студент четвёртого курса группы ИУ5-83Б Труфанов В.А.

1. **Назначение и цель работы**

Целью работы является проектирование и разработка приложения, способного организовывать файлы семантически на основе метаграфового подхода.

Назначением работы является обеспечение пользователей ПК более удобным способом организации файлов, использующим семантические связи вместо стандартных иерархических отношений.

1. **Содержание работы**
   1. **Задачи**
2. Исследовать предметную область, определить функциональные задачи.
3. Сравнить существующие аналоги проектируемой системы
4. Обосновать применение метаграфового подхода
5. Разработать архитектуру программного обеспечения.
6. Разработать метаграфовую модель
7. Разработать структуру базы данных
8. Реализовать приложение с интегрированной метаграфовой моделью и базой данных.
9. Провести тестирование информационно-программного продукта.
10. Провести отладку программного продукта.
11. Оформить техническую документацию.
    1. **Требования к функциональным характеристикам**

Разрабатываемая система должна выполнять следующие функции:

1. Создание проекта, содержащего рабочий метаграф
2. Редактирование названия проекта
3. Создание категорий
4. Создание одной связи и множества связей
5. Объединение категорий в мета-категорию
6. Включение категорий в существующую мета-категорию
7. Удаление мета-категории без вложенных объектов
8. Удаление категорий и мета-категорий
9. Удаление связей
10. Редактирование названий созданных элементов метаграфа
11. Отображение категорий, подкатегорий и связей в виде метаграфа
12. Привязка файлов и папок к созданным элементам
13. Возможность заново привязать документ при его переименовании или перемещении
14. Открепление файлов и папок от созданных элементов
15. Дублирование элементов метаграфа
16. Поиск по файлам и папкам, добавленных в проект, по именам с отображением прикрепленных категорий и мета-категорий
17. Поиск элементов метаграфа, добавленных в проект, по присвоенным именам с отображением прикрепленных документов
18. Удаление проекта, добавленного в приложение
    1. **Требования к архитектуре программного изделия**

Программное изделие представляет из себя клиентское десктопное программное обеспечение. Пользователь взаимодействует с изделием как с стандартным приложением, установленным через инсталлятор на ПК. Приложение должно быть кроссплатформенным и поддерживать наиболее популярные ОС – Windows, macOS, Ubuntu.

* 1. **Требования к входным и выходным данным**
     1. **Требования к входным данным**

Данные, полученные от пользователя:

1. Имена создаваемых элементов метаграфа
2. Пути файлов и папок, указывающих их расположение в файловой системе
   * 1. **Требования к выходным данным**

Выходные данные представляют собой метаграфовую модель, которая описывает семантические отношения, заданные пользователем, и содержит ссылки на прикрепленные файлы. Модель визуализируется в удобном для взаимодействия с пользователем виде.

* 1. **Требования к надежности**

Программное изделие не должно выдавать ошибок, не предусмотренных работой; программа должна надежно и устойчиво функционировать.

* 1. **Лингвистические требования**

Интерфейс приложения должен быть русифицирован.

* 1. **Требования к составу программных средств**

Для работы приложения на компьютере пользователя требуется:

1. ОС, указанная в следующем списке:
   1. Windows 7 или выше, x86 или x64
   2. macOS 10.10 или выше, x64
   3. Ubuntu 14.04 или выше, x64
   4. **Требования к составу технических средств**

Минимальные системные требования для работы приложения:

1. Процессор Intel Pentium 4 и выше с поддержкой SSE3 или аналогичный процессор.
2. Оперативная память - 1 ГБ
3. Видеоадаптер и монитор, способные обеспечить графический режим 1024\*768 точек с 32-ти битной цветопередачей.
4. Жёсткий диск объемом 64 ГБ.
5. Манипулятор «мышь» или другое указывающее устройство.
6. Клавиатура.
7. **Этапы работы**

График выполнения отдельных этапов работ приведен в соответствии с приказом об организации учебного процесса в 2020/2021 учебном году.

Таблица 1: Этапы разработки

| № п/п | Наименование этапа и содержание работ | Сроки исполнения |
| --- | --- | --- |
| 1 | Разработка и утверждение ТЗ | Март 2021г. |
| 2 | Исследование предметной области | Март-Апрель 2021г. |
| 3 | Разработка архитектуры программного обеспечения | Апрель 2021 г. |
| 4 | Создание программ | Апрель — Май 2021 г. |
| 5 | Тестирование и отладка | Май 2021 г. |
| 6 | Оформление документации | Май — Июнь 2021 г. |
| 7 | Защита работы | Июнь 2021 г. |

1. **Техническая документация**

По окончании работы предъявляется следующая техническая документация:

1. Техническое задание.
2. Расчётно-пояснительная записка.
3. Программа и методика испытаний.
4. Графический материал по проекту в формате презентации.
5. **Порядок приема работы**

Приём и контроль программного изделия осуществляется в соответствие с методикой испытаний (см. документ «Программа и методика испытаний»).

1. **Дополнительные условия**

Данное техническое задание может уточняться в установленном порядке.

# ПРИЛОЖЕНИЕ C ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

**Министерство образования и науки Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего профессионального образования**

**«Московский государственный технический университет имени Н.Э. Баумана»**

**(МГТУ им. Н.Э.Баумана)**

Утверждаю Согласовано

\_\_\_\_\_\_\_\_\_\_\_ Михеев В.А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_»\_\_\_\_\_\_\_\_\_\_2021 г. «\_\_\_»\_\_\_\_\_\_\_\_\_\_2021 г.

**«Модуль семантической организации файлов на основе метаграфового подхода»**

Программа и методика испытаний

(вид документа)

писчая бумага

(вид носителя)

9

(количество листов)

Исполнитель:

студент группы ИУ5-83Б

Труфанов В.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_»\_\_\_\_\_\_\_\_\_\_2021 г.

Москва, 2021 г.

1. **Объект испытаний**

Объектом испытаний является приложение для семантической организации файлов на основе метаграфового подхода. Краткое название объекта испытаний: «приложение».

1. **Цель испытаний**

Испытания проводятся с целью проверки соответствия системы требованиям, указанным в техническом задании.

1. **Состав предъявляемой документации**

Перед проведением испытаний предъявляются следующие документы:

1. Техническое задание
2. Программа и методика испытаний
3. **Технические требования**
   1. **Требования к программному обеспечению**

Для работы приложения на компьютере пользователя требуется:

1. ОС, указанная в следующем списке:
   1. Windows 7 или выше, x86 или x64
   2. macOS 10.10 или выше, x64
   3. Ubuntu 14.04 или выше, x64
   4. **Требования к аппаратному обеспечению**

Минимальные системные требования для работы приложения:

1. Процессор Intel Pentium 4 и выше с поддержкой SSE3 или аналогичный процессор.
2. Оперативная память - 1 ГБ
3. Видеоадаптер и монитор, способные обеспечить графический режим 1024\*768 точек с 32-ти битной цветопередачей.
4. Жёсткий диск объемом 64 ГБ.
5. Манипулятор «мышь» или другое указывающее устройство.
6. Клавиатура.
7. **Порядок проведения испытаний**

Необходимо обеспечить соответствие состава и структуры технических и программных средств техническому заданию.

Для проведения требуется:

1. Проверить статус установки приложения на ПК
2. Открыть приложение на ПК с помощью нажатия на ярлык
3. **Последовательность проведения испытаний**

| № | Пункт ТЗ | Проверяемая функция | Действия пользователя | Результат |
| --- | --- | --- | --- | --- |
|  | 5.2.1 Создание проекта, содержащего рабочий метаграф | Создание проекта | На начальном экране нажать на кнопку «Добавить» (пиктограмма «Плюс») | В таблицу на начальном экране добавится новая строка, отображающая проект с названием по умолчанию «МетаПроцесс». |
|  | 5.2.2 Редактирование названия проекта | Изменение названия проекта | Нажать на добавленный проект в таблице, в открывшемся окне с помощью верхнего поля ввода изменить название проекта по умолчанию на «Тест» | Название проекта изменится на «Тест», при закрытии проекта отобразится новое имя в таблице на начальном экране. |
|  | 5.2.3 Создание категорий | Добавление категории | В открытом проекта нажать на кнопку «Добавить категорию» в разделе «Категории» | В рабочей области появится элемента метаграфа с именем «категория». |
|  | 5.2.4 Создание одной связи и множества связей | Добавление связей | Добавить несколько категорий согласно пункту 5.2.3. Нажать на кнопку «Добавить связь» в разделе «Связи», выбрать начальную и конечную категорию. Нажать на кнопку «Добавить связи с общим началом» в разделе «Связи», выбрать начальную и конечные категории, нажать Ввод. | Успех:  В рабочей области появится связь между элементами метаграфа. При добавлении связей с общим началом появится множество связей. |
| Неудача:  При повторной попытке добавления связи появится ошибка о существовании связи.  При попытке создать связь-петлю, отобразится соответствующая ошибка. |
|  | 5.2.5 Объединение категорий в мета-категорию | Создание мета-категорий на основе категорий | Нажать на кнопку «Объединить в мета-категорию» в разделе «Категории», выбрать категории, нажать Ввод. | Успех:  В рабочей области выбранные категории будут помещены в прямоугольную область, отображающую мета-категорию. |
| Неудача:  При попытке объединения категорий, включенных в разные мета-категории, появится ошибка о невозможности выполнения операции. |
|  | 5.2.6 Включение категорий в существующую мета-категорию | Включение категорий в мета-категорию | Нажать на кнопку «Включить в мета-категорию» в разделе «Категории», выбрать мета-категорию, затем включаемые категории, нажать Ввод. | В рабочей области выбранные категории будут помещены в мета-категорию, если в качестве мета-категория выбрана обычная категория, то она становится мета-категорией. Если выбранные категории были вложены в мета-категории, то они исключаются из этих мета-категорий. |
|  | 5.2.7 Удаление мета-категории без вложенных объектов | Удаление мета-категорий без вложенных элементов | Нажать на кнопку «Удалить мета-категорию» в разделе «Категории», выбрать мета-категорию, нажать Ввод. | В рабочей области исчезнет мета-категория и связи с ней (при наличии), вложенные элементы сохранятся. |
|  | 5.2.8 Удаление категорий и мета-категорий | Удаление категорий и мета-категорий со всеми вложенными элементами | Нажать на кнопку «Удалить категорию» в разделе «Категории», выбрать категорию и мета-категорию, нажать Ввод. | В рабочей области исчезнет выбранная категория и связи с ней (при наличии), также исчезнет мета-категория и все вложенные элементы со связями. |
|  | 5.2.9 Удаление связей | Удаление связей между категориями | Нажать на кнопку «Удалить связь» в разделе «Связи», выбрать связи, нажать Ввод. | В рабочей области исчезнут выбранные связи между элементами. |
|  | 5.2.10 Редактирование названий созданных элементов метаграфа | Изменение названия элементов метаграфа | Выделить связь, категорию или мета-категорию, в открывшейся боковой панели ввести новое имя в поле ввода, нажать Ввод. | На рабочей области отобразится новое имя элемента метаграфа. |
|  | 5.2.11 Отображение категорий, подкатегорий и связей в виде метаграфа | Визуализация метаграфа | Выделить элементы метаграфа, поменять их положение с помощью мыши, настроить требуемый масштаб с помощью колеса прокрутки. Нажать на кнопку «Вписать все элементы» в разделе «Инструменты». | Элементы метаграфа передвинулись в новое местоположение, отображаемые размеры объектов изменились, после нажатия кнопки на экране находятся все элементы рабочего метаграфа. |
|  | 5.2.12 Привязка файлов и папок к созданным элементам | Привязка документов к элементам метаграфа. | Выделить элемент метаграфа, в открывшейся боковой панели нажать на кнопку «Добавить файл» ниже поля ввода, выбрать файл с помощью открывшегося диалогового окна. Повторить действия, используя кнопку «Добавить папку».  Открыть документ с помощью нажатия на имя документа в появившейся таблице. | В боковой панели ниже имени элемента метаграфа появится таблица, содержащая информацию о документе. После нажатия на имя документа, он открывается в приложении по умолчанию для ОС. |
|  | 5.2.13 Возможность заново привязать документ при его переименовании или перемещении | Привязка документов, изменивших свойства, к элементам метаграфа. | Переместить или переименовать документ, привязанный к рабочему метаграфу, вне приложения с помощью стандартного проводника ОС. Выделить элемент метаграфа, который имеет связь с данным документом. Нажать на имя документа в таблице. После сообщения об ошибке открытия заново привязать документ с помощью открывшегося диалогового окна после повторного нажатия на имя. | В боковой панели в таблице, после попытки открыть измененный документ отображается ошибка об открытии, и строка с документов выделяется красным цветом. После выбора документа при повторном нажатии на документ, информация о документе в таблице обновлена. |
|  | 5.2.14 Открепление файлов и папок от созданных элементов | Открепление документов от элементов метаграфа. | Выделить элемент метаграфа, который имеет связь с данным документом. В открывшейся боковой панели в строке с документом нажать на кнопку «Удалить», выделенную красным цветом. | Строка, содержащая информацию о документе, исчезнет из таблицы. |
|  | 5.2.15 Дублирование элементов метаграфа | Создание копии элементов метаграфа с привязанными документами. | Перевести переключатель в разделе «Режим» в состояние «Процесс». Нажать на кнопку «Дублировать элементы» в разделе «Категории»,  выделить мета-категорию, в которую вложены несколько категорий и связей с привязанными документами согласно 5.2.12, нажать Ввод. Уменьшить масштаб рабочей области. | В рабочей области появится копия мета-категории с исходным расположением категорией и связей. К именам элементов и документов добавлена строка «\_копия», документы в таблице в боковой панели отображаются красным цветом, обозначая возможность новой привязки согласно 5.2.13. |
|  | 5.2.16 Поиск по файлам и папкам, добавленных в проект, по именам с отображением прикрепленных категорий и мета-категорий | Поиск по прикрепленным документам | После привязки документов согласно 5.2.12 нажать на кнопку «Открыть поиск» в разделе «Инструменты». Ввести поисковой запрос, содержащий название документа в поле ввода | Успех:  При наличии результатов поиска выводится список документов с возможностью раскрытия его элементов для открытия документа приложением по умолчанию и отображения привязанных категорий и мета-категорий. |
| Неудача:  При отсутствии результатов поиска или при пустом поисковом запросе выводится соответствующее сообщение. |
|  | 5.2.17 Поиск элементов метаграфа, добавленных в проект, по присвоенным именам с отображением прикрепленных документов | Поиск по элементам метаграфа | После создания элементов согласно 5.2.3-5.2.6 и привязки документов согласно 5.2.12 нажать на кнопку «Открыть поиск» в разделе «Инструменты». Ввести поисковой запрос, содержащий название элемента метаграфа в поле ввода | Успех:  При наличии результатов поиска выводится список элементов метаграфа с возможностью раскрытия его элементов для выделения элемента на рабочей области и отображения привязанных документов с возможностью открытия приложением по умолчанию. |
| Неудача:  При отсутствии результатов поиска или при пустом поисковом запросе выводится соответствующее сообщение. |
|  | 5.2.18 Удаление проекта, добавленного в приложение | Удаление проекта с рабочим метаграфом | Закрыть окно с открытым проектом. На начальном экране нажать на строке, содержащий проект «Тест», нажать на кнопку «Удалить», выделенную красным цветом. | В таблице на начальном экране исчезнет строка, отображающая проект с названием «Тест». |