МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки	09.04.04 «Программная инженерия»
Магистерская программа	Программное обеспечение вычислительной техники и автоматизированных систем

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

Представление семантических метаданных в информационных системах

Студент _____ _____Дзюба Е.А._____
(подпись, дата) (ФИО полностью)

Руководитель _____ _____Макарычев П.П._____
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ _____Такташкин Д.В._____
(подпись, дата) (фамилия, инициалы)

Рецензент _____ _____ _____Афонин А.Ю._____
(должность, место работы) (подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____№_____)

Заведующий кафедрой _____ _Макарычев П.П._____
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____№____)

Секретарь ГЭК _____ ____Попова Н.А._____
(подпись) (фамилия, инициалы)

Пенза, 2020

Содержание

Введение 3
1 Анализ видов и функций метаданных и их представления в информационных системах 6
1.1 Определение и классификация метаданных 6
1.2 Обзор систем, использующих метаданные 10
1.3 Обобщенная архитектура систем, основанных на семантических метаданных 13
2 Представление семантических метаданных в информационных системах. 16
2.1 Варианты представления семантических метаданных 16
2.2 Формализация схемы метаданных 17
2.3 Варианты отображения метаданных в реляционной модели. 22
Выводы по главе 2 31
3 Представление семантических метаданных в подсистеме «Формы» 32
3.1. Назначение и функциональные возможности подсистемы «Формы» 32
3.1 Построение схемы семантических метаданных. 33
3.2 Проектирование БД 37
3.3 Архитектура ПС 43
3.5 Реализация подсистемы Форма 47
3.6 Оценка алгоритма добавления новой формы в базу метаданные 54
Вывод по главе 3 56
Заключение 57
Список использованных источников 58
Приложение А Карта знаний 63
Приложение Б Листинг 66

Введение

Актуальность работы. Одно из первых упоминаний метаданных в научных статьях датируется 1986 годом, хотя метаданные начали использоваться задолго до рождения этого термина. В настоящее время выделяется большое количество определений и типов метаданных, представляемое множеством сфер применения, разнообразием природы описываемых ресурсов и подходов к представлению метаданных.

По версии Meta Data Coalition, метаданные определяются, как описательная информация о структуре и смысле данных, а также приложений и процессов, которые манипулируют данными [1]. А информационные системы, основанных на знаниях, нашли применение в бизнесе, производстве, в военной промышленности, сельском хозяйстве, медицине, социологии, геология, космосе и многих других отраслях.

Для того чтобы получать конкурентные преимущества от использования готовых решений или применять новые возможности в быстро меняющейся бизнес-среде, информационные системы предприятий должны постоянно обновляться. Традиционные решения встраивания бизнес-логики в код влекут за собою разработку очередной версии продукта. Информационную систему, которая помимо жестко запрограммированного каркаса содержит выделенную и настраиваемую часть, управляемую знаниями, сформулированными в виде метаданных, гораздо дешевле, быстрее и легче модифицировать.

С появлением стандартов и языков описания, системы могут добавлять новые знания не только от экспертов и аналитиков, но и обмениваясь информацией с другими системами.

Объектом исследования являются реляционное представление семантических метаданных подсистемы «Формы» платформы «Эксперимент». **Предметом исследования** являются варианты представления, онтологическое описание и алгоритм отображения в реляционную модель семантических метаданных информационной системы.

Цель и задачи работы. Целью работы являются исследование видов, функций метаданных, вариантов представления их в информационных системах, алгоритм преобразования семантических метаданных в реляционную модель метаданных информационной системы. На данный момент системы обработки событий являются интенсивно развивающейся областью исследований в информационных технологиях. Для достижения поставленной цели планируется решить ряд важных задач.

1. Исследование метаданных информационных систем, их видов и функций, современных методов представления метаданных в информационных системах.

2. Обзор существующих решений информационных систем на основе метаданных.

3. Разработка формального представления семантических метаданных.

4. Разработка алгоритма отображения онтологии семантических метаданных в реляционную модель метаданных системы.

5.Реализация отображения метаданных в подсистеме «Форма» и программных средств для работы с ней.

Научная новизна, оригинальность идей в работе.

1. Разработана карта знаний метаданных с выделением функций и видов современных метаданных информационных систем.

2. Разработано формальное описание семантических метаданных в виде онтологии.

3. Разработаны алгоритм отображения семантических метаданных в реляционную базу метаданных.

Методы проведенных исследований. Методы исследований основаны на теоретических положениях анализа данных, объектно-ориентированного моделирования, онтологического исследования.

Основные положения, выносимые на защиту.

1. Карта систематизации знаний метаданных

2. Онтология семантических метаданных и ее формализация

3. Алгоритмы отображения семантических метаданных в реляционную базу данных.

4. Архитектура и реализация подсистемы с использованием сематнических метаданных.

Апробация работы. Основные положения и результаты работы докладывались на следующих конференциях:

VII Всероссийской межвузовской научно-практической конференции «Информационные технологии в науке и образовании. Проблемы и перспективы Сборник статей по материалам VII Всероссийской межвузовской научно-практической конференции» 2020 г.

Программные системы и вычислительные методы. 2018.

Моделирование, оптимизация и информационные технологии. 2018.

XXXII Международной научно-технической конференции Проблемы автоматизации и управления в технических системах 2017

XIII Международной научно-технической конференции Новые информационные технологии и системы сборник научных статей XIII Международной научно-технической конференции. 2016

XVIII Международная научно-методическая конференция, посвященная 200-летию со дня рождения М. Ю. Лермонтова Университетское образование, 2014 г.

Международного симпозиума Надежность и качество. 2012, 2010 гг

Структура и объем диссертации. Диссертация состоит из введения, 3 глав, заключения, списка использованных источников, включающего 43 источников, и 3 приложений. Материал диссертации изложен на 99 листах, включает 21 рисунок, 5 таблиц.

1 Анализ видов и функций метаданных и их представления в информационных системах

Данная глава посвящена анализу видов и функций метаданных, построении классификации метаданных, анализу общей архитектуры информационных систем, основанных на метаданных, обзор существующих решений.

1.1 Определение и классификация метаданных

Наличие выделенных в особую часть информационной системы метаданных приводит к более быстрому и удобному способу изменять бизнес-логику работы системы без переписывания основного кода. Такое преимущество позволяет использовать разные виды метаданных. Определение, виды и цели применения метаданных в литературе имеют разную трактовку. Самое распространенное определение описывает метаданные как данные о данных и рассматривает как средство описания контента и характеристик информационных ресурсов [4]. Информационные системы, основанных на знаниях, нашли применение в бизнесе, производстве, в военной промышленности, сельском хозяйстве, медицине, социологии, геология, космосе и многих других отраслях.

Выделяется большое количество определений и типов метаданных, представляемое множеством сфер применения, разнообразием природы описываемых ресурсов и подходов к представлению метаданных. По версии Meta Data Coalition, метаданные определяются, как описательная информация о структуре и смысле данных, а также приложений и процессов, которые манипулируют данными[1]. Понятие метаданных уже давно и успешно используют в таких контекстах, как информационные хранилища и системы аналитической обработки данных, электронный документооборот и управление потоками работ, управление знаниями. К метаданным относят информацию о составе данных, их структуре, месте хранения поддерживающих их информационных систем, технологий, пользователей, методов и полномочия доступа, источниках данных, области действия,

критериях поиска и выборки, структурные характеристики данных, ограничения целостности, семантика данных и другие их свойства.

Существуют классификации метаданных по их функциям, по уровням семантической абстрактности, по их свойствам, по уровням информационной архитектуры, к которым относятся описываемые ресурсы, а также по ряду других критериев. Классификация метаданных в виде карты знаний приведена на рисунке 1.

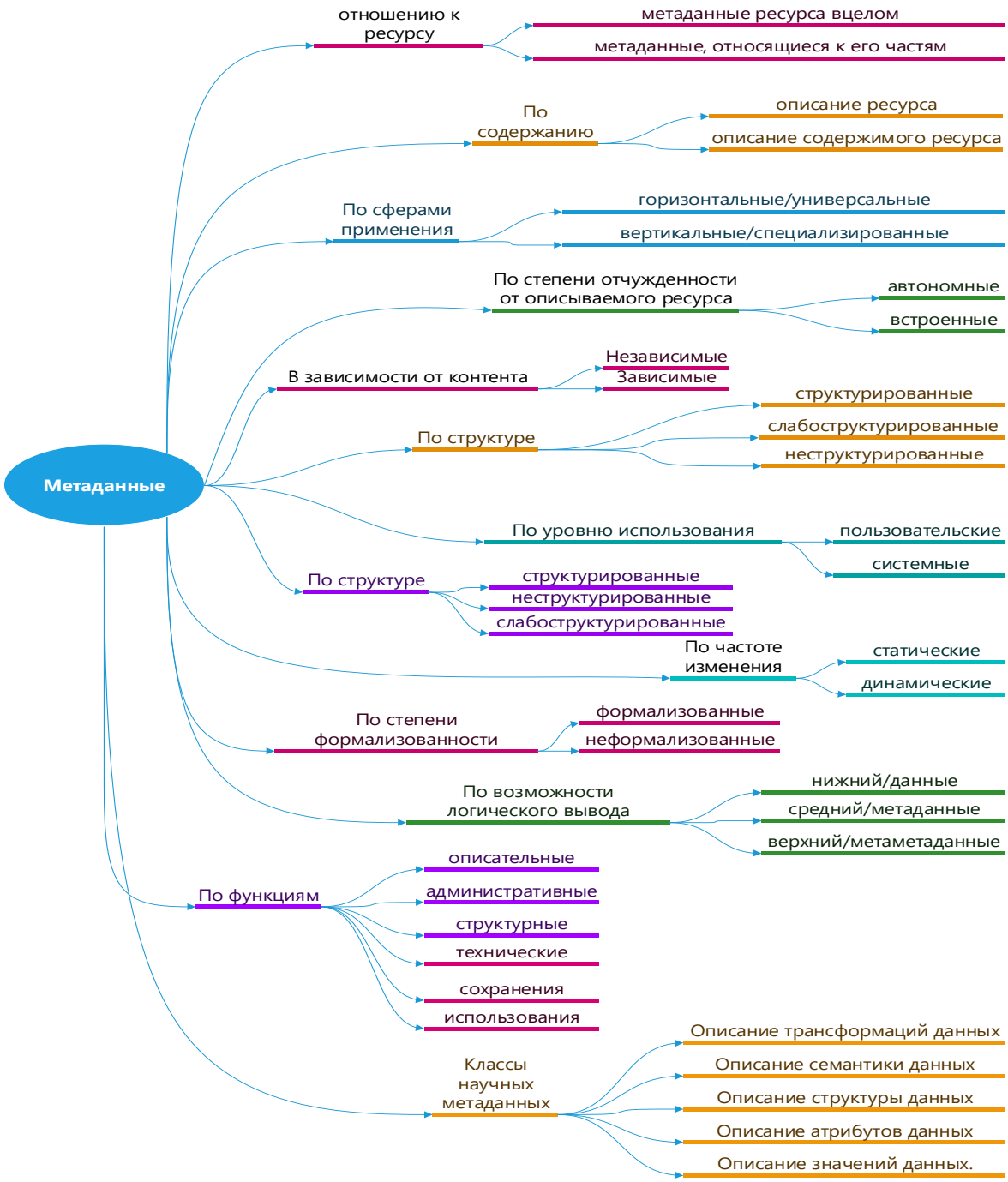


Рисунок 1- Классификация метаданных

Метаданные делятся на типы по сферам применения: горизонтальной сферы, универсальные и метаданные вертикальной сферы, то есть специализированные. Структура метаданных позволяет выделить структурированные, неструктурированные и слабоструктурированные метаданные. Структурированные в виде иерархии метаданные называют схемой метаданных или онтологией.

По степени отчужденности от описываемого ресурса можно выделить автономные метаданные, отчужденные от описываемого ресурса и встроенные метаданные. В зависимости от контента описываемых ресурсов выделяют независимые и зависимые метаданные. По уровню использования системные и пользовательские метаданные. По частоте изменения метаданные делят на статические и динамические, а по степени формализованности - формализованные и неформализованные метаданные. По содержанию метаданные можно разделить на описание ресурса и описание содержимого ресурса. По отношению к ресурсу можно выделить метаданные ресурса и метаданные, относящиеся к его частям. По возможности логического вывода метаданные можно подразделить на три слоя: нижний, средний и верхний. Нижний слой — это сами данные. Средний слой — метаданные, описывающие данные нижнего уровня. Верхний слой — метаданные, которые позволяют делать логический вывод, используя второй слой. Это позволяет поддерживать многоуровневость метаданных.

По функциям можно выделить описательные, структурные и административные метаданные. Административные метаданные служат для управления и администрирования информационными ресурсами, хранят информацию о датах создания и обновления ресурса, о владельце ресурса, местоположении, правах и условиях доступа.

Описательные метаданные используются для идентификации и описания информационных ресурсов, его контента и содержат информацию для помощи при поиске, описание версий ресурсов, специализированные индексы, аннотации создателей и пользователей. Как вариант описательных

метаданных можно привести набор значений элементов метаданных Дублинского ядра.

Структурные метаданные характеризуют общую структуру ресурса, его компоненты и взаимодействие между ними.

Метаданные выполняют разнообразные функции в системах, их использующих. Функции и состав метаданных зависят от информационных технологий, на которых базируется система, от ее функциональности, свойств поддерживаемых в ней информационных ресурсов, способов их организации в системе, особенностей задач их обработки и от многих других факторов[3].

Функции у метаданных довольно много, так как видов метаданных большое количество. Функции представлены на карте знаний на рисунке 2.

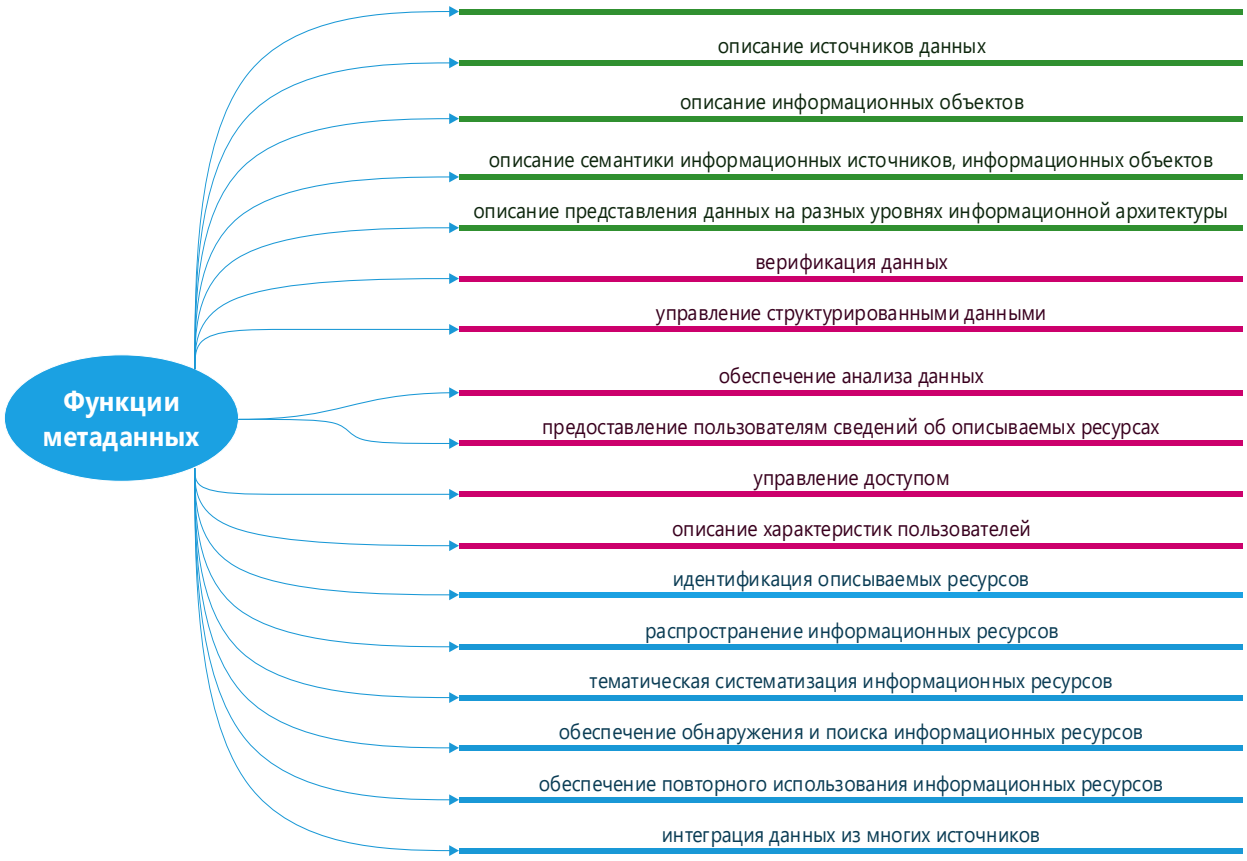


Рисунок 2 – Функции метаданных.

Важнейшая функция метаданных - описание абстрактных моделей предметной области, ее семантики встроенным внешним способом в информационных системах.

1.2 Обзор систем, использующих метаданные

К информационным системам нового поколения предъявляются и новые требования в сфере открытой архитектуры, динамичности, стандартизации. Необходима возможность внесения изменений и расширения функциональности в короткие сроки без остановки работы системы с использованием высокоуровневого объектного интерфейса, с накоплением информации об обновлениях версий прикладной системы (например, в виде знаний. Открытая архитектура позволяет осуществлять локальную и глобальную интеграцию через Internet, а прогрессивная технология разработки позволяет создавать новые версии систем в более короткие сроки меньшим числом сотрудников.

Рассмотренные требования не поддерживаются системами использующие для хранения данных только традиционные методы систем управления базами данных, так как структура систем при таком подходе статична и при необходимости изменить или дополнить ее необходимо вмешательство разработчиков в основной код системы.

Как пример первых больших систем, основанных на метаданных можно привести систему METAS. Она предназначена для создания динамически настраиваемых информационных систем, с помощью многоуровневой модели и использования технологии взаимосвязанных данных, представляя информационную систему на разных уровнях абстракции[5].

Описание предметной области с помощью данной технологии используется во время работы системы, функции обработки и представления данных работают по принципу интерпретации правил для метаданных.

Системы, оснащенные технологией METAS, способны изменять структуру приложения во время работы с ним, изменения происходят сразу же

после их внесения, и они отображаются в метаданных, в информационных объектах, графическом интерфейсе, функционале системы.

В METAS можно выделить три основных слоя:

– презентационная модель для описания графического интерфейса;

– логическая модель для сущностей и их поведения в рамках предметной области;

– физическая модель для представления объектов системы в базе данных.

Преимущества такой организации системы обеспечивает возможность изменения структуры и функционала системы с минимальными трудозатратами, что влечет за собой динамическую адаптацию системы в процессе работы с системой. Архитектура системы приведена на рисунке 3.

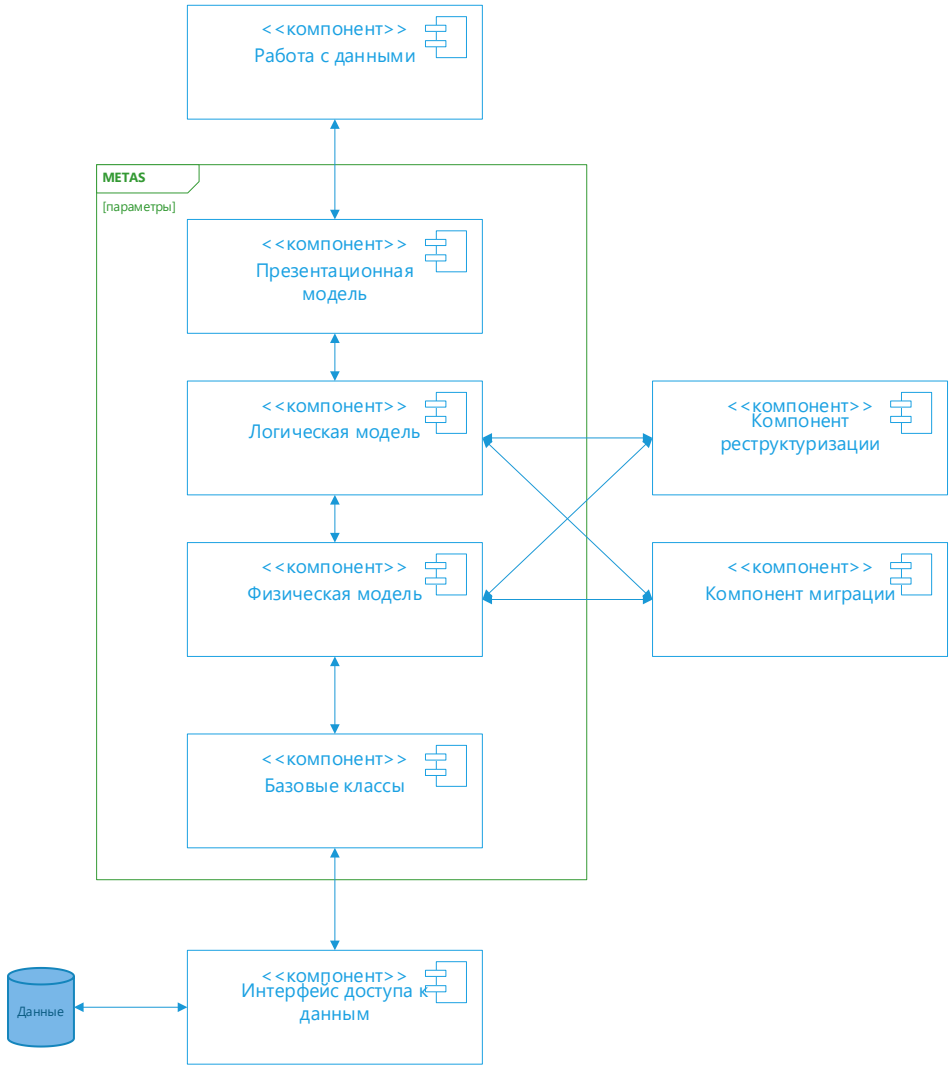


Рисунок 3 - Модели метаданных и компоненты METAS

Преимущества такой организации системы обеспечивает возможность изменения структуры и функционала системы с минимальными трудозатратами, что влечет за собой динамическую адаптацию системы в процессе работы с системой.

Определенную популярность завоевала система Business Objects одноименной компании, которая предоставляет простой в использовании и развитый интерфейс для анализа бизнес-информации.

Система Versata предоставляет набор функциональности писаниях классов и бизнес-правил, позволяя, в частности, импортировать диаграммы классов из Rational Rose, отображая объекты в классы Java. В системе нет описания динамического поведения объектов.

Система «Мегаполис 2» компании «Софтлайн» применяет для хранения данных и метаданных реляционную систему управления базами данных, а также есть возможность использование объектного подхода. Компания создает информационные системы для крупных и средних предприятий. Ядро системы обеспечивает возможность расширения как структуры и состава этих объектов, так и логики поведения.

На разработку таких систем, на предоставления серверов и сервисов данных компаний требуется большие финансовые затраты. Применение подобных решений сокращает затраты на обслуживание и модификацию систем, позволяя системам с выделенными метаданными развиваться, модифицироваться, подстраиваться под требования уже на этапе внедрения и сопровождения.

Недостатком применение сторонних серверов можно назвать изолированность настраиваемых систем от программного обеспечения, уже работающего на предприятии, что ведет к осложнению обмена информацией, в том числе, и метаданными внутри гетерогенного развернутого комплекса программных средств.

Так же недостатком можно назвать и полную открытость архитектуры, что не поддерживает некоторые ограничения конфиденциальности данных, необходимость предоставления данных сторонним предприятиям.

1.3 Обобщенная архитектура систем, основанных на семантических метаданных

Метаданные информационной системы влияют на состав, интерфейс и логику работы этой системы. Изменения в метаданных принесет изменения в работе системы. Данное изменение не будет зависеть от уровня встроенности и однородности метаданных. У систем, основанных на метаданных и знаниях, выделяют несколько типов пользователей: пользователи системы разного уровня доступа к данным и эксперты-аналитики, которые получают доступ к метаданным.

Чаще метаданные анализируются и проектируются на этапе анализа требований к информационной системе и ее проектирования. На последнем этапе крайне важной задачей является выбор архитектуры информационной системы и места в ней метаданных.

Компоненты информационной системы по выполняемым функциям можно разделить на три уровня: уровень пользовательского представления, уровень бизнес-логики и уровень доступа к данным.

Уровень представления обеспечивает интерактивное взаимодействие с пользователем. Уровень бизнес-логики - реализация основных действий и функций, реализация алгоритмов реакции приложения на действия пользователя или на внутренние события, правил обработки данных и формирование запросов к данным. Уровень доступа к данным - хранение, выборка, модификация и удаление данных, связанных с решаемой приложением прикладной задачей. Часто на данном уровне можно выделить взаимодействие с метаданными, так как метаданные и данные обладают единой структурой и средствами доступа.

Обобщенная архитектура информационных систем приведена на рисунке 4.

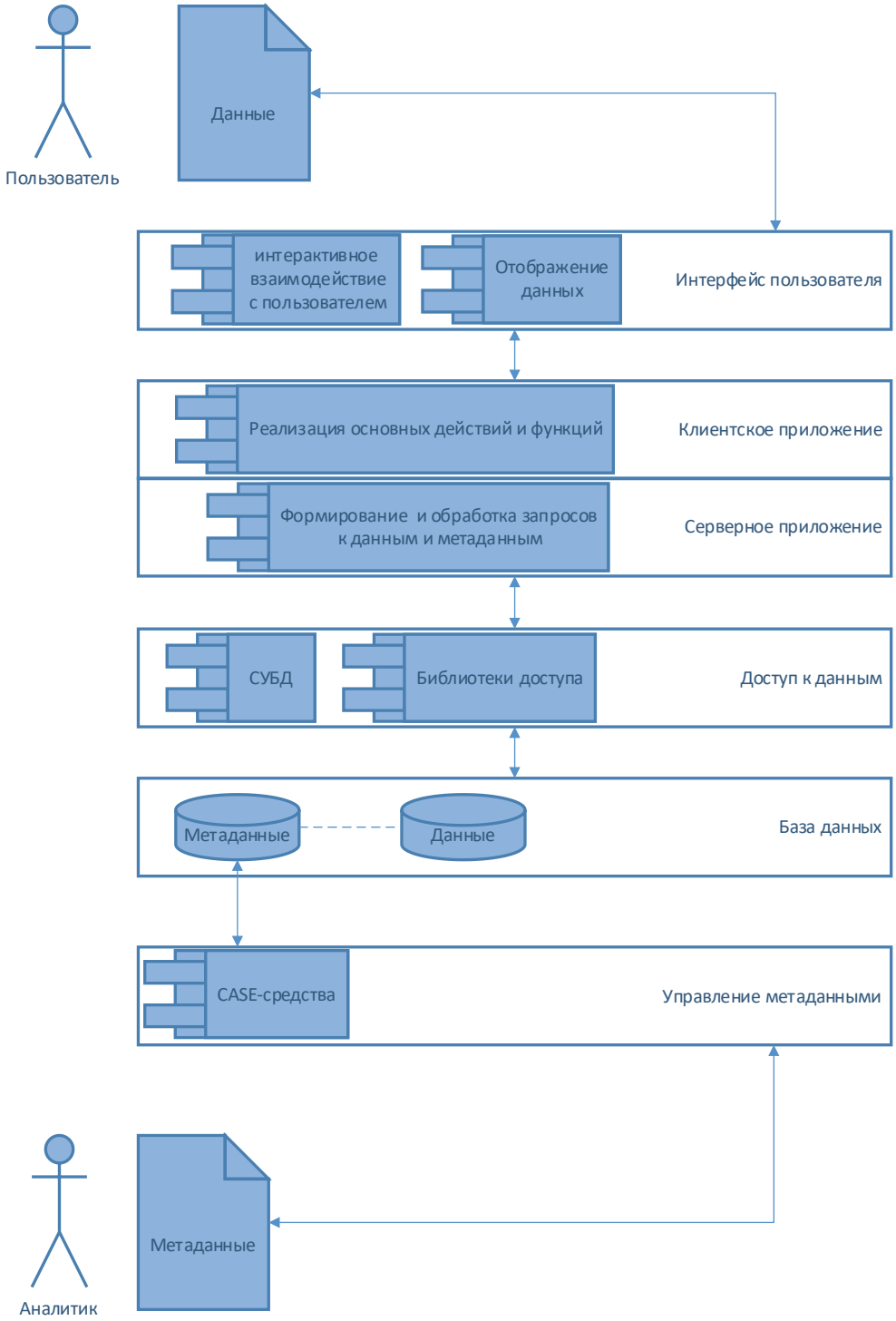


Рисунок 4 – Обобщенная архитектура информационных систем с метаданными

Метаданные могут быть реализованы в различных форматах: это могут быть и файлы xml или базы данных различных СУБД, но в современных вариантах архитектур они выделены вне кода приложений.

На рисунке приведена обобщенная клиент-серверная вариация архитектуры. Архитектура проекта может быть отличной от выбранной. Кроме того, выбранная архитектура определит организацию программной системы и ее структурные элементы, поведения этих элементов, архитектурный стиль, определяющий логическую и физическую организацию системы: статические и динамические элементы, их интерфейсы и способы их объединения, а так же правила ее использования и интеграции с другими системами, функциональность, производительность, гибкость, надежность, возможность повторного применения, полноту, экономические и технологические ограничения, а также вопрос пользовательского интерфейса.

Выводы по главе 1

В данной главе рассмотрены актуальность и проблематика метаданных в информационных системах. Описаны классификации и функции метаданных в работе информационных систем. Рассмотрены характерные требования, предъявляемые к метаданным информационных систем.

Приводится обзор информационных систем с метаданным и описана обобщенная архитектура.

2 Представление семантических метаданных в информационных системах.

Данная глава работы посвящена разработке схемы семантических метаданных. Выделены набор объектов семантических метаданных. Предложен алгоритм отображения схемы семантических метаданных в реляционную модель метаданных информационных систем.

2.1 Варианты представления семантических метаданных

Для представления метаданных могут использоваться естественные языки, а также различные искусственные языки, такие как языки разметки, схемы метаданных, онтологии и графические средства.

Варианты представления метаданных в информационных системах представлены на карте знаний на рисунке 4.

Естественные языки не обеспечивают однозначности и строгости интерпретации метаданных, представленных их средствами и не пригодны для компьютерной обработки. Данное представление наиболее важно на этапе анализа предметной области и выделения метаданных.

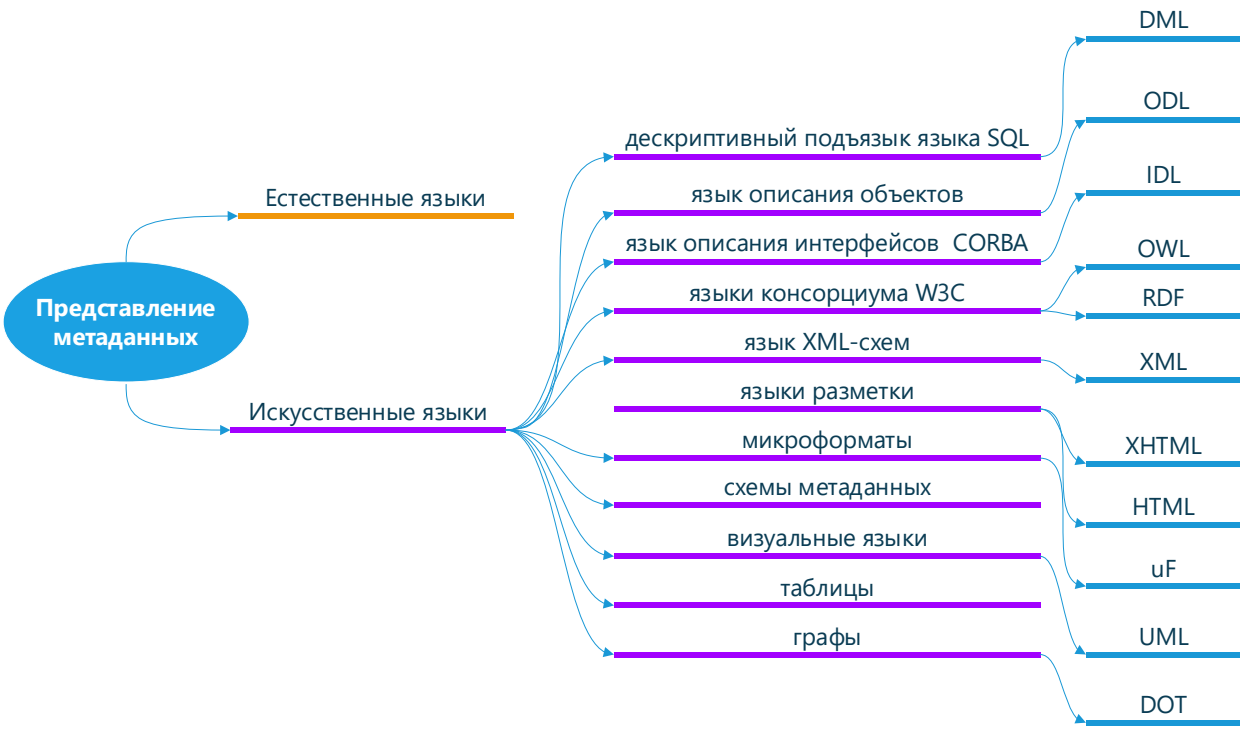


Рисунок 4 – Примеры искусственных языков для описания метаданных

Техническое задание чаще представляет собой описание функции системы, требуемых для реализации. А также включает описание предметной области, основных объектов системы и бизнесс-процессов. На данном этапе для анализа применяются графические представление, например, схемы UML или схемы знаний, на основе которых получают формализованное описание на искусственном языке.

К их числу искусственных языков относят языки описания данных систем управления базами данных, языки концептуального моделирования, описания онтологий, бизнес-процессов, потоков работ.

Онтологии могут быть формализованы языками описания, такими как OWL. OWL2 консорциума W3C стали иногда использоваться для описания концептуальной схемы предметной области.

2.2 Формализация схемы метаданных

Модификация заложенных в систему знаний или повторное их использование в других системах затруднительны в силу того, что концептуализация базы знаний и исходных предположений не выражена явно. В соответствие с приведенным определением онтология должна явно специфицировать концептуализацию какой- либо предметной области.

В информационной системе, основанной на онтологии, семантика объектов должна описываться в терминах онтологии, и, следовательно, необходимы подходы к описанию семантики объектов ИС. Одним из подходов является создание семантических метаданных.

Семантические метаданные - метаданные, описывающие контекст и/или контент объекта в информационной системе с помощью понятий предметной области, определенных на некотором языке описания онтологии.[7]

Семантические метаданные являются одним из подвидов метаописаниях. В литературе выделяют три типа: системные, структурные и

семантические. К системным относят инофрмацию, необходимую для поддержания работоспособности информационной системы на уровне администрирования. К структурным – справочную информацию, категоризацию объектов. Иногда данный тип метаданных так же переходит в различные характеристики контента, то есть семантические метаданные.

Одной из наиболее приближенным к управлению знаниями можно выделить формальную модель Гавриловой Т.А. и Хорошевского. Онтология состоит из терминов, организованных в таксономию, их определений, атрибутов, а также связанных с ними аксиом и правил вывода

Под онтологией понимается упорядоченная тройка вида

$$O = <C, R, F>, \quad (1)$$

где C – конечное множество концептов (понятий, терминов) предметной области, которую представляет онтология O;

R – конечное множество отношений между концептами заданной предметной области;

F – конечное множество функций интерпретации (аксиоматизации), заданных на концептах и/или отношениях онтологий O.

Более сокращенная запись, например, представление тезауруса, включает только два компонента из данной тройки:

$$O = <C, R> \quad (2)$$

Для семантических метаданных, одним из важных представлений является иерархия понятий. Для данной задачи конечное множество отношений между концептами заданной предметной области содержит следующие базовые типы отношений:

- Rg – гипонимия («is_a»,«класс – подкласс»);
- Rm – меронимия («part_of», «часть – целое»);
- Rs – синонимия («synonym_of»).

Для расширения описательных задач онтологии следует учесть и другие функции отношений. Если обратиться к задачам отображения схемы данных, то на основании представления универсальной модели схемы метаданных

можно рассмотреть связи модели Чена «Сущность-связь». Она является одним из распространенных средств спецификации модельных представлений, основанных на анализе семантики данных, то есть семантических моделей.

Базис понятий семантического моделирования в этом случае включает:

– определение объекта-экземпляра;
– определение объекта-типа;
– определение связи между объектами;
– определение свойства объекта;
– определение идентифицирующего свойства объекта.

Для описания структкуры предметной области универсальной моделью требуется выделить основные понятия:

– сущность;
– связь;
– атрибут.

Атрибут, в данном случае, так же является объектом метаданных, который связывается с объектом «Сущность» или «Объект» предметной области функцией «Является атрибутом». Кроме того, в модели «сущность-связь» предусматриваются базовые связи. Связи обладают свойствами:

– тип:
– степень;
– кардинальность;
– внешний ключ сущности.

Степень связи – унарная, бинарная, тернарная, N-арная. Внешний ключ показывает в модели связь между отношениями. А основные типы связи можно разбить на следующие группы:

– категориальная;
– идентифицирующая;
– неидентифицирующая.

Еще одно понятие, которое помогает моделировать семантическое описание – домен. Домен характеризуется следующими свойствами:

– имеет уникальное имя;

– определен на некотором простом (скалярном) типе данных или на другом домене;

– может иметь некоторое логическое условие, позволяющее описать

– подмножество данных, допустимых для данного домена;

– может быть задан перечислением множества допустимых элементов данных;

– несет определенную смысловую нагрузку.

По отношению к атрибутам домены играют роль области определения. Соответствие между доменами и атрибутами примерно такое же, как между типами данных и переменными в языках программирования. Атрибут в такой интерпретации – это предметная переменная, определенная на домене. Домен представляет собой подмножество допустимых элементов данных некоторого типа, имеющее семантически значимое имя. Отличие домена от подмножества данных некоторого типа состоит именно в том, что домен отражает семантику, определенную предметной областью. Может быть несколько доменов, совпадающих как подмножества, но несущие различный смысл.

Для выделения базовых объектов семнатических метаданных можно рассмотреть стандарт Дублинского ядра (Dublin Core) [7]. С помощью данной структуры возможно описание разнотипных объектов. Все атрибуты, составляющие структуру метаданных Дублинского ядра можно разделить на три группы: семантические атрибуты, права собственности и системные атрибуты.

Простой набор элементов метаданных Дублинского ядра (DCMES) состоит из нескольких элементов метаданных: название, создатель, тема, описание, тип, формат, идентификатор, источник, язык, отношения. Рекомендовано поддержание словарей, которые будут содержать необходимый набор. В данном случае, такая структура хорошо отображается в универсальную модель данных с доменами.

В базовую онтологическую модель семантических метаданных необходимо включить два множества: конечное множество концептов и конечное множество отношений между концептами заданной предметной области.

В конечное множество концептов *C* предлагается внести следующие множества:

$$C=\{C_o, C_a, C_v, C_d\},$$

где C_o – множество концептов – объектов предметной области;

C_a – множество атрибутов объектов;

C_v – множество значений;

C_d – множество доменов.

В множество связей между концептами необходимо выделить следующие:

$$R = \{R_g, R_m, R_s, R_d\},$$

где R_g – гипонимия («is_a», «класс – подкласс)

R_m – меронимия «part_of», «часть – целое» ;

Rs – является атрибутом, описывает объект;

Rd – является доменом;

Rf – функциональная связь.

Каждый объект описывается предикатным триплетом

$$<C_i, R_k, C_j>,$$

где C_i, R_k, C_j - объекты предметной области,

R_k - связь между ними.

В набор необходимых объектов семантических метаданных предметной области на основе данной модели можно включить**:** Объект, Атрибут, Значение, Тип, Описание, Отношение, Ограничения. Данная модель является базовой универсальной моделью данных, и подобна моделям, используемым для описания реляционных отношений в системах управления базами данных. Данную модель можно дополнить семантикой бизнес-процессов предметной

области, описанием ресурсов, задействованных в них, и пользователей с разграничением доступа к ним.

2.3 Варианты отображения метаданных в реляционной модели.

Проектирование базы метаданных, также как и базы данных, состоит из нескольких этапов: инфологическое, логическое, физическое проектирование. В статье «Отображение метаданных в реляционную модель данных» на этапе логического проектирования предлагалось использовать два подхода: предметно-ориентированный и универсальный [14].

При использовании предметно-ориентированной модели сущности предметной области - метаданных, выделенные на этапе концептуального проектирования - отображаются в соответствующие отношения, а атрибуты сущностей – в атрибуты отношений. В другом варианте сущности предметной области будут отображаться в кортежи отношения Отношение, атрибуты сущностей – в кортежи Атрибут.

В пункте 1.1 были описаны многоуровневые метаданные, которые имеют три уровня: нижний, средний и верхний. Нижний слой — это сами данные. Средний слой — метаданные, описывающие данные нижнего уровня. Верхний слой — метаданные, которые позволяют делать логический вывод, используя второй слой. Уровни показаны на рисунке 5.

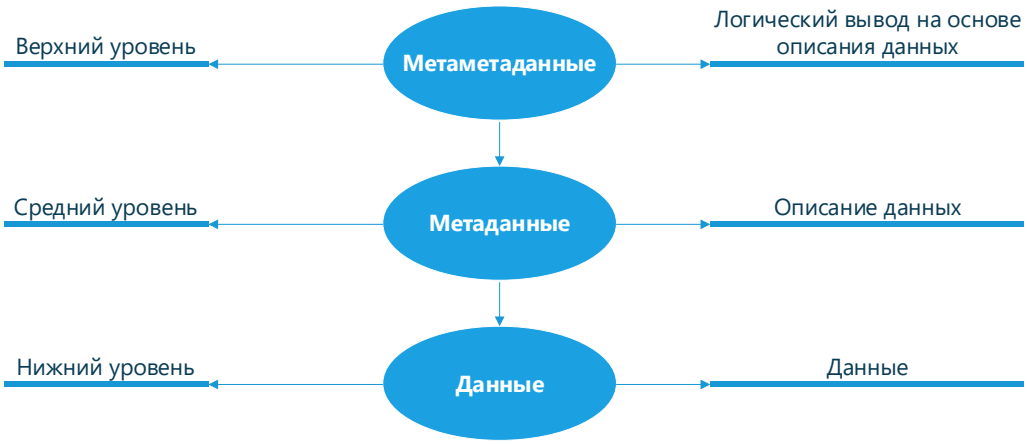


Рисунок 5 – Многоуровневость метаданных

В статье «Отображение метаданных в реляционную модель» [15] Дзюба Е.А., Шибанов С.В., Хмелевской Б.Г., Кочегаров И.И., Гришко А.К. были рассмотрены варианты отображения модели метаданных в реляционную модель:

– - предметно-ориентированная модель данных
– универсальная модель данных.

При внесении изменений в логическую модель универсальный подход имеет неоспоримое преимущество, но при этом можно потерять привычную естественность запросов к базе данных, которую имеет предметно-ориентированная модель. Но для универсальной модели полнотекстовый поиск программируется и выполняется быстрее.

Две модели хранения данных в равной степени применимы и к реляционным, и к объектным базам данных. Выбор между предметно-ориентированной и универсальной моделями данных является весьма сложной и нетривиальной задачей, которая зависит от множества факторов простота составления и зачастую может быть окончательно решена только экспериментальным путем.

Если рассматривать уровни метаданных при отображении в реляционную или объектно-ориентированную модель, то необходимо выбрать предметно-ориентированную или универсальную модель. Модели и варианты отображения приведены в таблице 1.

Таблица 1- Отображение уровней метаданных в различных моделях

Уровни онтологии	Предметно-ориентированная модель	Универсальная модель
Метаданные	Метаданные	Метаметаданные
Данные	Данные	Метаданные

Рассмотрим часто используемый алгоритм отображения схемы семантических метаданных в предметно-ориентированную модель. Таким образом, на выходе можно получить схему описания метаданных среднего уровня. Алгоритм представлен в таблице 2.

Таблица 2- Алгоритм отображение онтологической модели в предметно-ориентированную реляционную модель

Этапы	Объекты семантических метаданных	Предметно-ориентированная модель метаданных (реляционная модель)
Этап 1 Определение объектов семантических метаданных	Объект,C_o	Отношение реляционной модели метаданных
Этап 2 Определение атрибутов объектов	Атрибут C_a,	Атрибут отношений реляционной модели метаданных
Этап 3 Определение ограничений атрибутов	Тип	Тип атрибутов отношений реляционной модели метаданных
	Описание	Описание типа атрибута
	Ограничение C_d	Допустимый диапазон значений атрибутов
Этап 4 Определение связей между объектами	Связь между объектами «Is-a», R_g	Иерархическая связь в Отношении между Объектами. Добавляется Атрибут с идентификатором Объекта, к которому применяется иерархия

Продожение таблицы 2

Этапы	Объекты семантических метаданных	Предметно-ориентированная модель метаданных (реляционная модель)
	Связь между объектами «part_of», R_m	Иерархическая связь в Отношении между Объектами. Добавляется Атрибут с идентификатором Объекта, к которому применяется иерархия
	Связь между Объектом и Атрибутом	Учитывается на этапе 2 при составлении списка атрибутов
	*Связь «*является доменом» *,Rd*;	Объект домена описывается отдельной сущностью, а экземпляры этого объекта – допустимое множество значений соотвествующего поля
	Функциональная связь., *Rf*	Неидентифицирующая связь между объектами
Этап 5 Определение семантических данных	Описание объектов, атрибутов, типов, связей и значения	Кортежи в соответствующих отношениях

Для универсальной модели реляционной базы метаданных этапы остаются аналогичными, меняется отображение составных частей онтологии в модель. В таблице 3 приведен алгоритм отображение онтологической модели в универсальную реляционную модель.

Таблица 3- Алгоритм отображение онтологической модели в предметно-ориентированную реляционную модель

Этапы	Объекты семантических метаданных	Предметно-ориентированная модель метаданных (реляционная модель)
Этап 1 Определение объектов семантических метаданных	Объект, C_o	Кортеж в отношении Объект
Этап 2 Определение атрибутов объектов	Атрибут	Данные в таблице Атрибут, связанные с записями из таблицы Объект реляционной модели метаданных
Этап 3 Определение ограничений атрибутов	Тип	Тип атрибутов отношений указывается атрибутом в таблице Атрибут
	Описание	атрибутом в таблице Атрибут
	Ограничение	Кортеж в отношении Домен
Этап 4 Определение связей между объектами	Связь между объектами «Is-a», R_g	Иерархическая связь указывается в этом же отношении на другой объект
	Связь между объектами «part_of», R_m	Связь указывается в этом же отношении на другой объект
	Связь между Объектом и Атрибутом	Идентифицирующая связь между отношениями Объект и Атрибут

Продолжение таблицы 3

Этапы	Объекты семантических метаданных	Предметно-ориентированная модель метаданных (реляционная модель)
	*Связь «*является доменом» ,*Rd*;	Идентифицирующая связь между отношениями Атрибут и Домен
	Функциональная связь., *Rf*	Неидентифицирующая связь между объектами внутри отношения
Этап 5 Определение семантических данных	Описание объектов, атрибутов, типов, связей	Кортежи в соответствующих отношениях

Пример представления предметно ориентированной модели приведен на рисунке 6. Универсальная модель отображения приведена на рисунке 7.

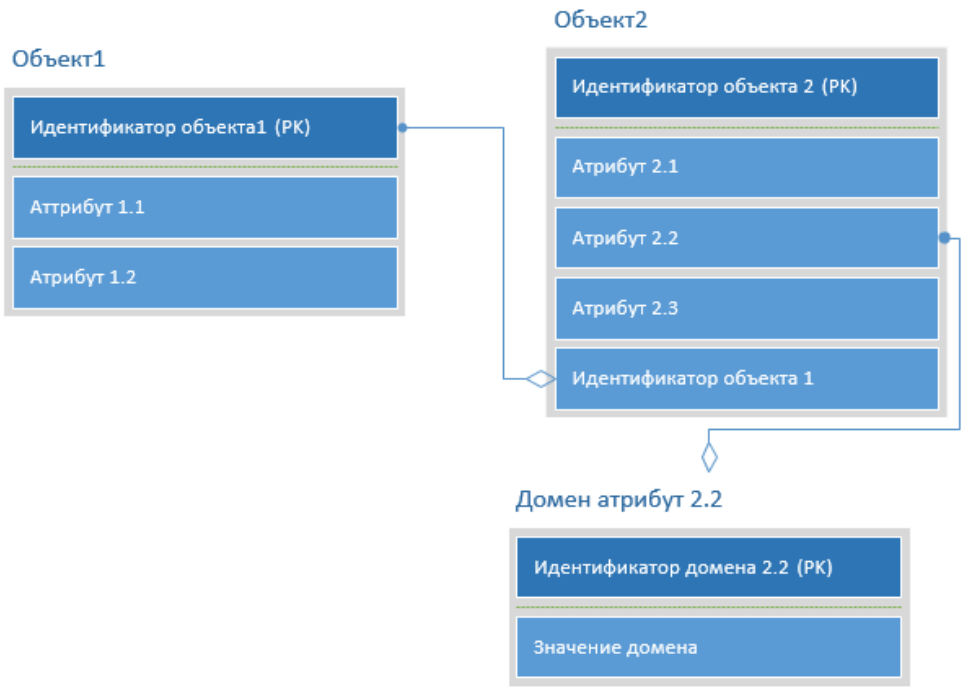


Рисунок 6 – Представление объектов в предметно-ориентированной модели

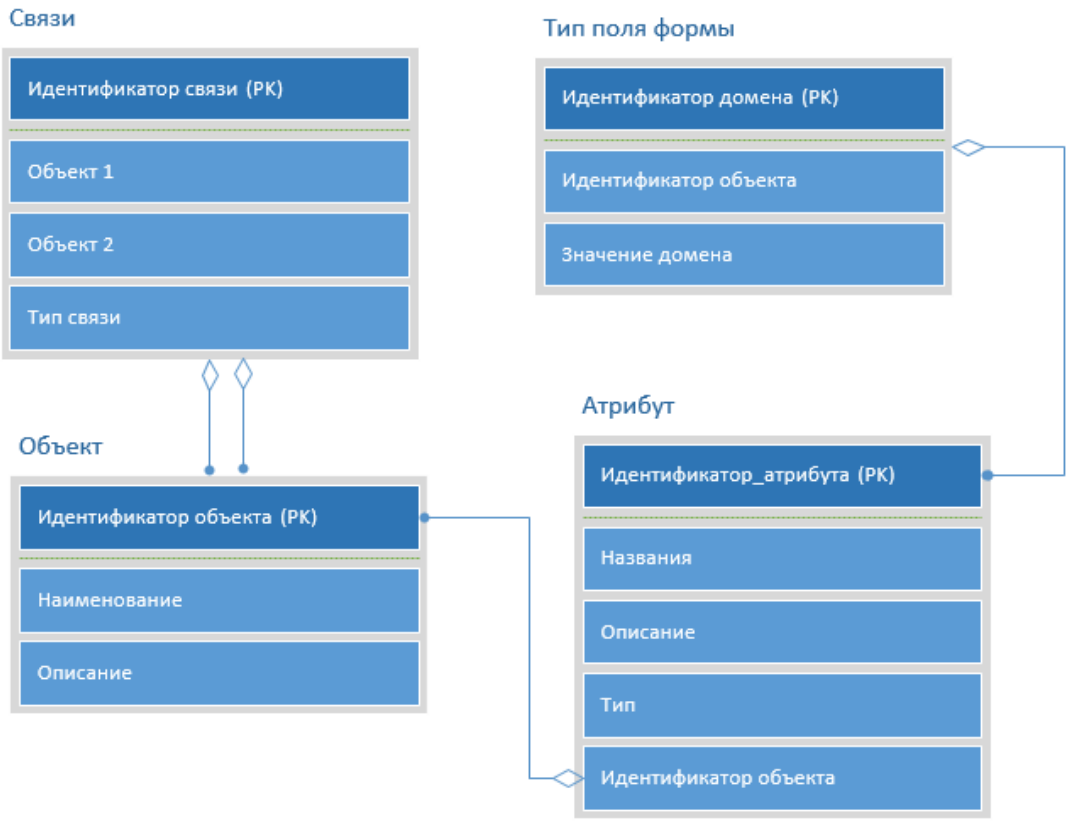


Рисунок 7 – Представление объектов в универсальной модели

Для повышения уровня абстракции можно использовать универсальную модель. Это приведет к получению верхнего уровня семантических метаданных, метаметаданных, которые позволят менять состав семантических метаданных, не изменяя работу системы. Получать более высокий уровень семантических метаданных используя не считается целесообразным только для семантических метаданных предметной области. Если рассматривать саму информационную систему как предметную область, то семантические метаданные при такой абстракции кроме описания предметной области, в которой разрабатывается система, будут содержать описание самой системы в целом.

Уровни абстракций метаописаний приведены на рисунке 8.

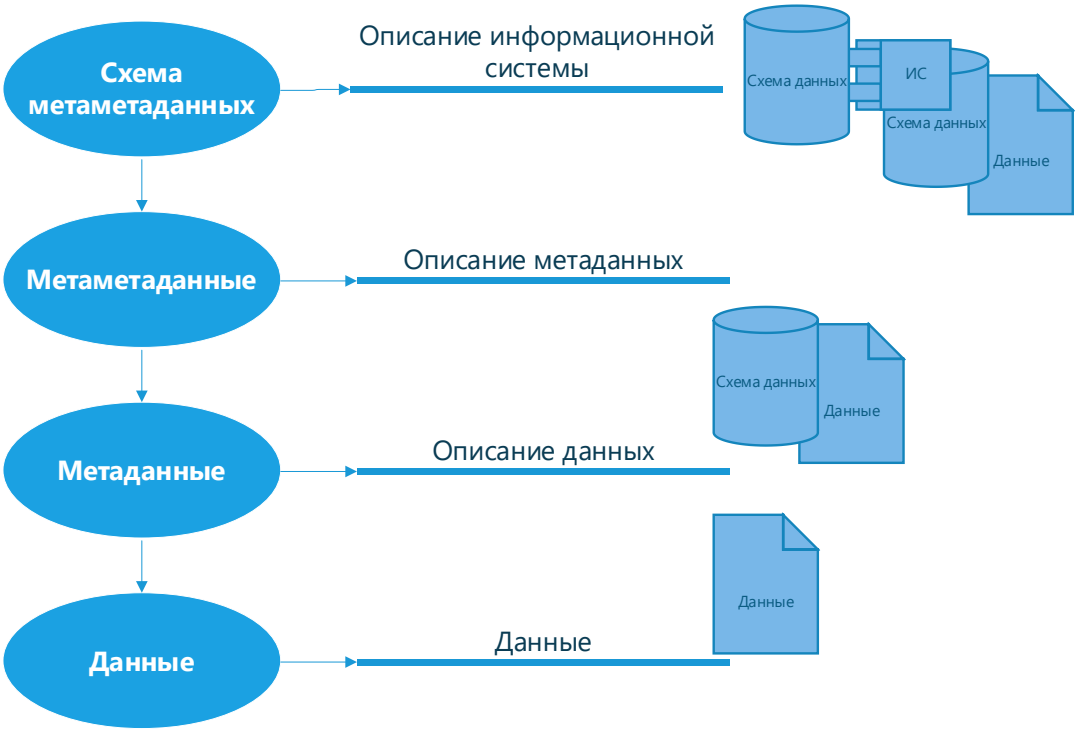


Рисунок 8 – Четвертый уровень метаданных

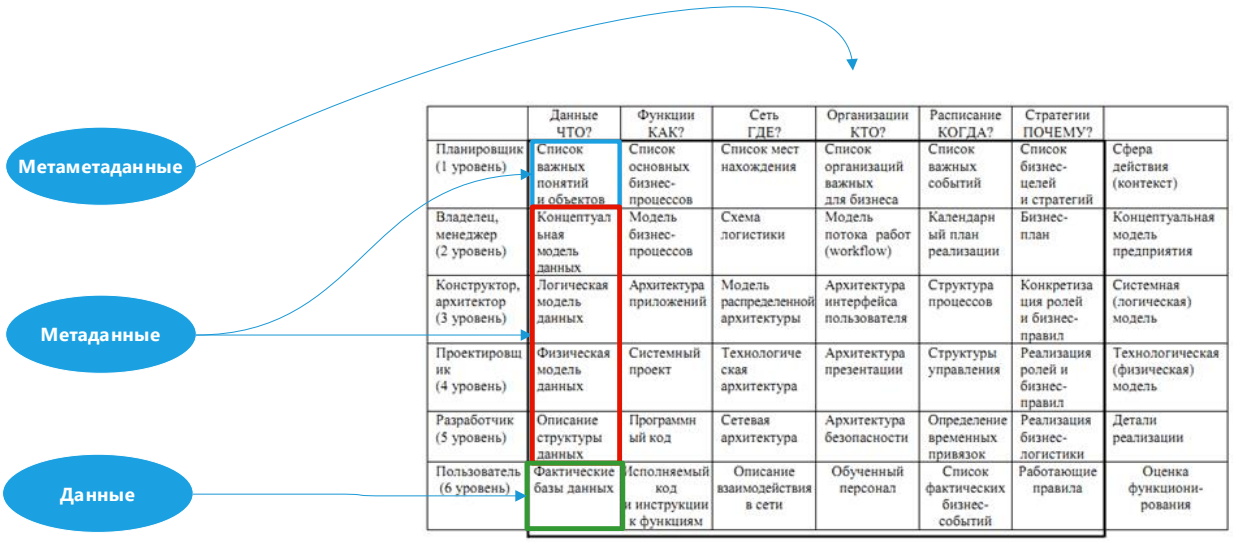
В качестве схемы метаданных можно использовать архитектурную схему Захмана, которая позволяет описать информационную систему или комплекс программных средств на разных уровнях и с разными видами метаданных. Табличный вариант архитектурного шаблона Захмана приведен на рисунке 9.

	Данные ЧТО?	Функции КАК?	Сеть ГДЕ?	Организации КТО?	Расписание КОГДА?	Стратегии ПОЧЕМУ?	
Планировщик (1 уровень)	Список важных понятий и объектов	Список основных бизнес-процессов	Список мест нахождения	Список организаций важных для бизнеса	Список важных событий	Список бизнес-целей и стратегий	Сфера действия (контекст)
Владелец, менеджер (2 уровень)	Концептуальная модель данных	Модель бизнес-процессов	Схема логистики	Модель потока работ (workflow)	Календарный план реализации	Бизнес-план	Концептуальная модель предприятия
Конструктор, архитектор (3 уровень)	Логическая модель данных	Архитектура приложений	Модель распределенной архитектуры	Архитектура интерфейса пользователя	Структура процессов	Конкретизация ролей и бизнес-правил	Системная (логическая) модель
Проектировщик (4 уровень)	Физическая модель данных	Системный проект	Технологическая архитектура	Архитектура презентации	Структуры управления	Реализация ролей и бизнес-правил	Технологическая (физическая) модель
Разработчик (5 уровень)	Описание структуры данных	Программный код	Сетевая архитектура	Архитектура безопасности	Определение временных привязок	Реализация бизнес-логистики	Детали реализации
Пользователь (6 уровень)	Фактические базы данных	Исполняемый код и инструкции к функциям	Описание взаимодействия в сети	Обученный персонал	Список фактических бизнес-событий	Работающие правила	Оценка функционирования

Рисунок 9 - Схема Захмана.

Модель Захмана основана на дисциплине классической архитектуры и обеспечивает общий словарь и набор перспектив, или структур, для описания современных сложных корпоративных систем.

Основная идея заключается в том, чтобы обеспечить возможность последовательного описания каждого отдельного аспекта системы в координации со всеми остальными. Для любой достаточно сложной системы общее число связей, условий и правил обычно превосходит возможности для одновременного рассмотрения. В то же время, отдельное рассмотрение каждого аспекта системы чаще всего приводит к неоптимальным решениям, как в плане производительности, так и стоимости реализации.



	Данные ЧТО?	Функции КАК?	Сеть ГДЕ?	Организации КТО?	Расписание КОГДА?	Стратегии ПОЧЕМУ?	
Планировщик (1 уровень)	Список важных понятий и объектов	Список основных бизнес-процессов	Список мест нахождения	Список организаций важных для бизнеса	Список важных событий	Список бизнес-целей и стратегий	Сфера действия (контекст)
Владелец, менеджер (2 уровень)	Концептуальная модель данных	Модель бизнес-процессов	Схема логистики	Модель потока работ (workflow)	Календарный план реализации	Бизнес-план	Концептуальная модель предприятия
Конструктор, архитектор (3 уровень)	Логическая модель данных	Архитектура приложений	Модель распределенной архитектуры	Архитектура интерфейса пользователя	Структура процессов	Конкретизация ролей и бизнес-правил	Системная (логическая) модель
Проектировщик (4 уровень)	Физическая модель данных	Системный проект	Технологическая архитектура	Архитектура презентации	Структуры управления	Реализация ролей и бизнес-правил	Технологическая (физическая) модель
Разработчик (5 уровень)	Описание структуры данных	Программный код	Сетевая архитектура	Архитектура безопасности	Определение временных привязок	Реализация бизнес-логистики	Детали реализации
Пользователь (6 уровень)	Фактические базы данных	Исполняемый код и инструкции к функциям	Описание взаимодействия в сети	Обученный персонал	Список фактических бизнес-событий	Работающие правила	Оценка функционирования

Рисунок 10 – Выделение схемы метаданных на Схеме Захмана

Первый столбец схемы, отвечающий на вопрос «Что?» представляет собой жизненный цикл разработки схемы данных или схемы метаданных, в зависимости от принятого уровня абстракции. Фактические данные находятся на пользовательском уровне, а сама схема является метаметаданными всей информационной системы вцелом.высокого уровня абстракции дает следующие преимущества:

– уменьшение структурной и семантической сложности проекта;
– уменьшение связности компонент;

– применение гетерогенных технологий или моделей на разных уронях.

Онтологии верхнего уровня описывают, общее знание о моделируемом мире, формируя общую для онтологий нижних уровней систему понятий. В случае применения Схемы Захмана как онтологии верхнего уровня, архитектурный шаблон станет общим концептуальным словарем для описания информационной системы вцелом.

Выводы по главе 2

В данной главе были рассмотрены варианты описания метаданных и их формализации. Выделен набор объектов семантических метаданных. Предложен алгоритм отображения схемы семантических метаданных в реляционную модель метаданных информационных систем

3 Представление семантических метаданных в подсистеме «Формы»

В ходе выполнения работы было составлено техническое задание на разработку подсистемы «Формы» платформы «Эксперимент», сформулированы функциональные требования к разрабатываемому программному обеспечению, требования к архитектуре, проанализированы существующие подходы к представлению метаданных, а также выбраны технологии и средства для реализации доступа к ним. В данной главе рассмотрены вопросы практической реализации представления семантических метаданных. Рассмотрены вопросы состава онтологического описания предметной области и его отображения в реляционную базу метаданных.

3.1. Назначение и функциональные возможности подсистемы «Формы»

Рассмотрим представления семантических метаданных на примере подсистемы «Формы» образовательной платформы «Эксперимент».

Подсистема отчетов позволяет руководящим должностным ролям запрашивать у подчиненных результаты проведенной работы за какой-то определенный период времени. Платформа позволяет участникам воспользоваться функцией автоматической отправки запросов на заполнение отчётности и отправки готовых документов по электронной почте. Составление структуры нового отчета начинается с указания организаций, к которым будет отправлен запрос на его заполнение. Можно выбрать несколько организаций по типу, либо указать набор конкретных учреждений.

Далее следует указать роли пользователей в выбранных организациях, которые получат уведомление о необходимости заполнения отчета до определенной даты.

Платформа предоставляет пользователям широкий набор инструментов для упрощения выполнения работы с отчётной документацией. Так, каждый

отчет содержит подробное описание, которое помогает пользователям при его заполнении.

При этом отчётность является полностью настраиваемой с возможностью построения на основе базы данных. Функционал отчётов позволяет отслеживать историю обучения, предоставляет информацию об определённых пробелах в знаниях, отражает прогресс и тенденции в освоении образовательных программ. Полученные отчёты для удобства пользователей доступны в файле.

После того как пользователи прислали заполненные отчеты, автор отчета имеет возможность просмотра. Также предусмотрен анализ групп отчетов по организациям, что упрощает анализ данных и сокращает затраченное на него время.

Данная функция позволяет за короткое время получить информацию о деятельности подчиненных учреждений. Диаграмма вариантов использования приведена на рисунке

3.1 Построение схемы семантических метаданных.

Необходимо реализовать инструмент, который строил бы формы и опросы по заданным пользователем критериям и вопросам. Иногда необходимо в школе устроить опрос, кто хочет присоединиться к мероприятию, или какие книги прочитаны учеником за лето. Часто опросов недостаточно, тогда необходимо добавить формы и отчеты. Устроены они одинаково – из полей. Поле имеет название, тип данных и описание. Если необходимо, то поле хранит еще пользователя, который это поле заполнил.

Из технического задания стоит выделить следующие объекты предметной области: Пользователь, Форма, Поле, Тип.

В концептуальной модели это можно отобразить объектами данной предметной области.

Для моделирования применим графическое представление онтологии, где выделим условные обозначения, соответсвующие составу онтологии

семантических метаданных, описанных в главе 2. Обозначения приведены на рисунке 11.

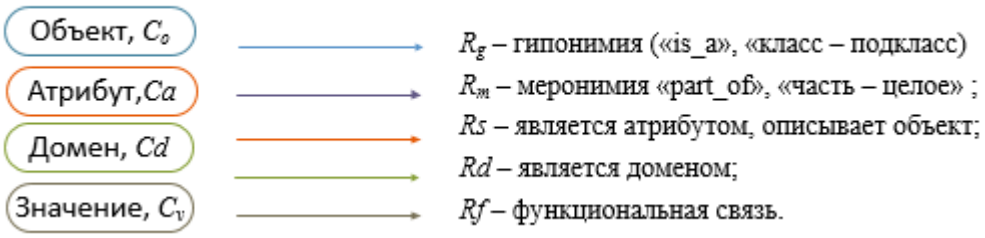


Рисунок 11 – Обозначения в предметной онтологии семантических метаданных.

Онтология «Форма» в графической нотации с выделенными концептами приведена на рисунке 12.

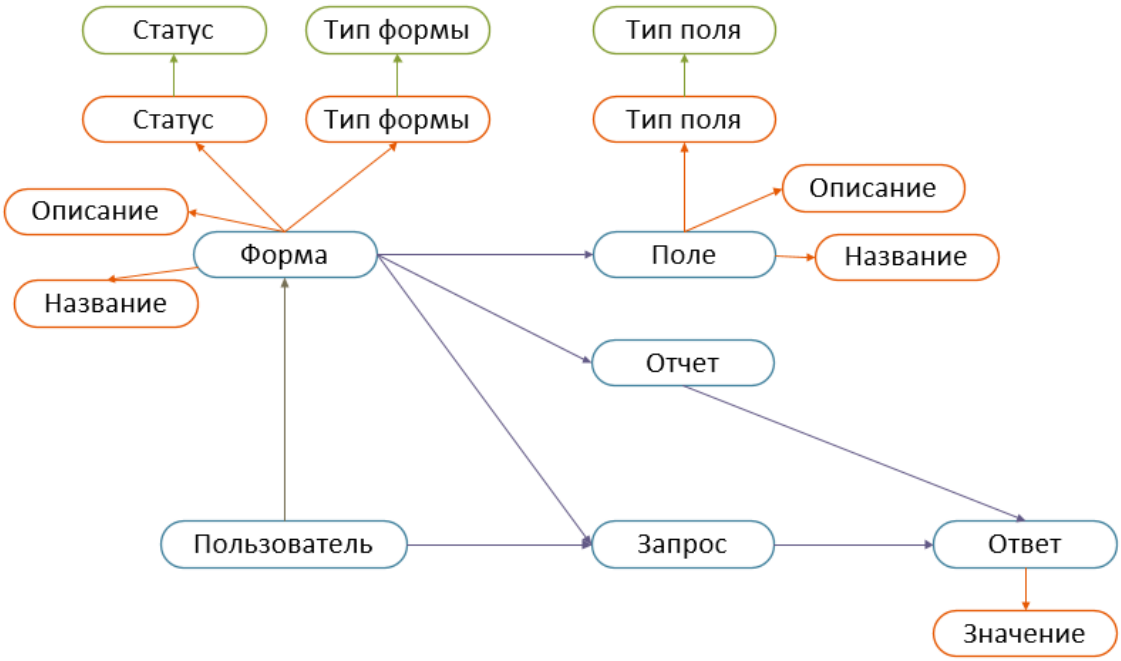


Рисунок 12 – Семантическая онтология предметной области Форма

Иерархия концептов в виде дерева и графическое представление редакторе Protégé приведено на рисунке 13.

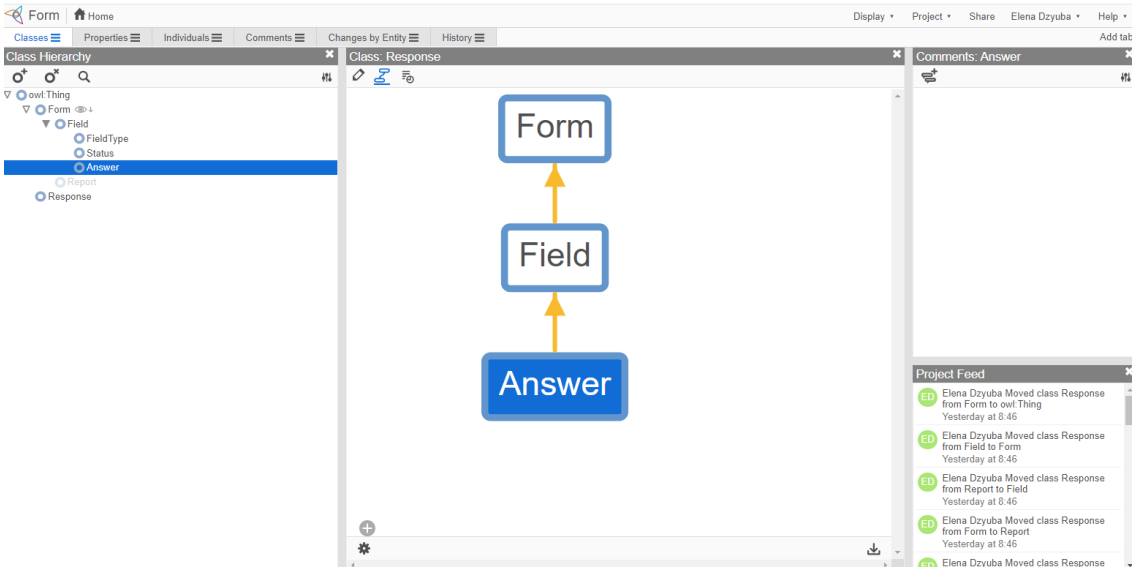


Рисунок 13 – Онтология предметной области в редакторе Protégé

Определим конечное множество концептов C предлагается внести следующим образом:

C_o = {*Форма, Поле, Отчет, Пользователь, Запрос, Ответ*}

C_a = {*Название формы, Описание формы, Статус формы, Тип формы, Тип поля, Описание поля, Название поля, Значение*}

C_v ={}

C_d – {*Статус формы, Тип формы, Тип поля*}

Каждый объект можно описать предикатным триплетом

$$<C_i, R_k, C_j>,$$

где C_i, R_k, C_j - объекты предметной области,

R_k - связь между ними.

Например, связи из множества R_m «Часть - целое» можно выделить следующие:

<Форма, состоит из, Поле>

<Форма, определяется в, Запрос>

<Форма, определяется в, Отчет>

<Пользователь, определяется в, Форма>

<Пользователь, определяется в, Запрос>

<Запрос, определяется в, Ответ>

Для отображения данных была выбрана предметно-ориентированную модель, для того чтобы сохранить семантическое отображение запросов. Так же стоит заметить, что данная предметно-ориентированная модель метаданных будет совпадать с универсальной моделью данных предметной области. Так как рассматриваются семантические метаданные, то есть информация об составе данных, то описывается именно эта информация, что совпадает с содержанием универсальной модели.

При использовании предметно-ориентированных вариантов отображения, объекты преобразуются в отношения. Тем самым мы получаем концептуальную модель базы метаданных.

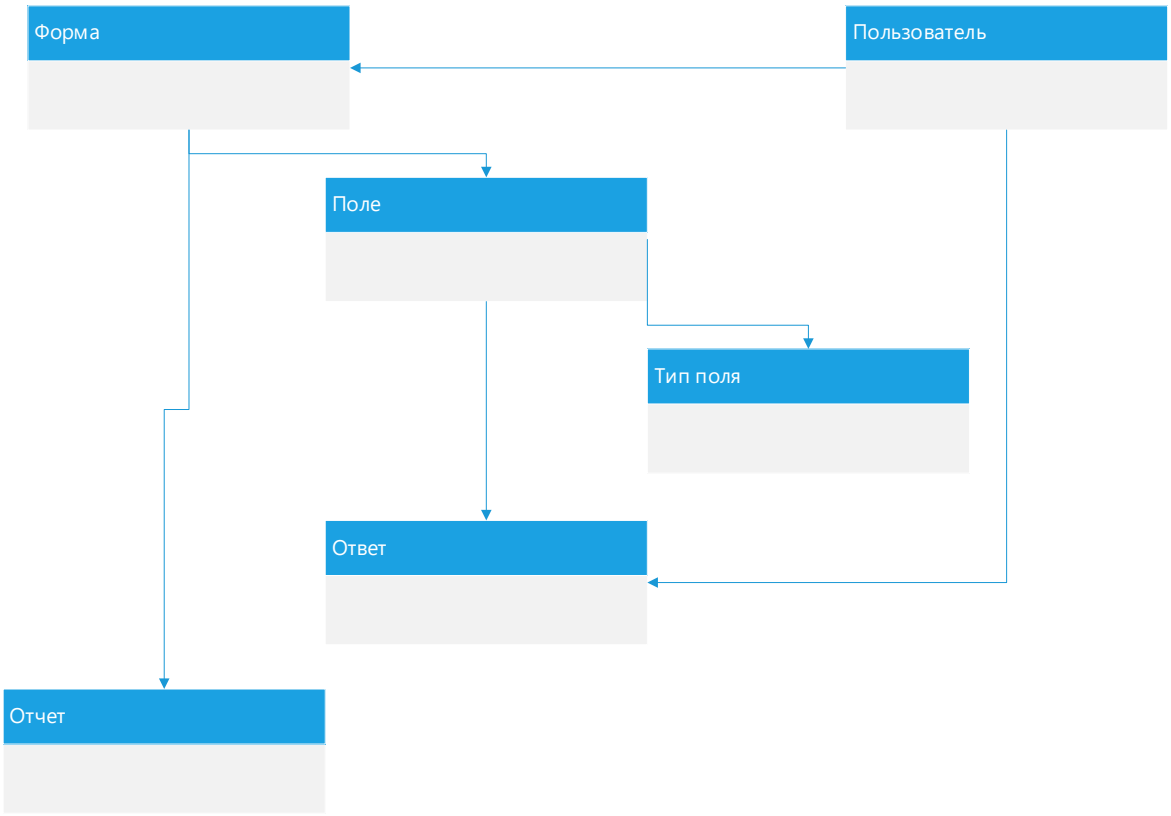


Рисунок 15– Концептуальная модель метаданных

Уточняем атрибуты отношений. Ориентированные связи преображаются в связи «многие ко многим». Дальнейшие проектирование базы данных нужно вести с учетом окружения Эксперимент.

3.2 Проектирование БД

Определяем атрибуты отношений, необходимые для хранения информации об объектах предметной области. Логическая модель в нотации IDEF1X, построенная в Microsoft Visio приведена на рисунке 15.

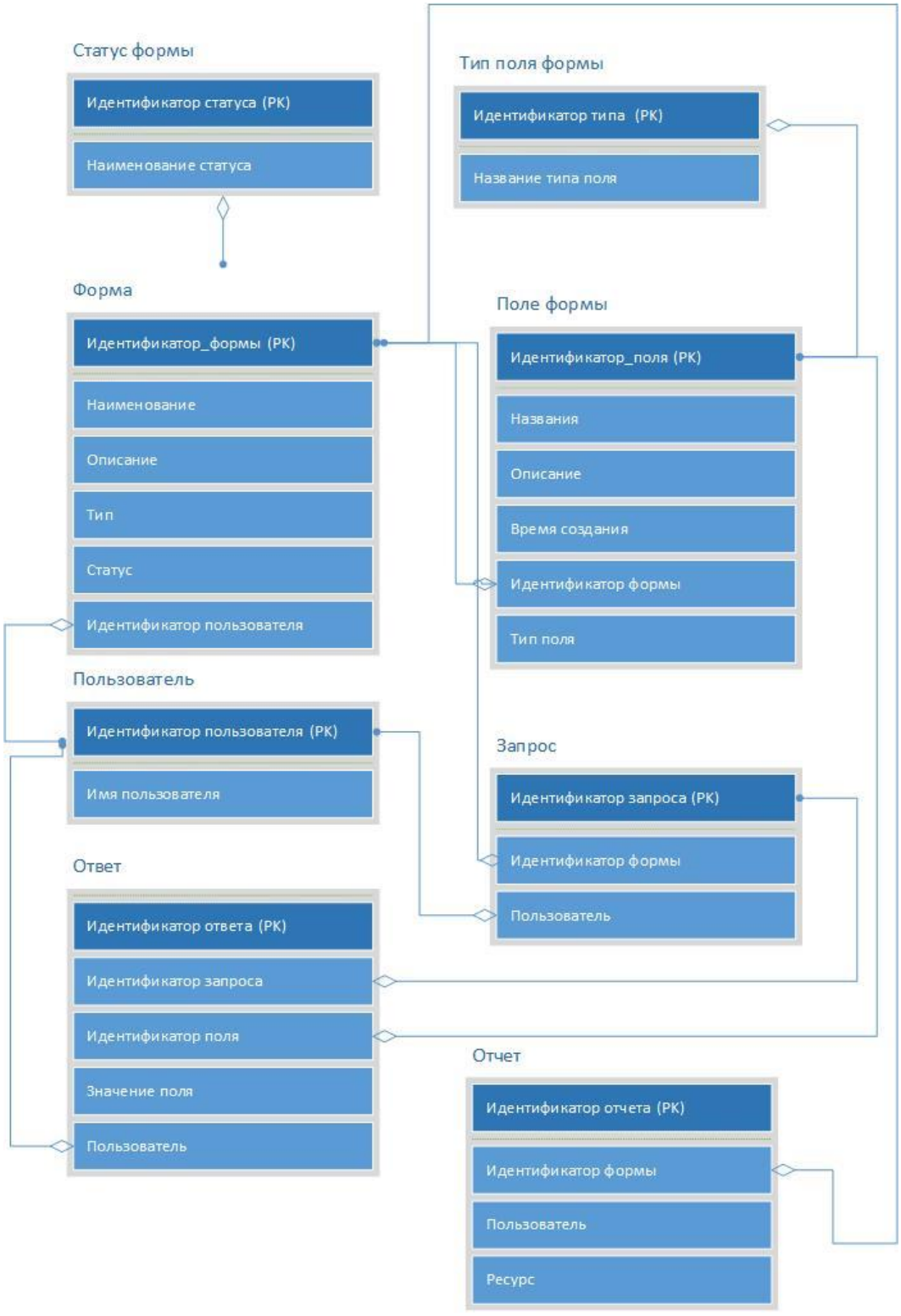


Рисунок 15 – Логическая модель базы метаданных

В таблице 4 приведены полученные сущности после отображения объектов семантических метаданных в предметно-ориентированную модель. Данные объекты получены из множества концептов C_o — объектов предметной области. Это является первым этапом алгоритма отображения семантических метаданных в предметно-ориентированную модель релаяционной базы метаданных (таблица 3)

Таблица 4 – Сущности реляционной модели

№	Название таблицы	Описание
1	Форма	Описание формы запроса
2	Поле формы	Все поля формы
3	Тип поля формы	Справочная таблица-домен, содержащая основные типы полей, не зависимые от физического представления конкретных СУБД
4	Статус формы	Справочная таблица-домен, хранящая возможные статусы форм
5	Пользователь	Информация о зарегистрированном пользователе
6	Запрос	Информация о отправке запроса по выбранной форме пользователям
7	Ответ	Содержание ответов на формы конкретных пользователей. Основа для создания представления, хранение информации в файле

Атрибуты отношений приведены в таблице 5

Таблица 5 – Атрибуты сущностей модели Форма

Название таблицы	Название атрибута	Тип	Ограни-чения	Описание
Форма	Идентификатор формы	целый	PK	Идентификатор формы
	Наименование	символьный		Наименование формы
	Описание	символьный		Подробное описание формы
	Тип	символьный		Тип формы: уровень доступа
	Статус	символьный		Статус: одобрена модераторами, заблокирована
	Идентификатор пользователя	целый	FK	Идентификатор пользователя
Поле формы	Идентификатор поля	целый	PK	Идентификатор поля
	Название	символьный		Название поля, его заголовок в форме
	Описание	символьный		Расширенное описание
	Время создания	Дата /время		Время создания
	Идентификатор формы	целый	FK	Идентификатор формы
	Тип поля	символьный		Тип поля

Продолжение таблицы 5

Название таблицы	Название атрибута	Тип	Ограни-чения	Описание
Тип поля формы	Идентификатор типа поля	целый	PK	Идентификатор типа поля
	Название типа поля	символьный		Название типа поля
Статус формы	Идентификатор статуса	целый	PK	Идентификатор статуса
	Название статуса	символьный		Название статуса
Пользователь	Идентификатор пользователя	целый	PK	Идентификатор пользователя
	Имя пользователя	символьный		Имя пользователя
Запрос	Идентификатор запроса	целый	PK	Идентификатор запроса
	Идентификатор формы	целый	FK	Идентификатор формы
	Идентификатор пользователя	целый	FK	Идентификатор пользователя
Ответ	Идентификатор ответа	целый	PK	Идентификатор ответа
	Идентификатор запроса	целый	FK	Идентификатор запроса
	Идентификатор поля	целый	FK	Идентификатор поля
	Значение поля	символьный		Значение поля
	Идентификатор пользователя	целый	FK	Идентификатор пользователя

На этапе физического проектирования было принято решение вынести таблицу «Пользователь» за пределы базы метаданных, так как это являлось частью базы данных платформы, которая уже успешно функционировала на сервере платформы. Схема физической модели приведена на рисунке 16

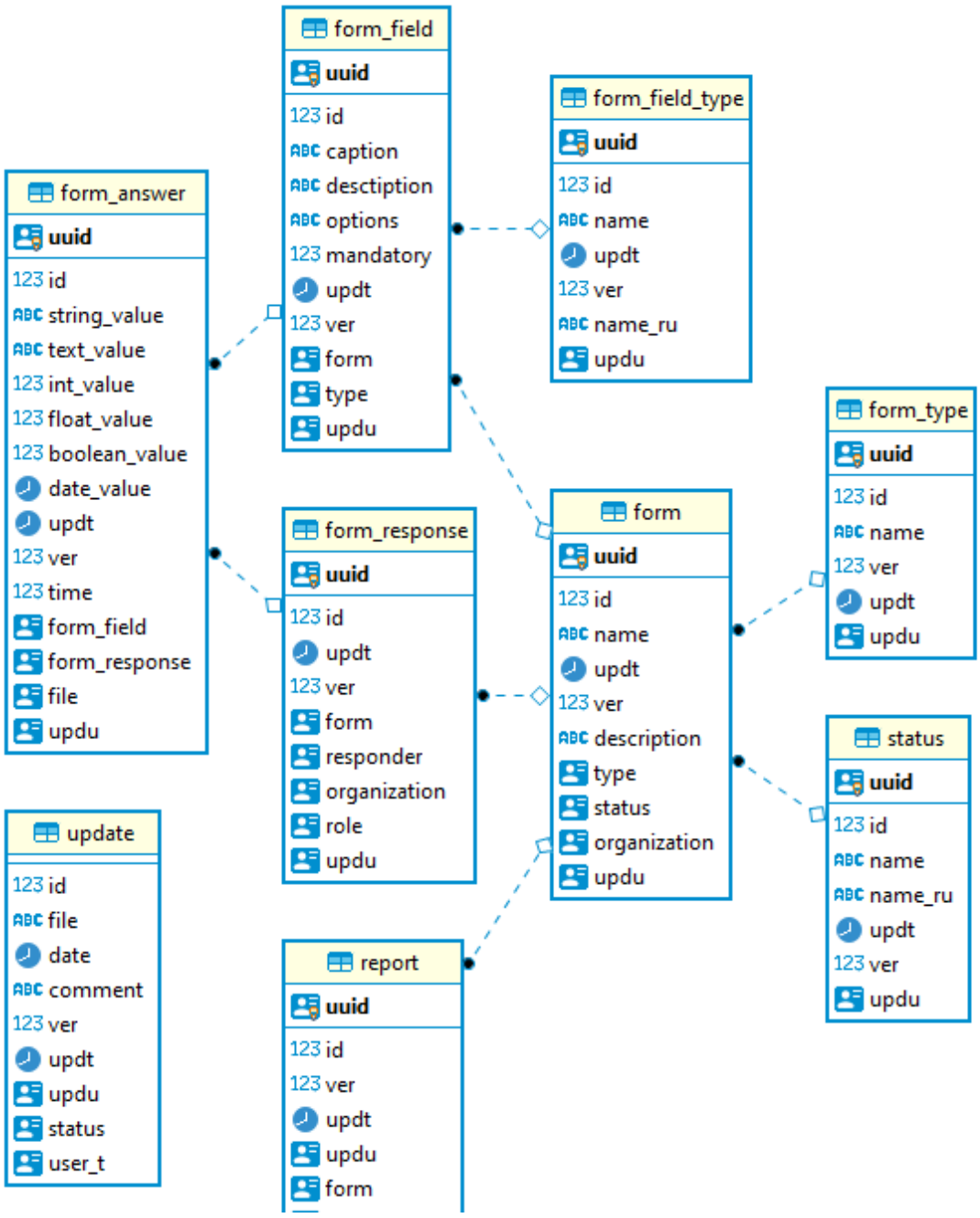


Рисунок 16- Физическая модель базы метаданных.

Через сервер приложения происходи авторизация пользователя и подсистемам, входящим в платформу «Эксперимент», необходимо отправлять запрос на аутентификацию к этому серверу. В ответ возвращается JSON-

объект, в которой содержится идентификатор пользователя, его имя и права доступа. В метаданных подсистемы «Форма» храниться только идентификатор пользователя, взятый из глобальных возвращаемых данных.

Так же произошли изменения в способе хранения значения поля. Так как было затруднительно преобразовывать «на лету» строковое значение под определённый тип поля, было принято экспериментальное решение хранить в одном из полей заданного типа, выделенные для системы управления базами данных Postgres (рисунок 17).

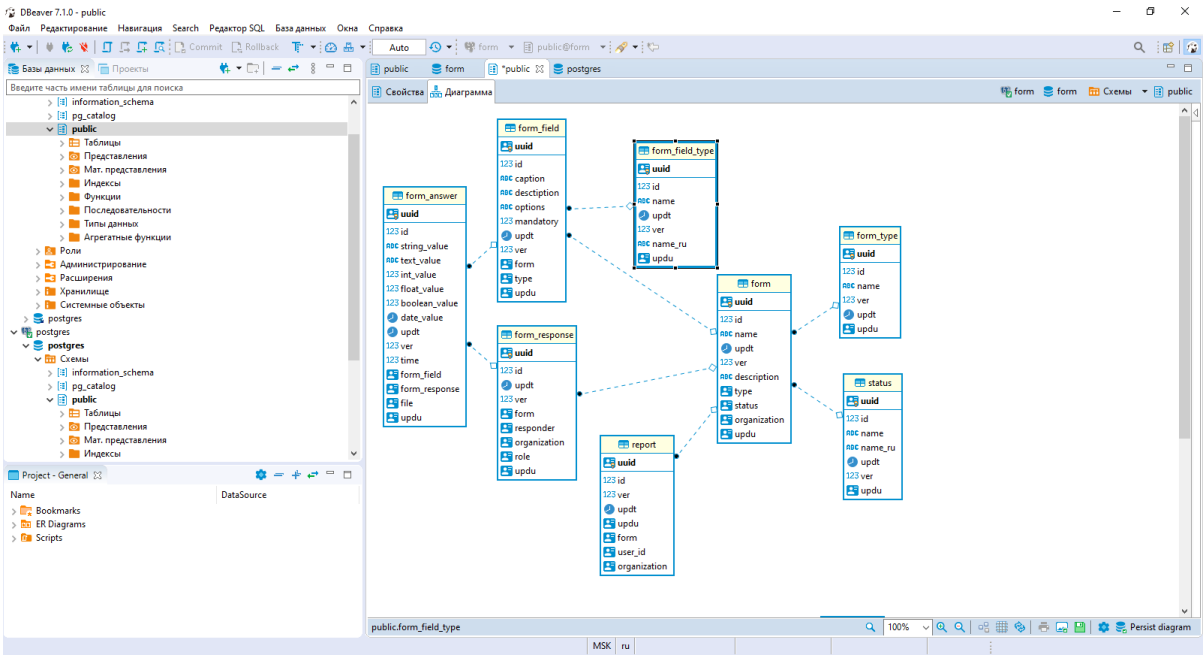


Рисунок 17– Схема Form в Dbeaver 7.0

Для доступа к базе метаданных был использован универсальный менеджер для баз данных Dbeaver 7.0. с подключенным соединением к серверу Postgres.

Обмен данными происходит через серверное приложение. Запросы выполняются на языке JPQL. Диаграмма деятельности запроса доступа пользователя к форме приведена на рисунке 17.

Рисунок 17 - Диаграмма деятельности запроса доступа пользователя к форме.

Подсистема «Форма» разрабатывается с учетом корпоративных библиотек компании «Сетап». Так как платформа является продуктом, выпустившим несколько версии в продажу с разработанной архитектурой, то архитектура подсистемы во многом будет зависеть от выбранных решений.

3.3 Архитектура ПС

Для хранения данных зарегистрированных пользователей, получения личных заданий, а также сохранения полученных результатов реализовано взаимодействие с сервером Experiment.

В настоящее время в разработке программного обеспечения достаточно часто применяется многоуровневая архитектура или многослойная архитектура, в рамках которой компоненты проекта разделяются на уровни. Классическое web-приложение часто имеет многоуровневую архитектуру, и состоит из трех или четырех уровней

Обобщенная архитектура представлена на рисунке 18

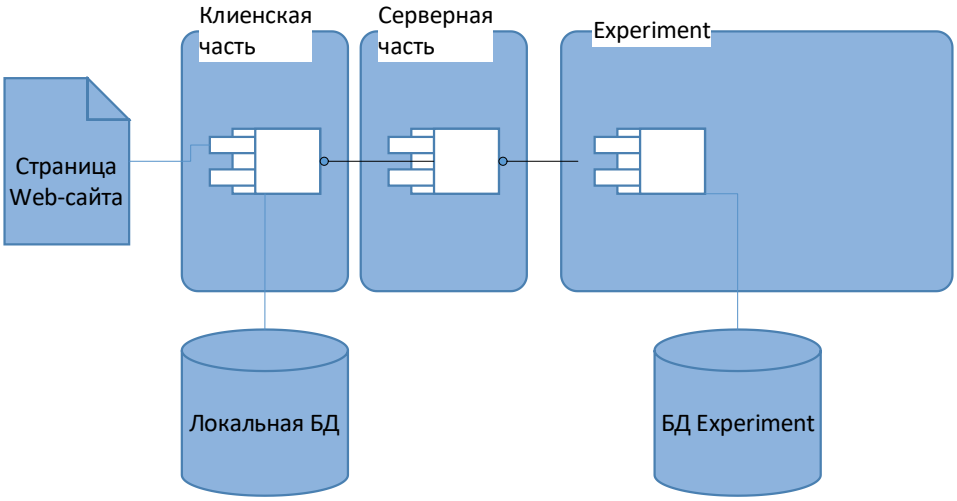


Рисунок 18 – Обобщенная архитектура

За хранение данных о пользователях отвечает серверная часть Experiment. Так же доступ к приложениям виртуальной и дополненной реальности осуществляется на основе информации о роли пользователя, зарегистрированного на образовательной платформе. Такую информацию приложение получает с сервера после авторизации. При входе в приложение. пользователь должен ввести свой логин и пароль от системы Experiment. После получения данных вызывается AJAX запрос, генерируемый приложением дополненной реальности к серверной части платформы.

AJAX – это технология, реализующая асинхронную передачу данных. Для этого в приложении создается объект «HttpRequest», выполняющий роль посредника между сервером и клиентом. Для проверки регистрации создается GET запрос, которому в роли аргумента передается URL, сформированный из адреса сервера Experiment, логина и пароля пользователя.

Отправленный запрос приходит на сервис–фильтр NGINX. Сам NGINX является веб–сервером, выступающим также в качестве балансировщика нагрузки, обратного прокси–сервера и прокси–сервера электронной почты.

Он обрабатывает полученный запрос и перенаправляет его к Userapi приложению, проверяющему по переданному токену доступ к заданному приложению.

Если доступ по данному токену не предоставляется, то отправляется ответ с ошибкой №403, информирующей о запрете доступа. Если доступ разрешен, то производится запрос к конкретному приложению на сервере, формирующему запрос к базе данных. Полученные данные отправляются на клиентское приложение в формате JSON.

Запросы данных реализованы на языка JPQL, который используется для написания запросов к сущностям, хранящимся в реляционной базе данных. JPQL во многом похож на SQL, но в отличие от последнего, оперирует запросами, составленными по отношению к сущностям JPA, в отличие от прямых запросов к таблицам базы данных.

У запросов на таком языке есть ряд преимуществ:

– простой синтаксис;
– операции объединения;
– операции группировки;
– динамические запросы;
– именованные параметры.

Запросы JPQL выполняются напрямую из программы, получая список значений Форм и Полей, запросов, ответов на них и отчета.

Для доступа к данным через метаданные используется цепочка запросов, которые позволяет получить данные на основе выбранной формы, ее полей, их типа, пользователей, вносящих данные и организации.

Запросы реализованы на языке JPQL, который используется для написания запросов к сущностям, хранящимся в реляционной базе данных. JPQL во многом похож на SQL, но в отличие от последнего, оперирует запросами, составленными по отношению к сущностям JPA, в отличие от прямых запросов к таблицам базы данных.

Листинг 1

```
1         public List<Form> findFormsForUserAndVisibility(UUID
userId, UUID orgId, Boolean isUser, UUID visibility) {
2             if (isUser) {
3                 return getEntityManager()
```

```
     4                          .createQuery("SELECT f FROM Form f WHERE
(f.author = :userId or (f.author = :zeroUuid and f.organization =
:orgId)) and f.visibility.id = :visibility", Form.class)
     5                          .setParameter("userId", userId)
     6                          .setParameter("orgId", orgId)
     7                          .setParameter("zeroUuid",
UUID.fromString("00000000-0000-0000-0000-000000000000"))
     8                          .setParameter("visibility", visibility)
     9                          .getResultList();
     10           } else {
     11               return getEntityManager()
     12                          .createQuery("SELECT f FROM Form f WHERE
f.organization=:orgId and f.visibility.id = :visibility", Form.class)
     13                          .setParameter("orgId", orgId)
     14                          .setParameter("visibility", visibility)
     15                         .getResultList();
     16           }
     17       }
```

Запросы JPQL выполняются напрямую из программы, получая список значений Форм и Полей.

Для доступа к данным через метаданные используется цепочка запросов, которые позволяет получить данные на основе выбранной формы, ее полей, их типа, пользователей, вносящих данные и организации. Запросы на выборку данных приведены на листинге 2. Остальные запросы приведены в Приложении А.

Листинг 2

```
     1    return getEntityManager()
     2    .createQuery("SELECT fr from FormResponse fr where
fr.form.id = :formId", FormResponse.class)
     3    .setParameter("formId", formId)
     4    .setFirstResult(start)
     5    .setMaxResults(100)
     6    .getResultList();}
     7    public List<FormResponse> getFormResponsesByForm(UUID
formId) {
     8    return getEntityManager()
     9    .createQuery("SELECT fr from FormResponse fr where
fr.form.id = :formId", FormResponse.class)
     10   .setParameter("formId", formId)
     11   .getResultList();}
     12   public List<Integer> getOrgListByForm(UUID formId) {
     13   return getEntityManager()
     14   .createQuery("SELECT distinct(fr.organization) from
FormResponse fr where fr.form.id = :formId", Integer.class)
     15   .setParameter("formId", formId)
     16   .getResultList();}
     17   public FormResponse getFormResponseByFormAndUser(UUID
formId, UUID userId) { List<FormResponse> frList = getEntityManager()
     18   .createQuery("SELECT fr from FormResponse fr where
fr.form.id = :formId and fr.responder = :userId", FormResponse.class)
     19   .setParameter("formId", formId)
```

```
    20    .setParameter("userId", userId)
    21    .getResultList();
    22    return frList == null || frList.size() == 0 ? null :
frList.get(0);}
```

JPA оперирует таким понятием, как сущность. При этом сущность может иметь непустые конструкторы, наследоваться или быть наследованной, содержать методы, не связанные со свойствами (методы get/set), и реализовывать интерфейсы. Сущности могут быть связаны друг с другом (один-к-одному, один-ко-многим, многие-к-одному и многие-ко-многим).

3.5 Реализация подсистемы Форма

Подсистема форм разрабатывалась в рамках образовательной платформы «Эксперимент». В рамках платформы были разработаны классы по архитектурному шаблону «Фасад» для доступа к данным и для работы с пользовательским интерфейсом.

Шаблон «Фасад» предоставляет унифицированный интерфейс вместо набора интерфейсов некоторой подсистемы. Фасад определяет интерфейс более высокого уровня, который упрощает использование подсистемы

Классы «Фасад» аккумулируют в себе высокоуровневый набор операций для работы со сложными механизмами доступа к функциям платформы. Таким образом, агрегируются классы, реализующие функциональность этой подсистемы, но не скрывает их. При этом клиентская сторона не лишается более низкоуровневого доступа к классам подсистемы, если ему это, конечно, необходимо. Фасад упрощает выполнение некоторых операций с подсистемой, но не навязывает клиенту свое использование.

Для каждой модели на языке java разработан интерфейс доступа к таблицам метаданных. Так же разработаны классы для доступа к функционалу и сервиссам системы. Диаграмма компонентов приведена на рисунке 19.

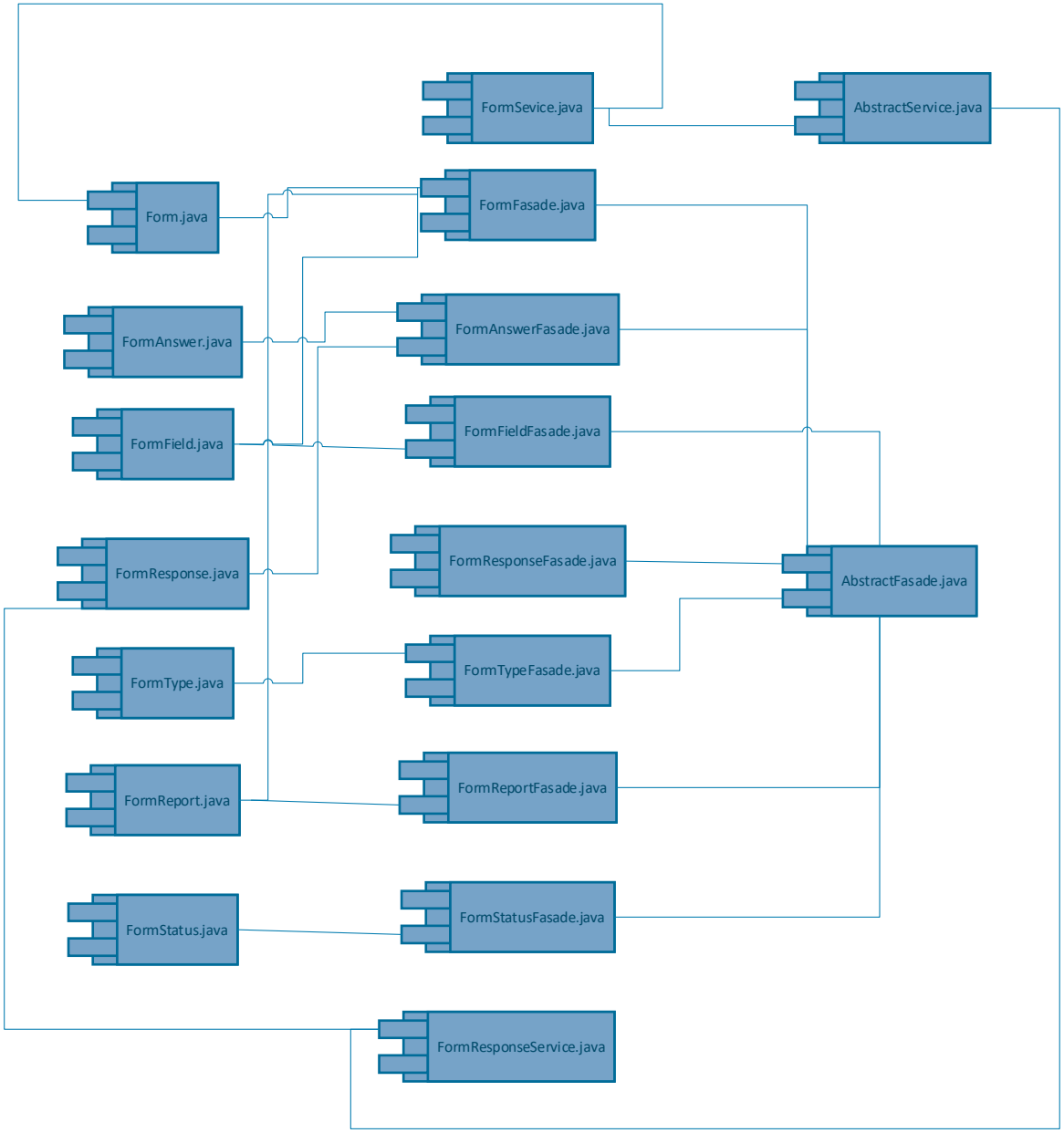


Рисунок 19 -. Диаграмма компонентов

Для отображения данных разработан пользовательский интерфейс, который отображается на платформе эксперимент. Для использования инструментария пользователь должен быть авторизован на платформе и у него должно быть право на создание форм запросов. К такому классу пользователей относятся администраторы, учителя, старосты и .т.д

Все роли имеют сложную комплексную структуру. Пользователям могут быть даны несколько ролей путем прямого назначения или с помощью предписывающих правил. С учетом занимаемой пользователем должности будет составлена подходящая совокупность ролей. Например, если родитель

одного из учеников является директором школы/учителем/главой Министерства Образования, то он одновременно может иметь две эти роли - родитель и администратор/учитель/министр.

Также пользователи могут просматривать требования к другим ролям, связанным с их собственными, и видеть границы их полномочий. Для поддержания контроля используются специальные роли администраторов школ, которые управляют доступом к информации и функциям портала. Решение предоставляет каждой из ролей разнообразные интерфейсы с настраиваемой приборной панелью для мониторинга учебной деятельности и управления.

Дашборд (Dashboard) - это начальная страница пользователя, позволяющая ему быстро и удобно выполнять различные учебные задачи и отслеживать важные события. Все виджеты находятся в пределах одной страницы, поэтому нет необходимости переключаться между вкладками..

Слева на странице есть меню перехода по подсистемам и инструментам. В инструментах есть пункт «Запросы и отчеты»

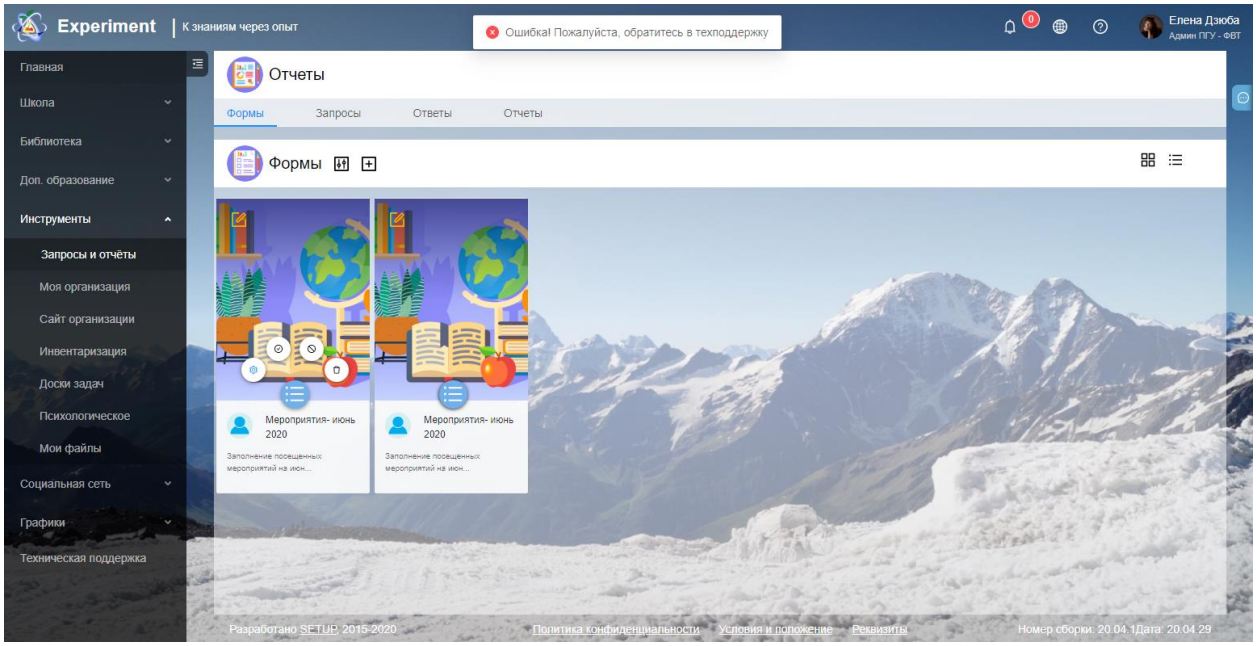


Рисунок 20 - Пользовательский интерфейс подсистемы Формы

Для создания формы необходимо нажать «+» на шапке панели, ввести название и атрибуты формы и сохранить. Алгоритм приведен на рисунке 21, а этапы создания формы из web-интерфейса на рисунке 22 .

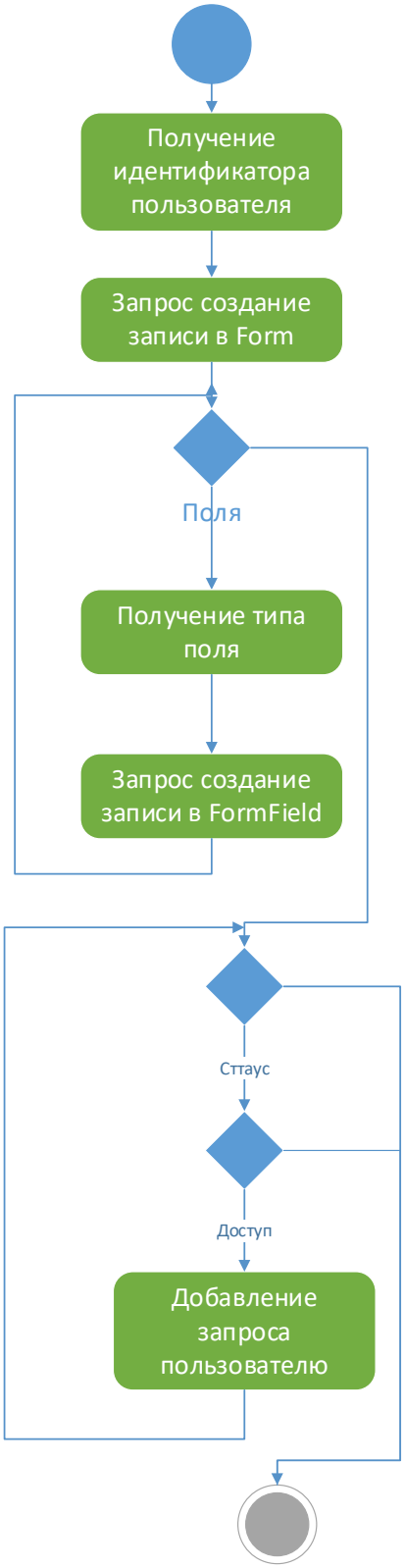


Рисунок 21 - Алгоритм добавления новой формы

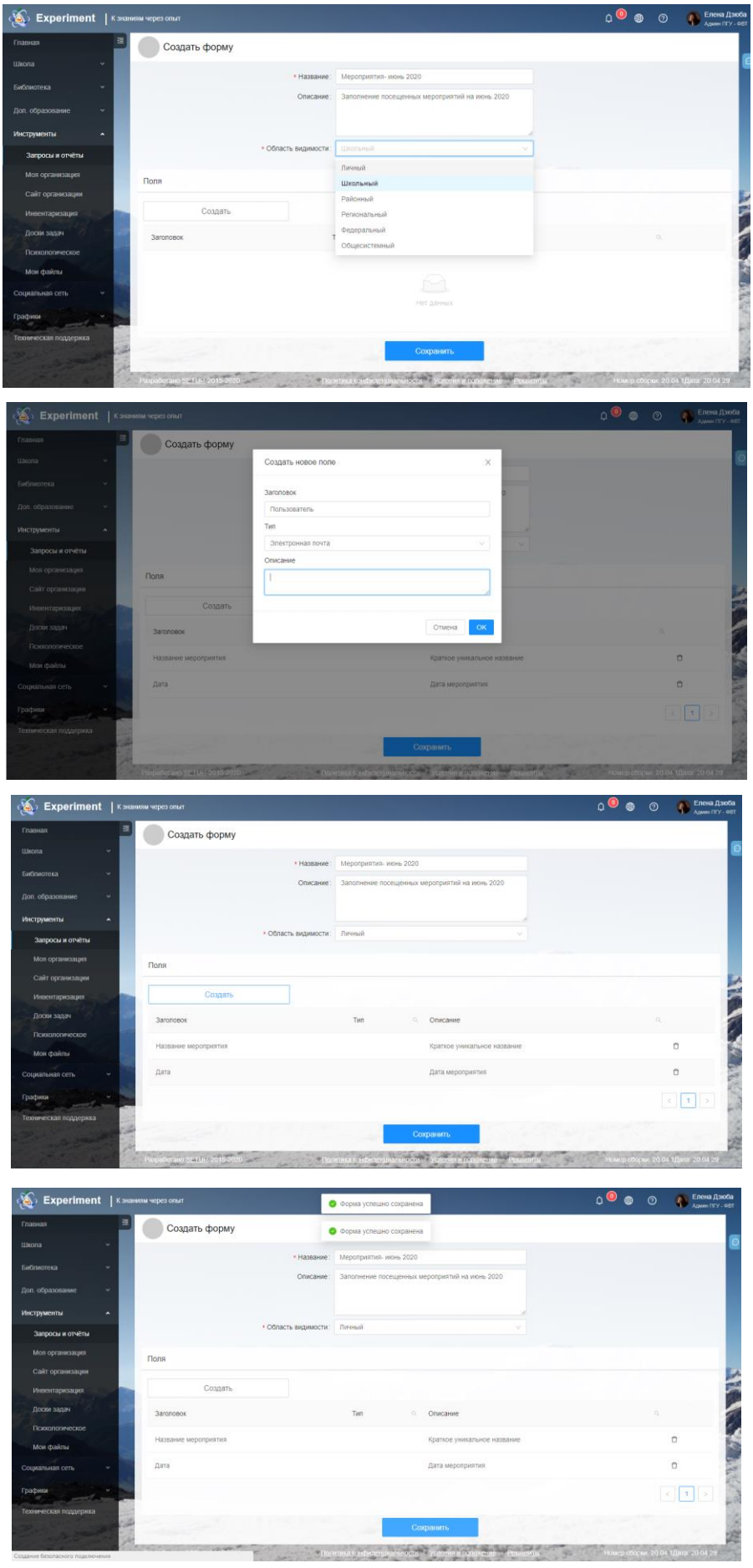


Рисунок 22 Этапы создания формы из web-интерфейса

Каждая форма представляет собой рисунок с подписью названия, описанием и всплывающим меню «Редактировать», «Удалить», «Подтвердить», «Заблокировать».

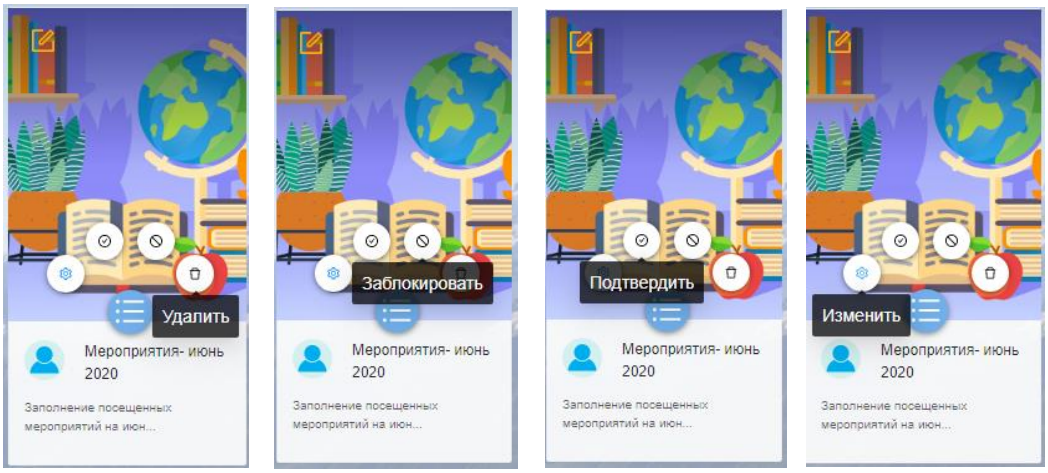


Рисунок 23 -Контекстное меню для форм

По кнопке редактировать или по кнопке «+» на панели Формы открывается страница для создания или редактирования Формы. На ней необходимо указать Название, описание, Область видимости, Поля с заголовком, типом данных и описанием.

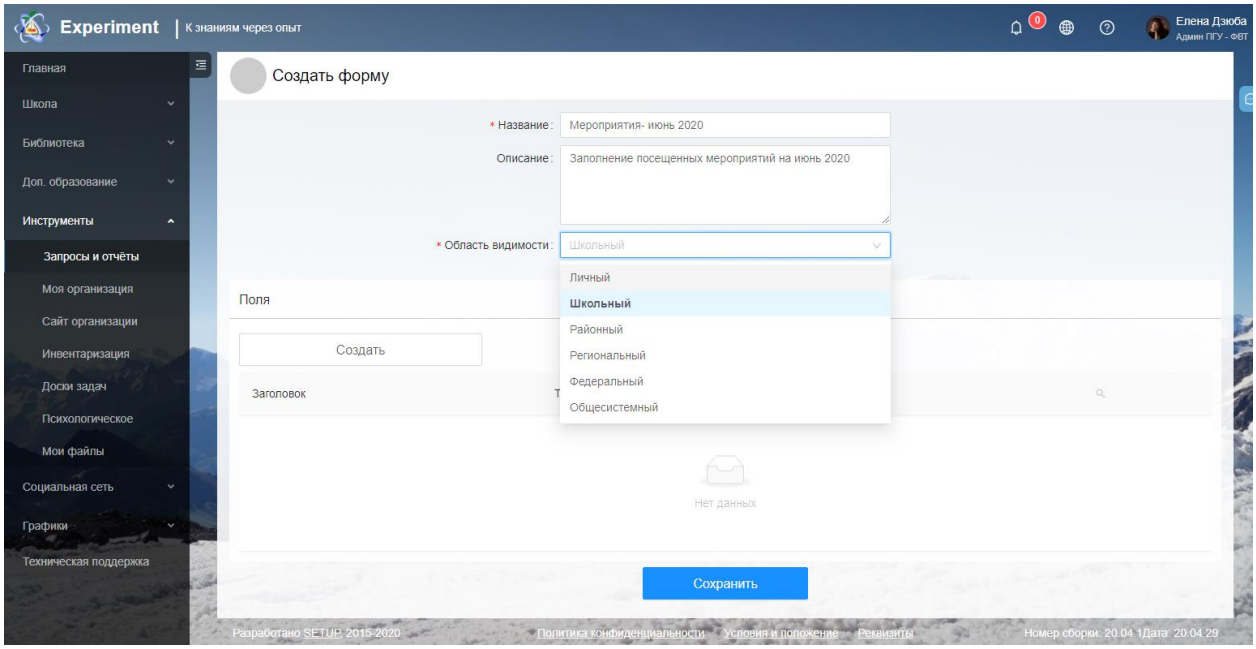


Рисунок 24 – Страница создания/редактирования формы

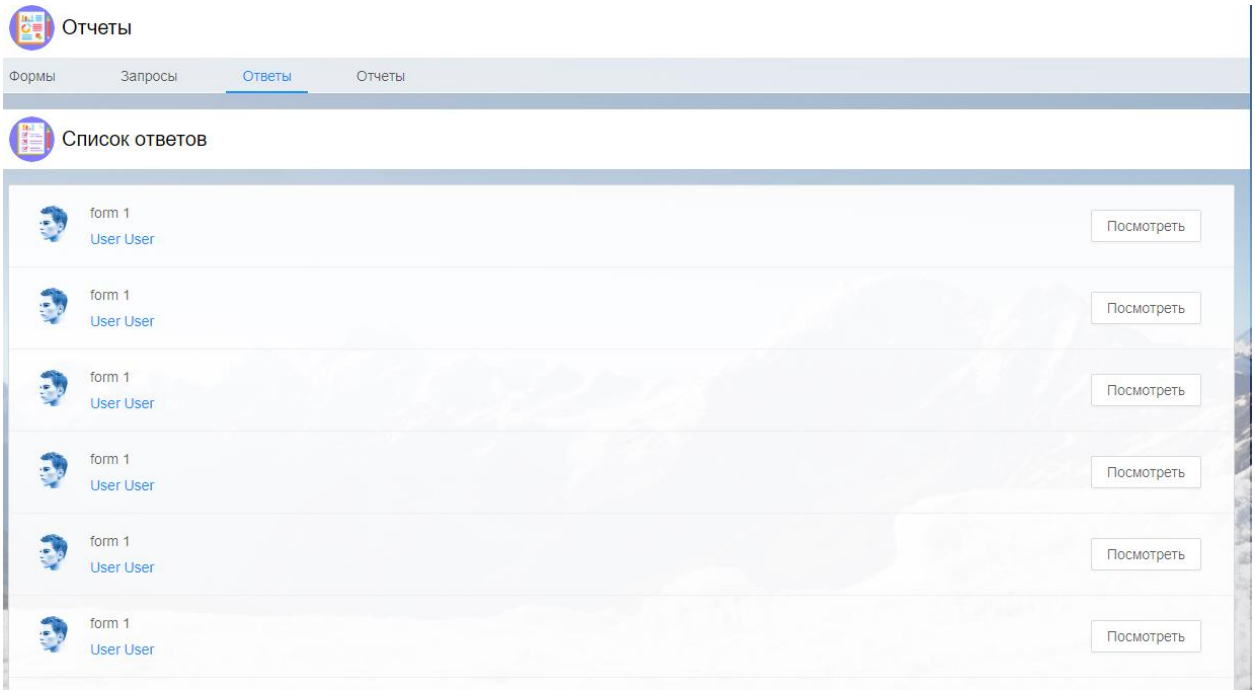


Рисунок 25 – Ответы на форму

У пользователей есть возможность оставить ответ на форму, которая им доступна. Ответы доступны администраторам данной организации и владельцам формы.

Ответ на форму выглядит как форма с заполненными полями из базы данных. Ответ приведен на рисунке 26.

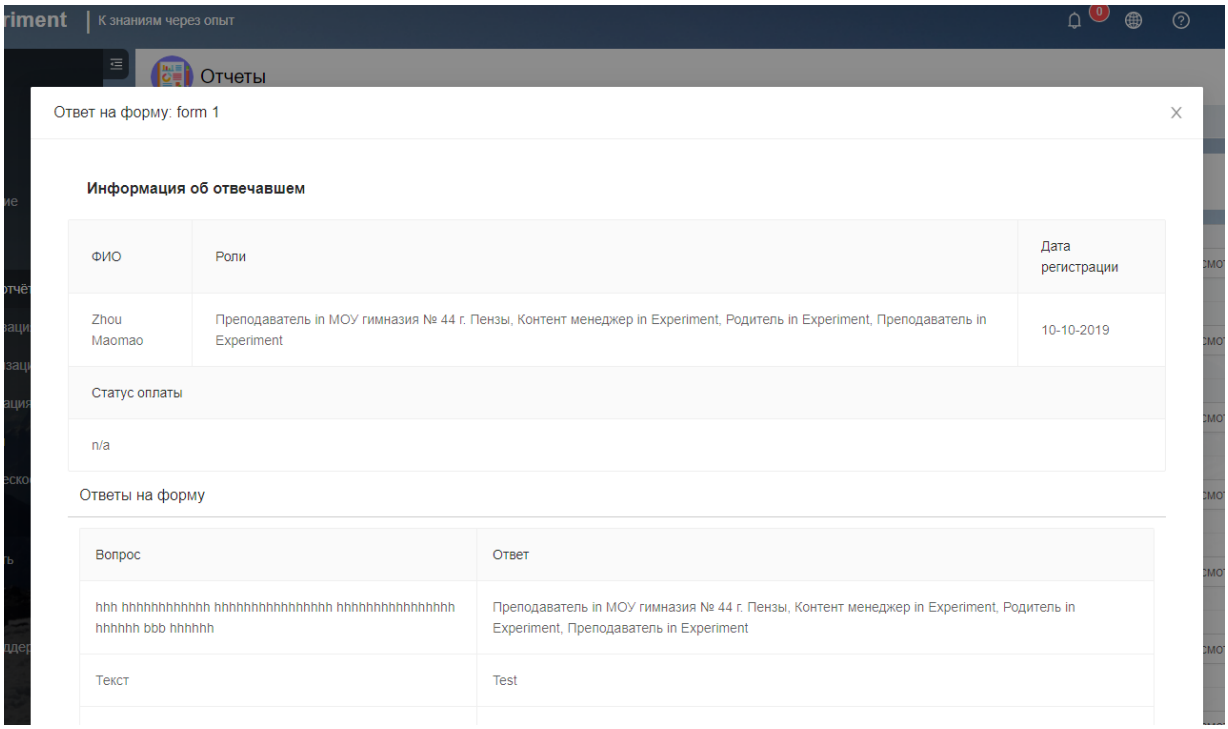


Рисунок 26 – Ответ на запрос по форме

Интерфейс выполнен по корпоративным требованиям в едином стиле платформы «Эксперимент».

3.6 Оценка алгоритма добавления новой формы в базу метаданные

Представленный на рисунке алгоритм добавления формы содержит действия, которые представляют собой один запрос на добавление одной записи в одну таблицу. По данному алгоритму построен управляющий граф, приведенный на рисунке

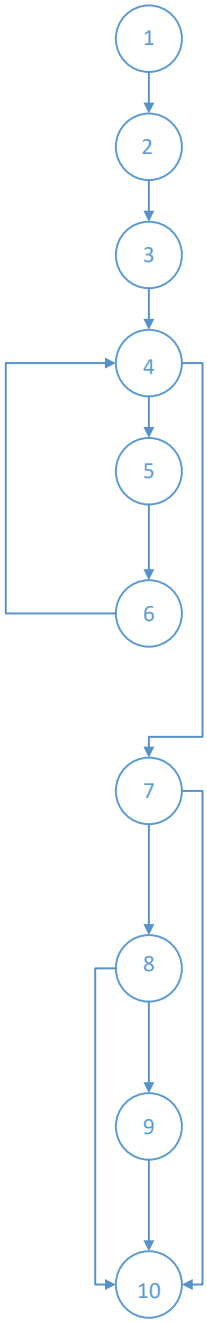


Рисунок 27 – Управляющий граф добавления формы

Цикломатическая сложность является одной из метрик программного обеспечения, которая позволяет количественно оценить логическую сложность программы.

Цикломатическая сложность может быть также применена для отдельных функций, модулей, методов и классов в пределах программы.

При вычислении цикломатической сложности используется граф потока управления программы. Узлы графа соответствуют неделимым группам команд программы, они соединены ориентированными рёбрами, если группа команд, соответствующая второму узлу, может быть выполнена непосредственно после группы команд первого узла.

Для вычисления цикломатической сложности Z(G) применяется следующая формула

$$Z(G)=e-v+2p,$$

где e - число дуг ориентированного графа G;

v - число вершин;

p - число компонентов связности графа.

Количество вершин равно десяти, количество дуг – двенадцати, таким образом цикломатическая сложность будет равна

$$Z(G)=e-v+2p = 12-10+2 = 4.$$

Данная характеристика показывает сложность поддержки кода. Чем индекс ниже, тем лучше, и тем легче в будущем будет менять структуру кода. Цикломатическая сложность метода не должна превышать 10 в поддерживаемых системах. Индекс больше говорит о том, что необходимо заменить алгоритм на частичное разбиение.

Добавление форм из 12 полей в локальной версии программы составаляет 0,25 мс. В развернутой на сайте версии, данной время, зависит от скорости подключения к Интернету.

Вывод по главе 3

В данной главе рассматривается пример применения алгоритма отображения семантических метаданных построения форм запросов в платформе «Эксперимент» и реализация подсистемы «Форма» с выделенными метаданными с предметно-ориентированной моделью. Приведены логические и физические схемы базы метаданных, диаграммы компонентов схемы, а так некоторые запросы к данным системы. . На основании бизнес-правил Проведен анализ алгоритма добавления описания новой формы в базу метаданных.

Заключение

Для того чтобы получать конкурентные преимущества от использования готовых решений или применять новые возможности в быстро меняющейся бизнес-среде, информационные системы предприятий оперативно подстраиваться под изменяющиеся требования Данное требование выполнимо при поддержке системой семантических метаданных, изменения в которых будут влиять на работу системы и ее подстраивать ее под новые задачи, процессы и данные. В данной работе предлагается использование онтологии предметной области как основы для создания семантических метаданных, отображение их в реляционную модель метаданных информационных систем.

В ходе работы были решены следующие задачи:

– проведено исследование метаданных информационных систем, их видов и функций, современных методов представления метаданных в информационных системах;

– сделан обзор существующих решений информационных систем на основе метаданных;

– разработано формального представления семантических метаданных и предложены основные компоненты онтологического представления

– разработан алгоритма отображения онтологии семантических метаданных в предметно-ориентированную и универсальную реляционные модели метаданных системы.

– реализовано отображение метаданных в подсистеме «Форма» и разработаны программные средства для работы с ней в платформе «Эксперимент»

Результаты исследования позволяют выделить онтологию предметной области, формализовать ее представление и отобразить по предложенному алгоритму в базу метаданных информационной системы.

Список использованных источников

1. Грищенко А. Системы на основе метаописаний / Андрей Грищенко А.И., Макаренко И.В. // Открытые системы, СУБД № 10, 2001

2. Когаловский М.Р. Метаданные, их свойства, функции, классификация и средства представления // Труды 14-й Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» — RCDL-2012, Переславль-Залесский, Россия, 2012 г.

3. Burnett K. A comparison of the two traditions of metadata development /J. of the American Society for Information Science. Special issue on integrating multiple overlapping metadata standards/ Burnett K., Kwong Bor Ng, Park S. //Vol. 50, Issue 13, 1999, pp.1209-1217.

4. Лядова Л. Метамоделирование многоуровневых метаданных как основа технологии создания адаптируемых информационных систем// Advanced Studies in Software and Knowledge Engineering, International Book Series "Information Science and Computing, 2009г – с. 125-132

5. Лядова Л.Н. Технология создания динамически адаптируемых информационных систем // Труды междунар. науч.-тех. конф. «Интеллектуальные системы» (AIS’07). Т. 2. – М.: Физматлит, 2007. С. 350-35

6. Васильев И. А. Методы и инструментальные средства построения семантических WEB-порталов// Томск, 2005 – с. 35-44

7. ГОСТР ИСО 15836— 2011 Набор элементов метаданных DUBLIN CORE(ISO 15836:2009, IDT) //Москва, Стандартинформ, 2018

8. Тузовский А.Ф. Системы управления знаниями (методы и технологии)/ Тузовский А.Ф., Чириков С.В., Ямпольский В.З. - Под общ. ред. В.З. Ямпольского. // Томск: Изд-во НТЛ, 2005. - 260 с

9. Уэно Х., Исидзука М. Представление и использование знаний. Том 2 Мир, 1989. 220 с ISBN 5-03-000685-0.

10. Г.М. Рудакова Представление знаний в информационных системах //«Информационные системы и технологии» - Москва 2010г

11. Советов Б.Я. Представление знаний в информационных системах: учебник для студ. учреждений высш. проф. образования / Б.Я.Советов, В.В.Цехановский, В.Д.Чертовской. — 2-е изд., стер. — М. : Издательский центр «Академия», 2012. — 144 с.

12. Вайнберг Д. Распределенные метаданные // Computerworld Россия 2003 № 05

13. Дзюба Е.А. Отображение метаданных в реляционную МОДЕЛЬ ДАННЫХ / Дзюба Е.А., Шибанов С.В., Хмелевской Б.Г., Кочегаров И.И., Гришко А.К. // Труды международного симпозиума Надежность и качество. 2010. Т. 1. С. 295-297.

14. Дзюба Е.А. Применение Карт Знаний В Описании Информационных Систем Дзюба Е.А. / Информационные технологии в науке и образовании. Проблемы и перспективы Сборник статей по материалам VII Всероссийской межвузовской научно-практической конференции. Под редакцией Л.Р. Фионовой.// Пенза - 2020. С. 139-141.

15. Прошкина Е.Н. Онтологическая модель и алгоритм принятия решений по выбору инструментальных средств разработки программного обеспечения/ Прошкина Е.Н., Балашова И.Ю., Дзюба Е.А.// Программные системы и вычислительные методы.// Пенза - 2018. Том № 1. С. 37-44.

16. Разработка интегрированных средств представления знаний в интеллектуальной системе поддержки жизненного цикла программных продуктов Балашова И.Ю., Дзюба Е.А., Прошкина Е.Н. Моделирование, оптимизация и информационные технологии. 2018. Т. 6. № 1 (20). С. 357-367.

17. Кубатин В.А. Объектная модель данных для хранения результатов работы школы битбокса /Кубатин В.А., Дзюба Е.А. В сборнике: Проблемы автоматизации и управления в технических системах сборник статей XXXII Международной научно-технической конференции. 2017. С. 135-138.

18. Болеев Д.А Проблемы верификации данных при регистрации предприятия /Болеев Д.А., Такташкин Д.В., Дзюба Е.А. В сборнике: Новые

информационные технологии и системы сборник научных статей XIII Международной научно-технической конференции. 2016. С. 5-7.

19. Шибанов С.В. Планирование и оценка учебных проектов по разработке информационных систем на основе схемы Захмана / Шибанов С.В., Дзюба Е.А. В сборнике: университетское образование XVIII Международная научно-методическая конференция, посвященная 200-летию со дня рождения М. Ю. Лермонтова. под редакцией А. Д. Гулякова, Р. М. Печерской. 2014. С. 482-484.

20. Дзюба Е.А Сравнительный анализ современных средств поддержки жизненного цикла информационных систем / Дзюба Е.А., Шибанов С.В., Герасина А.И. Труды международного симпозиума Надежность и качество. 2012. Т. 1. С. 420.

21. Дзюба Е.А Разработка информационных систем на основе конструирования знаний/ Дзюба Е.А., Шибанов С.В. Математическое и программное обеспечение систем в промышленной и социальной сферах. 2011. № 1-2. С. 207-212.

22. Дзюба Е.А. Отображение метаданных в реляционную модель данных /Дзюба Е.А., Шибанов С.В., Хмелевской Б.Г., Кочегаров И.И., Гришко А.К. Труды международного симпозиума Надежность и качество. 2010. Т. 1. С. 295-297.

23. Казакова Е.А. Проблема выбора модели данных для хранения сложных структур данных и классов/ Казакова Е.А., Шибанов С.В., Хмелевской Б.Г. Труды международного симпозиума Надежность и качество. 2008. Т. 1. С. 404-405.

24. Лядова Л.Н. Метамоделирование и многоуровневые метаданные как основа технологии создания адаптируемых информационных систем Пермь 2011(http://www.foibg.com/ibs_isc/ibs-04/IBS-04-p20.pdf)

25. Лядова Л.Н. Интеграция систем моделирования на основе dsm-платформы с использованием онтологий/ Лядова Л.Н. Сухов А.О. , Замятина Е.Б.OSTIS-2014. (Open Semantic Technologies for Intelligent Systems), 2014

26. Когаловский М.Р. Метаданные, их свойства, функции, классификация и средства представления

27. Когаловский М.Р. Метаданные в компьютерных системах Программирование, МАИК/Наука «Интерпериодика». 2013. Т. 39, № 4. С. 28-46

28. Жижимов О.Л. Метаданные и их роль в распределенных информационных системах на основе использования протокола Z39.50. Библиосфера. – 2006 – № 2 – С. 51-60.

29. Кремер О.Б. Метаданные как основа реализации программного обеспечения / Кремер О.Б. Недикова Т.Н. Вестник воронежского государственного технического университета Издательство: Воронежский государственный технический университет (Воронеж) -2013, с. 9-12

30. В. Волохов Онтологическая модель предметной области информационной системы поддержки инновационных разработок институтов РАН «Информационные ресурсы России» №3, 2011

31. Г.М. Рудакова Представление знаний в информационных системах / Курс лекций для студентов специальности 071900 (230201) «Информационные системы и технологии» очной, очно- сокращенной форм обучения , Пермь, 2012 г.

32. Б.Я.Советов Представление знаний в информационных системах : учебник для студ. учреждений высш. проф. образования / Б.Я.Советов, В.В.Цехановский, В.Д.Чертовской. — 2-е изд., стер. — М. : Издательский центр «Академия», 2012. — 144 с. — (Сер. Бакалавриат).

33. С. В. Карпенко Применение модели захмана для проектирования ИТ-архитектуры предприятия/ С. В. Карпенко, П. А. Андрухович Москва – 2013 г.

34. The Zachman Framework Evolution: overview of the evolution of the Zachman Framework by John P. Zachman at Zachman International, April 2009

35. Temnenco M. UML, RUP, and the Zachman Framework: Better together, , IBM, 15 Nov 2006.

36. Полукеев О. Моделирование бизнеса и архитектура информационной системы / О. Полукеев, Д. Коваль Системы управления базами данных #04/95

37. А.Е. Вовченко Многоуровневые спецификации в концептуальном и онтологическом моделировании А.Е. Вовченко, В.Н. Захаров, Л.А. Калиниченко, Д.Ю. Ковалёв

38. Gruber T. R., A Translation Approach to Portable Ontology Specifications. // Knowledge Acquisition. – 1993, 5(2). – pp. 199–220.

39. Муромцев Д.И. Онтологический инжиниринг знаний в системе Protégé // СПб.: СПб ГУ ИТМО, 2007.

40. Смирнов С.В. Открытая архитектура инструментальных средств моделирования сложных систем // Проблемы управления и моделирования в сложных системах: Труды международной конф. (Самара, 15-17 июня 1999 г.). — Самара: СамНЦ РАН, 1999. - С. 59

41. Burnett K., Kwong Bor Ng, Park S. A comparison of the two traditions of metadata development /J. of the American Society for Information Science. Special issue on integrating multiple overlapping metadata standards, Vol. 50, Issue 13, 1999, pp.1209-1217.

42. Grotschel M., Lugger J. Scientific Information System and Metadata. Konrad-Zuse-Zentrum fur Informationstechnik, Berlin,2015

Приложение А Карта знаний

Приложение А

КАРТА ЗНАНИЙ

(рекомендуемое)

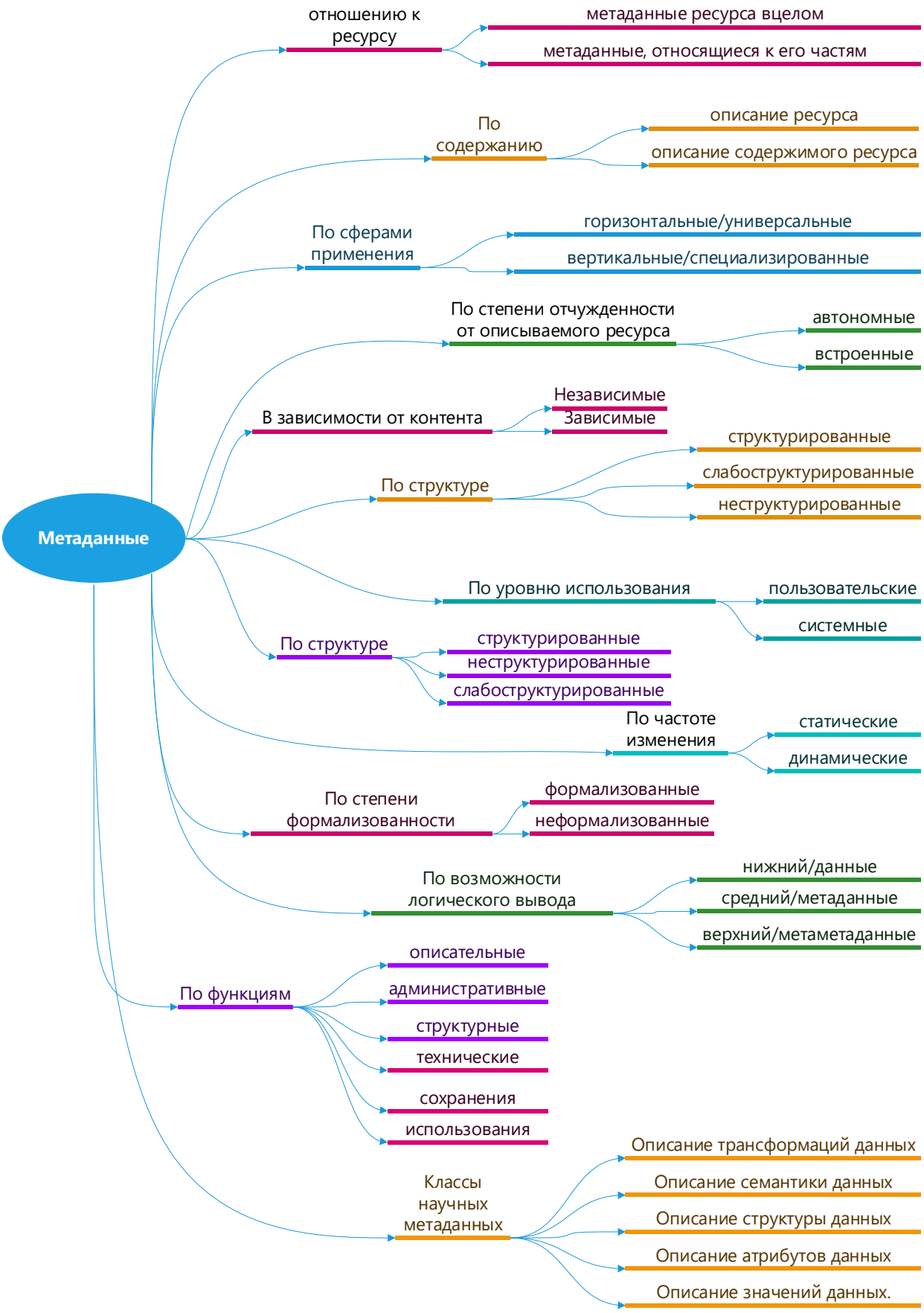


Рисунок А.1 – Виды метаданных

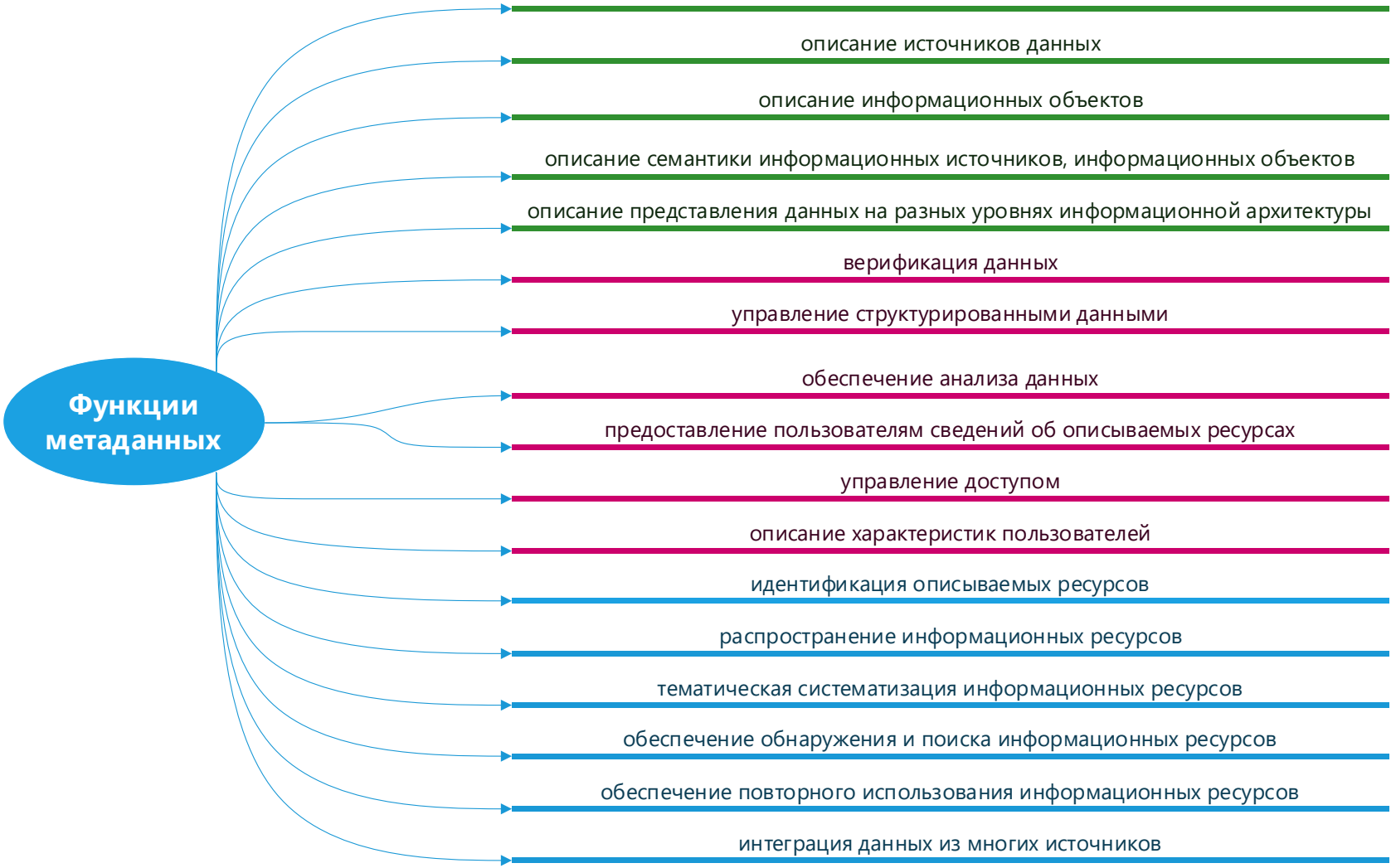


Рисунок А.2 – Функции метаданных

Приложение Б

ЛИСТИНГ

(рекомендуемое)

```
package school.experiment.form.model;

import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonProperty;
import java.util.List;
import java.util.UUID;
import javax.json.bind.annotation.JsonbTransient;
import javax.json.bind.annotation.JsonbTypeAdapter;
import javax.persistence.*;
import javax.validation.constraints.Size;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlTransient;
import org.eclipse.persistence.annotations.Convert;
import school.experiment.common.model.AbstractEntity;
import school.experiment.common.converter.UuidJsonConverter;

@Entity
@Table(name = "form")
@XmlRootElement
@NamedQueries({
      @NamedQuery(name = "Form.findAll", query = "SELECT f FROM Form f")
      , @NamedQuery(name = "Form.findByName", query = "SELECT f FROM Form f WHERE f.name
= :name")
      , @NamedQuery(name = "Form.findByOrganization", query = "SELECT f FROM Form f WHERE
f.organization = :organization")
      , @NamedQuery(name = "Form.findByUpdu", query = "SELECT f FROM Form f WHERE f.updu
= :updu")
      , @NamedQuery(name = "Form.findByUpdt", query = "SELECT f FROM Form f WHERE f.updt =
:updt")
      , @NamedQuery(name = "Form.findByVer", query = "SELECT f FROM Form f WHERE f.ver =
:ver")
      , @NamedQuery(name = "Form.findByDescription", query = "SELECT f FROM Form f WHERE
f.description = :description")})
public class Form extends AbstractEntity {

   @Lob
   @Column(name = "organization")
   @JsonbTypeAdapter(UuidJsonConverter.class)
   @Convert("uuidConverter")
   private UUID organization;

   @JsonbTransient
   @JsonIgnore
   @OneToMany(cascade = CascadeType.ALL, mappedBy = "form")
   private List<Report> reportList;

   private static final long serialVersionUID = 1L;
   @Basic(optional = false)

   @Size(min = 1, max = 1024)
   @Column(name = "name")
   private String name;

   @Size(max = 2147483647)
   @Column(name = "description")
   private String description;
   @JoinColumn(name = "type", referencedColumnName = "uuid")
   @ManyToOne(optional = false)
```

```
private FormType type;
@JoinColumn(name = "status", referencedColumnName = "uuid")
@ManyToOne(optional = false)
private Status status;

@Lob
@Column(name = "author")
@JsonbTypeAdapter(UuidJsonConverter.class)
@Convert("uuidConverter")
private UUID author;

@JoinColumn(name = "visibility", referencedColumnName = "uuid")
@ManyToOne(optional = false)
private RefData visibility;

@OneToMany(cascade = CascadeType.ALL, mappedBy = "form")
private List<FormField> formFieldList;
@JsonbTransient
@JsonIgnore
@OneToMany(cascade = CascadeType.ALL, mappedBy = "form")
private List<FormResponse> formResponseList;
@Transient
private List<FormField> formFieldListToDelete;


public Form() {
}

public Form(UUID id) {
   this.id = id;
}

public Form(UUID id, String name, UUID organization) {
   this.id = id;
   this.name = name;
   this.organization = organization;
}

public String getName() {
   return name;
}

public void setName(String name) {
   this.name = name;
}

public String getDescription() {
   return description;
}

public void setDescription(String description) {
   this.description = description;
}

public FormType getType() {
   return type;
}
```

```
public void setType(FormType type) {
   this.type = type;
}

public Status getStatus() {
   return status;
}

public void setStatus(Status status) {
   this.status = status;
}

@XmlTransient
public List<FormField> getFormFieldList() {
   return formFieldList;
}

public void setFormFieldList(List<FormField> formFieldList) {
   this.formFieldList = formFieldList;
}

@XmlTransient
@JsonbTransient
@JsonIgnore
public List<FormResponse> getFormResponseList() {
   return formResponseList;
}

public void setFormResponseList(List<FormResponse> formResponseList) {
   this.formResponseList = formResponseList;
}

@Override
public int hashCode() {
   int hash = 0;
   hash += (id != null ? id.hashCode() : 0);
   return hash;
}

@Override
public boolean equals(Object object) {
   // TODO: Warning - this method won't work in the case the id fields are not set
   if (!(object instanceof Form)) {
      return false;
   }
   Form other = (Form) object;
   if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {
      return false;
   }
   return true;
}

@Override
public String toString() {
   return "school.experiment.form.model.Form[ id=" + id + " ]";
}

/**
```

```
     * @return the formFieldListToDelete
     */
    public List<FormField> getFormFieldListToDelete() {
        return formFieldListToDelete;
    }

    /**
     * @param formFieldListToDelete the formFieldListToDelete to set
     */
    @JsonProperty
    public void setFormFieldListToDelete(List<FormField> formFieldListToDelete) {
        this.formFieldListToDelete = formFieldListToDelete;
    }

    @XmlTransient
    @JsonbTransient
    @JsonIgnore
    public List<Report> getReportList() {
        return reportList;
    }

    public void setReportList(List<Report> reportList) {
        this.reportList = reportList;
    }

    public UUID getAuthor() {
        return author;
    }

    public void setAuthor(UUID author) {
        this.author = author;
    }

    public RefData getVisibility() {
        return visibility;
    }

    public void setVisibility(RefData visibility) {
        this.visibility = visibility;
    }

    public UUID getOrganization() {
        return organization;
    }

    public void setOrganization(UUID organization) {
        this.organization = organization;
    }

}
package school.experiment.form.model;

import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonProperty;
import java.util.ArrayList;
import java.util.List;
import java.util.UUID;
import javax.faces.model.SelectItem;
```

```
import javax.json.bind.annotation.JsonbTransient;
import javax.persistence.Basic;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.JoinColumn;
import javax.persistence.Lob;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.persistence.Transient;
import javax.validation.constraints.Size;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlTransient;
import school.experiment.common.model.AbstractEntity;

@Entity
@Table(name = "form_field")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "FormField.findAll", query = "SELECT f FROM FormField f")
    , @NamedQuery(name = "FormField.findByCaption", query = "SELECT f FROM FormField f
WHERE f.caption = :caption")
    , @NamedQuery(name = "FormField.findByDesctiption", query = "SELECT f FROM FormField f
WHERE f.desctiption = :desctiption")
    , @NamedQuery(name = "FormField.findByOptions", query = "SELECT f FROM FormField f
WHERE f.options = :options")
    , @NamedQuery(name = "FormField.findByMandatory", query = "SELECT f FROM FormField f
WHERE f.mandatory = :mandatory")
    , @NamedQuery(name = "FormField.findByUpdu", query = "SELECT f FROM FormField f WHERE
f.updu = :updu")
    , @NamedQuery(name = "FormField.findByUpdt", query = "SELECT f FROM FormField f WHERE
f.updt = :updt")
    , @NamedQuery(name = "FormField.findByVer", query = "SELECT f FROM FormField f WHERE
f.ver = :ver")})
public class FormField extends AbstractEntity {

    private static final long serialVersionUID = 1L;
    @Basic(optional = false)
    @Size(min = 1, max = 1024)
    @Column(name = "caption")
    private String caption;
    @Lob
    @Size(max = 65535)
    @Column(name = "desctiption")
    private String desctiption;
    @Size(max = 2147483647)
    @Column(name = "options")
    private String options;
    @Basic(optional = false)

    @Column(name = "mandatory")
    private int mandatory;

    @JsonbTransient
    @JsonIgnore
```

```
@OneToMany(cascade = CascadeType.ALL, mappedBy = "formField")
private List<FormAnswer> formAnswerList;
@JoinColumn(name = "form", referencedColumnName = "uuid")
@ManyToOne(optional = false)
private Form form;
@JoinColumn(name = "type", referencedColumnName = "uuid")
@ManyToOne(optional = false)
private FormFieldType type;

public FormField() {
}

public FormField(UUID id) {
   this.id = id;
}

public FormField(UUID id, String caption, int mandatory) {
   this.id = id;
   this.caption = caption;
   this.mandatory = mandatory;
}


@Transient
@JsonProperty
@XmlTransient
public List<SelectItem> getOptionsList() {
   List<SelectItem> lsi = new ArrayList<>();
   if (options != null) {
      for (String option : options.split(",")) {
         SelectItem si = new SelectItem(option.trim(), option.trim());
         lsi.add(si);
      }
   }
   return lsi;
}

public void setOptions(String options) {
   this.options = options;
}

public String getCaption() {
   return caption;
}

public void setCaption(String caption) {
   this.caption = caption;
}

public String getDesctiption() {
   return desctiption;
}

public void setDesctiption(String desctiption) {
   this.desctiption = desctiption;
}

public int getMandatory() {
```

```
    return mandatory;
}

public void setMandatory(int mandatory) {
    this.mandatory = mandatory;
}

@XmlTransient
@JsonbTransient
@JsonIgnore
public List<FormAnswer> getFormAnswerList() {
    return formAnswerList;
}

public void setFormAnswerList(List<FormAnswer> formAnswerList) {
    this.formAnswerList = formAnswerList;
}

@JsonbTransient
@JsonIgnore
public Form getForm() {
    return form;
}

public void setForm(Form form) {
    this.form = form;
}

public FormFieldType getType() {
    return type;
}

public void setType(FormFieldType type) {
    this.type = type;
}

@Override
public int hashCode() {
    int hash = 0;
    hash += (id != null ? id.hashCode() : 0);
    return hash;
}

@Override
public boolean equals(Object object) {
    // TODO: Warning - this method won't work in the case the id fields are not set
    if (!(object instanceof FormField)) {
        return false;
    }
    FormField other = (FormField) object;
    if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {
        return false;
    }
    return true;
}

@Override
public String toString() {
```

```
        return "school.experiment.form.model.FormField[ id=" + id + " ]";
    }
}
package school.experiment.form.model;

import com.fasterxml.jackson.annotation.JsonIgnore;
import java.util.List;
import java.util.UUID;
import javax.json.bind.annotation.JsonbTransient;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.validation.constraints.Size;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlTransient;
import school.experiment.common.model.AbstractEntity;

@Entity
@Table(name = "form_field_type")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "FormFieldType.findAll", query = "SELECT f FROM FormFieldType f")
    , @NamedQuery(name = "FormFieldType.findByName", query = "SELECT f FROM FormFieldType
f WHERE f.name = :name")
    , @NamedQuery(name = "FormFieldType.findByUpdu", query = "SELECT f FROM FormFieldType f
WHERE f.updu = :updu")
    , @NamedQuery(name = "FormFieldType.findByUpdt", query = "SELECT f FROM FormFieldType f
WHERE f.updt = :updt")
    , @NamedQuery(name = "FormFieldType.findByVer", query = "SELECT f FROM FormFieldType f
WHERE f.ver = :ver")})
public class FormFieldType extends AbstractEntity {

    private static final long serialVersionUID = 1L;
    @Size(max = 45)
    @Column(name = "name")
    private String name;
    @Size(max = 45)
    @Column(name = "name_ru")
    private String nameRu;
    @JsonbTransient
    @JsonIgnore
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "type")
    private List<FormField> formFieldList;

    public FormFieldType() {
    }

    public FormFieldType(UUID id) {
        this.id = id;
    }


    public String getName() {
        return name;
```

```
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getNameRu() {
        return nameRu;
    }

    public void setNameRu(String name) {
        this.nameRu = name;
    }

    @XmlTransient
    @JsonbTransient
    @JsonIgnore
    public List<FormField> getFormFieldList() {
        return formFieldList;
    }

    public void setFormFieldList(List<FormField> formFieldList) {
        this.formFieldList = formFieldList;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id fields are not set
        if (!(object instanceof FormFieldType)) {
            return false;
        }
        FormFieldType other = (FormFieldType) object;
        if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "school.experiment.form.model.FormFieldType[ id=" + id + " ]";
    }

}
package school.experiment.form.model;

import com.fasterxml.jackson.annotation.JsonIgnore;
import java.util.Date;
import java.util.UUID;
import javax.json.bind.annotation.JsonbDateFormat;
import static javax.json.bind.annotation.JsonbDateFormat.TIME_IN_MILLIS;
```

```
import javax.json.bind.annotation.JsonbTransient;
import javax.json.bind.annotation.JsonbTypeAdapter;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.JoinColumn;
import javax.persistence.Lob;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.validation.constraints.Size;
import javax.xml.bind.annotation.XmlRootElement;
import org.eclipse.persistence.annotations.Convert;
import school.experiment.common.model.AbstractEntity;
import school.experiment.common.converter.UuidJsonConverter;

/**
 *
 * @author alexandr
 */
@Entity
@Table(name = "form_answer")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "FormAnswer.findAll", query = "SELECT f FROM FormAnswer f")
    , @NamedQuery(name = "FormAnswer.findByStringValue", query = "SELECT f FROM FormAnswer
f WHERE f.stringValue = :stringValue")
    , @NamedQuery(name = "FormAnswer.findByTextValue", query = "SELECT f FROM FormAnswer f
WHERE f.textValue = :textValue")
    , @NamedQuery(name = "FormAnswer.findByIntValue", query = "SELECT f FROM FormAnswer f
WHERE f.intValue = :intValue")
    , @NamedQuery(name = "FormAnswer.findByFloatValue", query = "SELECT f FROM FormAnswer
f WHERE f.floatValue = :floatValue")
    , @NamedQuery(name = "FormAnswer.findByBooleanValue", query = "SELECT f FROM
FormAnswer f WHERE f.booleanValue = :booleanValue")
    , @NamedQuery(name = "FormAnswer.findByDateValue", query = "SELECT f FROM FormAnswer f
WHERE f.dateValue = :dateValue")
    , @NamedQuery(name = "FormAnswer.findByFile", query = "SELECT f FROM FormAnswer f
WHERE f.file = :file")
    , @NamedQuery(name = "FormAnswer.findByUpdu", query = "SELECT f FROM FormAnswer f
WHERE f.updu = :updu")
    , @NamedQuery(name = "FormAnswer.findByUpdt", query = "SELECT f FROM FormAnswer f
WHERE f.updt = :updt")
    , @NamedQuery(name = "FormAnswer.findByVer", query = "SELECT f FROM FormAnswer f
WHERE f.ver = :ver")})
public class FormAnswer extends AbstractEntity {

    @Lob
    @Column(name = "file")
    @JsonbTypeAdapter(UuidJsonConverter.class)
    @Convert("uuidConverter")
    private UUID file;

    @Column(name = "time")
    private Integer time;
```

```
    private static final long serialVersionUID = 1L;
    @Size(max = 1024)
    @Column(name = "string_value")
    private String stringValue;
    @Lob
    @Size(max = 65535)
    @Column(name = "text_value")
    private String textValue;
    @Column(name = "int_value")
    private Integer intValue;
    // @Max(value=?)  @Min(value=?)//if you know range of your decimal fields consider using these
annotations to enforce field validation
    @Column(name = "float_value")
    private Double floatValue;
    @Column(name = "boolean_value")
    private Integer booleanValue;
    @Column(name = "date_value")
    @JsonbDateFormat(TIME_IN_MILLIS)
    @Temporal(TemporalType.TIMESTAMP)
    private Date dateValue;
    @JoinColumn(name = "form_field", referencedColumnName = "uuid")
    @ManyToOne(optional = false)
    private FormField formField;
    @JsonbTransient
    @JsonIgnore
    @JoinColumn(name = "form_response", referencedColumnName = "uuid")
    @ManyToOne(optional = false)
    private FormResponse formResponse;

    public FormAnswer() {
    }

    public FormAnswer(UUID id) {
        this.id = id;
    }


    public String getStringValue() {
        return stringValue;
    }

    public void setStringValue(String stringValue) {
        this.stringValue = stringValue;
    }

    public String getTextValue() {
        return textValue;
    }

    public void setTextValue(String textValue) {
        this.textValue = textValue;
    }

    public Integer getIntValue() {
        return intValue;
    }

    public void setIntValue(Integer intValue) {
```

```
    this.intValue = intValue;
}

public Double getFloatValue() {
    return floatValue;
}

public void setFloatValue(Double floatValue) {
    this.floatValue = floatValue;
}

public Integer getBooleanValue() {
    return ((booleanValue != null) && (booleanValue != 0)) ? 1 : 0;
}

public void setBooleanValue(Integer booleanValue) {
    this.booleanValue = booleanValue;
}

public Date getDateValue() {
    return dateValue;
}

public void setDateValue(Date dateValue) {
    this.dateValue = dateValue;
}

public FormField getFormField() {
    return formField;
}

public void setFormField(FormField formField) {
    this.formField = formField;
}

public FormResponse getFormResponse() {
    return formResponse;
}

public void setFormResponse(FormResponse formResponse) {
    this.formResponse = formResponse;
}

@Override
public int hashCode() {
    int hash = 0;
    hash += (id != null ? id.hashCode() : 0);
    return hash;
}

@Override
public boolean equals(Object object) {
    // TODO: Warning - this method won't work in the case the id fields are not set
    if (!(object instanceof FormAnswer)) {
        return false;
    }
    FormAnswer other = (FormAnswer) object;
    if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {
```

```
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "school.experiment.form.model.FormAnswer[ id=" + id + " ]";
    }

    public Integer getTime() {
        return time;
    }

    public void setTime(Integer time) {
        this.time = time;
    }

    public UUID getFile() {
        return file;
    }

    public void setFile(UUID file) {
        this.file = file;
    }

}
package school.experiment.form.model;

import com.fasterxml.jackson.annotation.JsonIgnore;
import java.util.List;
import java.util.UUID;
import javax.json.bind.annotation.JsonbTransient;
import javax.json.bind.annotation.JsonbTypeAdapter;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.JoinColumn;
import javax.persistence.Lob;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlTransient;
import org.eclipse.persistence.annotations.Convert;
import school.experiment.common.model.AbstractEntity;
import school.experiment.common.converter.UuidJsonConverter;

@Entity
@Table(name = "form_response")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "FormResponse.findAll", query = "SELECT f FROM FormResponse f")
    , @NamedQuery(name = "FormResponse.findByResponder", query = "SELECT f FROM
FormResponse f WHERE f.responder = :responder")
```

```
    , @NamedQuery(name = "FormResponse.findByUpdu", query = "SELECT f FROM FormResponse f
WHERE f.updu = :updu")
    , @NamedQuery(name = "FormResponse.findByUpdt", query = "SELECT f FROM FormResponse f
WHERE f.updt = :updt")
    , @NamedQuery(name = "FormResponse.findByVer", query = "SELECT f FROM FormResponse f
WHERE f.ver = :ver")})
public class FormResponse extends AbstractEntity {

    @Lob
    @Column(name = "responder")
    @JsonbTypeAdapter(UuidJsonConverter.class)
    @Convert("uuidConverter")
    private UUID responder;
    @Lob
    @Column(name = "organization")
    @JsonbTypeAdapter(UuidJsonConverter.class)
    @Convert("uuidConverter")
    private UUID organization;
    @Lob()

    @Column(name = "role")
    @JsonbTypeAdapter(UuidJsonConverter.class)
    @Convert("uuidConverter")
    private UUID role;

    private static final long serialVersionUID = 1L;

    @JsonbTransient
    @JsonIgnore
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "formResponse")
    private List<FormAnswer> formAnswerList;
    @JoinColumn(name = "form", referencedColumnName = "uuid")
    @ManyToOne(optional = false)
    private Form form;



    public FormResponse() {
    }

    public FormResponse(UUID id) {
        this.id = id;
    }

    public FormResponse(UUID id, UUID responder) {
        this.id = id;
        this.responder = responder;
    }

    public FormResponse(Form form, UUID responder) {
        this.form = form;
        this.responder = responder;
    }

    @XmlTransient
    @JsonbTransient
    @JsonIgnore
    public List<FormAnswer> getFormAnswerList() {
```

```
    return formAnswerList;
}

public void setFormAnswerList(List<FormAnswer> formAnswerList) {
    this.formAnswerList = formAnswerList;
}

public Form getForm() {
    return form;
}

public void setForm(Form form) {
    this.form = form;
}

@Override
public int hashCode() {
    int hash = 0;
    hash += (id != null ? id.hashCode() : 0);
    return hash;
}

@Override
public boolean equals(Object object) {
    // TODO: Warning - this method won't work in the case the id fields are not set
    if (!(object instanceof FormResponse)) {
        return false;
    }
    FormResponse other = (FormResponse) object;
    if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {
        return false;
    }
    return true;
}

@Override
public String toString() {
    return "school.experiment.form.model.FormResponse[ id=" + id + " ]";
}

public UUID getResponder() {
    return responder;
}

public void setResponder(UUID responder) {
    this.responder = responder;
}

public UUID getOrganization() {
    return organization;
}

public void setOrganization(UUID organization) {
    this.organization = organization;
}

public UUID getRole() {
    return role;
```

```
    }

    public void setRole(UUID role) {
        this.role = role;
    }
}
package school.experiment.form.model;

import com.fasterxml.jackson.annotation.JsonProperty;
import java.util.List;
import java.util.UUID;
import javax.json.bind.annotation.JsonbTypeAdapter;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.JoinColumn;
import javax.persistence.Lob;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.persistence.Transient;
import javax.xml.bind.annotation.XmlRootElement;
import org.eclipse.persistence.annotations.Convert;
import school.experiment.common.model.AbstractEntity;
import school.experiment.common.converter.UUIDConverterForList;
import school.experiment.common.converter.UuidJsonConverter;

@Entity
@Table(name = "report")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "Report.findAll", query = "SELECT r FROM Report r")
    , @NamedQuery(name = "Report.findByUserId", query = "SELECT r FROM Report r WHERE
r.userId = :userId")})
public class Report extends AbstractEntity {

    @Lob
    @Column(name = "user_id")
    @JsonbTypeAdapter(UuidJsonConverter.class)
    @Convert("uuidConverter")
    private UUID userId;
    @Lob
    @Column(name = "organization")
    @JsonbTypeAdapter(UuidJsonConverter.class)
    @Convert("uuidConverter")
    private UUID organization;

    private static final long serialVersionUID = 1L;
    @JoinColumn(name = "form", referencedColumnName = "uuid")
    @ManyToOne(optional = false)
    private Form form;

    @Transient
    @JsonProperty
    @JsonbTypeAdapter(UUIDConverterForList.class)
    private List<UUID> urIds;
    @Transient
    @JsonProperty
```

```
private Boolean answered;
@Transient
@JsonProperty
@JsonbTypeAdapter(UUIDConverterForList.class)
private UUID response;

public Report() {
}

public Report(UUID id) {
   this.id = id;
}

public Report(UUID id, UUID userId) {
   this.id = id;
   this.userId = userId;
}


public UUID getUserId() {
   return userId;
}

public void setUserId(UUID userId) {
   this.userId = userId;
}

public Form getForm() {
   return form;
}

public void setForm(Form form) {
   this.form = form;
}

@Override
public int hashCode() {
   int hash = 0;
   hash += (id != null ? id.hashCode() : 0);
   return hash;
}

@Override
public boolean equals(Object object) {
   // TODO: Warning - this method won't work in the case the id fields are not set
   if (!(object instanceof Report)) {
      return false;
   }
   Report other = (Report) object;
   if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {
      return false;
   }
   return true;
}

@Override
public String toString() {
   return "school.experiment.form.model.Report[ id=" + id + " ]";
```

```
    }

    /**
     * @return the urIds
     */
    public List<UUID> getUrIds() {
        return urIds;
    }

    /**
     * @param urIds the urIds to set
     */
    public void setUrIds(List<UUID> urIds) {
        this.urIds = urIds;
    }

    public UUID getOrganization() {
        return organization;
    }

    public void setOrganization(UUID organization) {
        this.organization = organization;
    }


    /**
     * @return the answered
     */
    public Boolean getAnswered() {
        return answered;
    }

    /**
     * @param answered the answered to set
     */
    public void setAnswered(Boolean answered) {
        this.answered = answered;
    }

    /**
     * @return the response
     */
    public UUID getResponse() {
        return response;
    }

    /**
     * @param response the response to set
     */
    public void setResponse(UUID response) {
        this.response = response;
    }
}
package school.experiment.form.service;

import io.swagger.annotations.Api;
import java.text.ParseException;
import java.util.*;
```

```
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.ejb.Stateless;
import javax.inject.Inject;
import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.HeaderParam;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import school.experiment.common.dto.StatusMessage;
import school.experiment.common.service.AbstractService;
import school.experiment.form.facade.FormResponseFacade;
import school.experiment.form.model.FormResponse;
import school.experiment.form.service.requests.FormSubmitRQ;

@Stateless
@Path("formResponse")
@Api(tags = {"Form response rest controller"})
public class FormResponseService extends AbstractService {

    public static String STUDENT_ROLE = "Student";

    @Inject
    private FormResponseFacade formResponseFacade;

    private static final Logger logger = Logger.getLogger(FormResponseService.class.getName());

    @GET
    @Path("{id}")
    @Produces({MediaType.APPLICATION_JSON})
    public Response find(
            @PathParam("id") UUID id,
            @HeaderParam("roleName") String roleName,
            @HeaderParam("user") UUID userId) {
        FormResponse formRes = formResponseFacade.find(id);
        if (roleName != null && roleName.equals(STUDENT_ROLE)
                && userId != null && !(userId.equals(formRes.getResponder()))) {
            return Response.status(Response.Status.NOT_FOUND).build();
        }
        FormResponse res = formResponseFacade.find(id);
        return Response.ok(res).build();
    }

    @POST
    @Path("getRespCountByFormListIds")
    @Consumes({MediaType.APPLICATION_JSON})
    @Produces({MediaType.APPLICATION_JSON})
    public Response getRespCountByFormListIds(List<UUID> formIds) {
        HashMap<UUID, Integer> result = formResponseFacade.doGetRespCountByFormListIds(formIds);
        return Response.ok(result).build();
    }

    @GET
    @Path("getOrgListByForm/{formId}")
```

```
    @Produces({MediaType.APPLICATION_JSON})
    public Response getResponsesByFormAndUserList(@PathParam("formId") UUID formId) {
        List<Integer> res = formResponseFacade.getOrgListByForm(formId);
        return Response.ok(res).build();
    }

    @POST
    @Path("get100responsesByFormAndUserList/{formId}/{isByUser}/{start}")
    @Produces({MediaType.APPLICATION_JSON})
    public Response getResponsesByFormAndUserList(
            @PathParam("formId") UUID formId,
            @PathParam("start") Integer start,
            @PathParam("isByUser") Integer isByUser, List<UUID> userIdsList) {
        try {
            List<FormResponse> res;
            if (isByUser.equals(1)) {
                res = formResponseFacade.getFormResponsesByFormAndUserIdsList(formId, userIdsList,
start);
            } else {
                res = formResponseFacade.getFormResponsesByForm(formId, start);
            }
            return Response.ok(res).build();

        } catch (Exception e) {
            return Response.serverError().entity(new StatusMessage(e)).build();
        }

    }

    @POST
    @Path("getFinalReportByFormAndUserList/{formId}/{isByUser}")
    @Produces({MediaType.APPLICATION_JSON})
    public Response getFinalReportByFormAndUserList(
            @PathParam("formId") UUID formId,
            @PathParam("isByUser") Integer isByUser,
            List<UUID> userIdsList) {
        HashMap<String, Object> retMap =
formResponseFacade.doGetFinalReportByFormAndUserList(formId, isByUser, userIdsList);
        return Response.ok(retMap).build();
    }


    /*@GET
    @Path("getFormAnswers/{frId}")
    @Produces({MediaType.APPLICATION_JSON})
    public String getFormAnswers(@PathParam("frId") Integer frId) throws JsonProcessingException {
        FormResponse formRes = formResponseFacade.find(frId);
        Map<Integer, Object>
        return (new JSONFunctions()).convert(formRes);
    }*/
    @POST
    @Path("saveResp")
    @Consumes({MediaType.APPLICATION_JSON})
    @Produces({MediaType.APPLICATION_JSON})
    public Response saveResponse(
            FormSubmitRQ fsrq,
            @HeaderParam("user") UUID userId,
            @HeaderParam("org") UUID orgId,
```

```
            @HeaderParam("role") UUID roleId) {
        try {
            UUID res = formResponseFacade.doSaveResponse(fsrq, userId, orgId, roleId);
            return Response.ok(res).build();
        } catch (ParseException ex) {
            logger.log(Level.SEVERE, null, ex);
            return Response.serverError().entity(new StatusMessage(ex)).build();
        }
    }

    @POST
    @Path("saveRespApp")
    @Consumes({MediaType.APPLICATION_JSON})
    @Produces({MediaType.APPLICATION_JSON})
    public Response saveResponseApp(
            FormSubmitRQ fsrq,
            @HeaderParam("user") UUID userId,
            @HeaderParam("org") UUID orgId,
            @HeaderParam("role") UUID roleId) {
        try {
            FormResponse formResp = formResponseFacade.doSaveResponseApp(fsrq, userId, orgId,
roleId);
            return Response.ok(formResp).build();
        } catch (ParseException ex) {
            logger.log(Level.SEVERE, null, ex);
            return Response.serverError().entity(new StatusMessage(ex)).build();
        }
    }

    @POST
    @Consumes("application/json")
    public Response createFormRes(
            FormSubmitRQ fsrq,
            @HeaderParam("user") UUID userId,
            @HeaderParam("org") UUID orgId,
            @HeaderParam("role") UUID roleId) {
        try {
            String res = formResponseFacade.doCreateFormRes(fsrq, userId, orgId, roleId);
            return Response.ok(res).build();
        } catch (ParseException ex) {
            logger.log(Level.SEVERE, null, ex);
            return Response.serverError().entity(new StatusMessage(ex)).build();
        }
    }

}package school.experiment.form.service;

import io.swagger.annotations.Api;
import java.util.*;
import javax.ejb.Stateless;
import javax.inject.Inject;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.HeaderParam;
import javax.ws.rs.POST;
```

```
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import school.experiment.common.service.AbstractService;
import school.experiment.form.facade.*;
import school.experiment.form.model.*;

@Stateless
@Path("form")
@Api(tags = {"Form rest controller"})
public class FormService extends AbstractService {

    @PersistenceContext(unitName = "org.setupit_form_war_1.0PU")
    private EntityManager entityManager;

    @Inject
    private FormFacade formFacade;
    @Inject
    private FormResponseFacade formResponseFacade;
    @Inject
    private FormAnswerFacade formAnswerFacade;
    @Inject
    private FormFieldTypeFacade formFieldTypeFacade;
    @Inject
    private FormFieldFacade formFieldFacade;
    @Inject
    private FormTypeFacade formTypeFacade;
    @Inject
    private StatusFacade statusFacade;
    @Inject
    private ReportFacade reportFacade;
    @Inject
    private RefDataFacade refDataFacade;

    @GET
    @Path("fieldTypes")
    @Produces({MediaType.APPLICATION_JSON})
    public Response getFieldTypes() {
        List<FormFieldType> res = formFieldTypeFacade.findAll();
        return Response.ok(res).build();
    }

    @GET
    @Path("reportsForUser")
    @Produces({MediaType.APPLICATION_JSON})
    public Response getReportsForUser(@HeaderParam("user") UUID userId) {
        List<Report> newR = reportFacade.getReportsForUser(userId);
        for (Report r : newR) {
            r.setAnswered(Boolean.FALSE);
        }
        return Response.ok(newR).build();
    }

    @GET
    @Path("allReportsForUser")
```

```
    @Produces({MediaType.APPLICATION_JSON})
    public Response getAllReportsForUser(@HeaderParam("user") UUID userId) {
        List<Report> ansR = reportFacade.doGetAllReportsForUser(userId);
        return Response.ok(ansR).build();
    }

    @GET
    @Path("answeredReportsForUser")
    @Produces({MediaType.APPLICATION_JSON})
    public Response getAnsweredReportsForUser(@HeaderParam("user") UUID userId) {
        List<Report> ansR = reportFacade.doGetAnsweredReportsForUser(userId);
        return Response.ok(ansR).build();
    }

    @GET
    @Path("reportsForMinister")
    @Produces({MediaType.APPLICATION_JSON})
    public Response getReportsForMinister(@HeaderParam("org") UUID orgId) {
        entityManager.getEntityManagerFactory().getCache().evictAll();
        List<Form> res = reportFacade.getReportsForMinister(orgId);
        return Response.ok(res).build();
    }

    @POST
    @Path("getNamesListByIds")
    @Produces({MediaType.APPLICATION_JSON})
    @Consumes({MediaType.APPLICATION_JSON})
    public Response getNamesByListIds(List<UUID> ids){
        HashMap<UUID, String> result = formFacade.doGetNamesByListIds(ids);
        return Response.ok(result).build();
    }

    @POST
    @Path("reports")
    @Produces({MediaType.APPLICATION_JSON})
    @Consumes({MediaType.APPLICATION_JSON})
    public Response getNamesByListIds(@HeaderParam("org") UUID orgId, Report rep){
        formFacade.doGetNamesByListIds(orgId, rep);
        return Response.ok().build();
    }



    @POST
    @Path("getListByIdsHashMap")
    @Produces({MediaType.APPLICATION_JSON})
    @Consumes({MediaType.APPLICATION_JSON})
    public Response getListByIdsHashMap(List<UUID> ids){
        HashMap<UUID, Form> result = formFacade.doGetListByIdsHashMap(ids);
        return Response.ok(result).build();
    }

    @GET
    @Path("formsByUserId/{visibility}")
    @Produces({MediaType.APPLICATION_JSON})
    public Response getMyForms(@HeaderParam("user") UUID userId, @HeaderParam("org") UUID
orgId, @PathParam("visibility") UUID visibility){
        List<Form> res = formFacade.doGetMyForms(userId, orgId, visibility);
```

```
        return Response.ok(res).build();
    }

    @GET
    @Path("getFieldForm/{id}")
    @Produces({MediaType.APPLICATION_JSON})
    public Response getFieldForm(@PathParam("id") UUID id) {
        FormField res = formFieldFacade.find(id);
        return Response.ok(res).build();
    }

    @GET
    @Path("getFieldsFormByIdForm/{id}")
    @Produces({MediaType.APPLICATION_JSON})
    public Response getFieldsFormByIdForm(@PathParam("id") UUID id) {
        List<FormField> res = formFieldFacade.findByForm(id);
        return Response.ok(res).build();
    }

    @GET
    @Path("getFormAnswersByIdForm/{id}")
    @Produces({MediaType.APPLICATION_JSON})
    public Response getFormAnswersByIdForm(@PathParam("id") UUID id,
@HeaderParam("roleName") String roleName, @HeaderParam("user") UUID userId){
        if (roleName != null && roleName.equals(FormResponseService.STUDENT_ROLE)
                && userId != null && !(userId.equals(formResponseFacade.find(id).getResponder()))) {
            return Response.status(Response.Status.NOT_FOUND).build();
        }
        List<FormAnswer> res = formAnswerFacade.findByFormResponseId(id);
        return Response.ok(res).build();
    }

    @GET
    @Path("{id}")
    @Produces({MediaType.APPLICATION_JSON})
    public Response find(@PathParam("id") UUID id) {
        entityManager.getEntityManagerFactory().getCache().evictAll();
        Form res = formFacade.find(id);
        return Response.ok(res).build();
    }

    @POST
    @Consumes({MediaType.APPLICATION_JSON})
    @Produces({MediaType.APPLICATION_JSON})
    public Response create(Form curForm, @HeaderParam("org") UUID orgId, @HeaderParam("user")
UUID userId) {
        formFacade.doCreate(curForm, orgId, userId);
        return Response.ok(curForm).build();
    }

    @PUT
    @Path("approve/{id}")
    @Consumes({MediaType.APPLICATION_JSON})
    @Produces({MediaType.APPLICATION_JSON})
    public Response approve(@PathParam("id") UUID id) {
        Form form = formFacade.doApprove(id);
        return Response.ok(form).build();
    }
```

```
    @PUT
    @Path("block/{id}")
    @Consumes({MediaType.APPLICATION_JSON})
    @Produces({MediaType.APPLICATION_JSON})
    public Response block(@PathParam("id") UUID id) {
        Form form = formFacade.doBlock(id);
        return Response.ok(form).build();
    }

    public Object setFunction(Object oldObj, Object newObj){
        if (oldObj.equals(newObj) || newObj ==null){
            return oldObj;
        }
        return newObj;
    }

    @PUT
    @Consumes({MediaType.APPLICATION_JSON})
    public Response edit(Form curForm) {
        formFacade.doEdit(curForm);
        return Response.ok().build();
    }

    @GET
    @Path("get100/{start}")
    @Produces({MediaType.APPLICATION_JSON})
    public Response find100(@PathParam("start") Integer start) {
        List<Form> res = formFacade.get100Form(start);
        return Response.ok(res).build();
    }

    @DELETE
    @Path("{id}")
    public Response remove(@PathParam("id") UUID id) {
        formFacade.remove(formFacade.find(id));
        return Response.ok().build();
    }

    public void removeFormField(Form curForm) {
        for (int i = 0; i < curForm.getFormFieldListToDelete().size(); i++) {

formFieldFacade.remove(formFieldFacade.find(curForm.getFormFieldListToDelete().get(i).getId()));
        }
    }

    public void createFormField(Form form, List<FormField> ffList) {
        for (FormField ff : ffList) {
            ff.setForm(form);
            formFieldFacade.create(ff);
        }
    }

    @GET
    @Path("getFormAnswersByFormResp/{id}")
    @Produces({MediaType.APPLICATION_JSON})
    public Response getFormAnswersByFormResp(@PathParam("id") UUID respId) {
        FormResponse fr = formResponseFacade.find(respId);
```

```
        List<FormAnswer> res = fr.getFormAnswerList();
        return Response.ok(res).build();
    }
    @GET
    @Path("getFormIdByFormResp/{id}")
    @Produces({MediaType.APPLICATION_JSON})
    public Response getFormIdByFormResp(@PathParam("id") UUID respId) {
        FormResponse fr = formResponseFacade.find(respId);
        UUID res = fr.getForm().getId();
        return Response.ok(res).build();
    }
}package school.experiment.form.facade;

import java.util.List;
import java.util.UUID;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import school.experiment.common.facade.AbstractFacade;
import school.experiment.form.model.FormField;

@Stateless
public class FormFieldFacade extends AbstractFacade<FormField> {

    @PersistenceContext(unitName = "org.setupit_form_war_1.0PU")
    private EntityManager entityManager;

    @Override
    protected EntityManager getEntityManager() {
        return entityManager;
    }

    public FormFieldFacade() {
        super(FormField.class);
    }

    public List<FormField> findByForm(UUID euId) {
        return getEntityManager().createQuery("SELECT ff FROM FormField ff WHERE ff.form.id =
:euId")
                .setParameter("euId", euId)
                .getResultList();
    }
}package school.experiment.form.facade;

import java.util.List;
import java.util.UUID;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import school.experiment.common.facade.AbstractFacade;
import school.experiment.form.model.FormAnswer;
import school.experiment.form.model.FormResponse;

@Stateless
public class FormAnswerFacade extends AbstractFacade<FormAnswer> {

    @PersistenceContext(unitName = "org.setupit_form_war_1.0PU")
    private EntityManager entityManager;
```

```
    @Override
    protected EntityManager getEntityManager() {
        return entityManager;
    }

    public FormAnswerFacade() {
        super(FormAnswer.class);
    }

    public void removeList(FormResponse fr) {
        getEntityManager()
                .createQuery("DELETE FROM FormAnswer fa WHERE fa.formResponse = :id",
FormAnswer.class)
                .setParameter("id", fr)
                .executeUpdate();
    }

    public FormAnswer findByFormResponseAndFieldId(UUID frId, UUID fieldId) {
        try {
            List<FormAnswer> resList = getEntityManager().createQuery("SELECT fa FROM FormAnswer
fa WHERE fa.formResponse.id = :frId AND fa.formField.id = :fieldId")
                    .setParameter("frId", frId)
                    .setParameter("fieldId", fieldId)
                    .getResultList();
            if (resList.size() > 0) {
                return resList.get(0);
            } else {
                return null;
            }
        } catch (Exception e) {
            return null;
        }
    }

    public List<FormAnswer> findByFormResponseId(UUID frId) {
        return getEntityManager().createQuery("SELECT fa FROM FormAnswer fa WHERE
fa.formResponse.id = :frId")
                .setParameter("frId", frId)
                .getResultList();
    }
}
package school.experiment.form.facade;

import java.util.List;
import java.util.UUID;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import school.experiment.common.facade.AbstractFacade;
import school.experiment.form.model.FormField;

@Stateless
public class FormFieldFacade extends AbstractFacade<FormField> {

    @PersistenceContext(unitName = "org.setupit_form_war_1.0PU")
    private EntityManager entityManager;
```

```
    @Override
    protected EntityManager getEntityManager() {
        return entityManager;
    }

    public FormFieldFacade() {
        super(FormField.class);
    }

    public List<FormField> findByForm(UUID euId) {
        return getEntityManager().createQuery("SELECT ff FROM FormField ff WHERE ff.form.id =
:euId")
                .setParameter("euId", euId)
                .getResultList();


package school.experiment.form.service;


import io.swagger.annotations.Api;

import java.text.ParseException;

import java.util.*;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.ejb.Stateless;

import javax.inject.Inject;

import javax.ws.rs.Consumes;

import javax.ws.rs.GET;

import javax.ws.rs.HeaderParam;

import javax.ws.rs.POST;

import javax.ws.rs.Path;

import javax.ws.rs.PathParam;

import javax.ws.rs.Produces;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import school.experiment.common.dto.StatusMessage;

import school.experiment.common.service.AbstractService;

import school.experiment.form.facade.FormResponseFacade;

import school.experiment.form.model.FormResponse;

import school.experiment.form.service.requests.FormSubmitRQ;


@Stateless

@Path("formResponse")

@Api(tags = {"Form response rest controller"})

public class FormResponseService extends AbstractService {
```

```
public static String STUDENT_ROLE = "Student";

@Inject
private FormResponseFacade formResponseFacade;

private static final Logger logger = Logger.getLogger(FormResponseService.class.getName());

@GET
@Path("{id}")
@Produces({MediaType.APPLICATION_JSON})
public Response find(
        @PathParam("id") UUID id,
        @HeaderParam("roleName") String roleName,
        @HeaderParam("user") UUID userId) {
    FormResponse formRes = formResponseFacade.find(id);
    if (roleName != null && roleName.equals(STUDENT_ROLE)
            && userId != null && !(userId.equals(formRes.getResponder()))) {
        return Response.status(Response.Status.NOT_FOUND).build();
    }
    FormResponse res = formResponseFacade.find(id);
    return Response.ok(res).build();
}

@POST
@Path("getRespCountByFormListIds")
@Consumes({MediaType.APPLICATION_JSON})
@Produces({MediaType.APPLICATION_JSON})
public Response getRespCountByFormListIds(List<UUID> formIds) {
    HashMap<UUID, Integer> result = formResponseFacade.doGetRespCountByFormListIds(formIds);
    return Response.ok(result).build();
}

@GET
@Path("getOrgListByForm/{formId}")
@Produces({MediaType.APPLICATION_JSON})
public Response getResponsesByFormAndUserList(@PathParam("formId") UUID formId) {
    List<Integer> res = formResponseFacade.getOrgListByForm(formId);
    return Response.ok(res).build();
```

```
    }

    @POST
    @Path("get100responsesByFormAndUserList/{formId}/{isByUser}/{start}")
    @Produces({MediaType.APPLICATION_JSON})
    public Response getResponsesByFormAndUserList(
            @PathParam("formId") UUID formId,
            @PathParam("start") Integer start,
            @PathParam("isByUser") Integer isByUser, List<UUID> userIdsList) {
        try {
            List<FormResponse> res;
            if (isByUser.equals(1)) {
                res = formResponseFacade.getFormResponsesByFormAndUserIdsList(formId, userIdsList,
start);
            } else {
                res = formResponseFacade.getFormResponsesByForm(formId, start);
            }
            return Response.ok(res).build();

        } catch (Exception e) {
            return Response.serverError().entity(new StatusMessage(e)).build();
        }

    }

    @POST
    @Path("getFinalReportByFormAndUserList/{formId}/{isByUser}")
    @Produces({MediaType.APPLICATION_JSON})
    public Response getFinalReportByFormAndUserList(
            @PathParam("formId") UUID formId,
            @PathParam("isByUser") Integer isByUser,
            List<UUID> userIdsList) {
        HashMap<String, Object> retMap =
formResponseFacade.doGetFinalReportByFormAndUserList(formId, isByUser, userIdsList);
        return Response.ok(retMap).build();
    }


    /*@GET
```

```
@Path("getFormAnswers/{frId}")
@Produces({MediaType.APPLICATION_JSON})
public String getFormAnswers(@PathParam("frId") Integer frId) throws JsonProcessingException {
   FormResponse formRes = formResponseFacade.find(frId);
   Map<Integer, Object>
   return (new JSONFunctions()).convert(formRes);
}*/
@POST
@Path("saveResp")
@Consumes({MediaType.APPLICATION_JSON})
@Produces({MediaType.APPLICATION_JSON})
public Response saveResponse(
      FormSubmitRQ fsrq,
      @HeaderParam("user") UUID userId,
      @HeaderParam("org") UUID orgId,
      @HeaderParam("role") UUID roleId) {
   try {
      UUID res = formResponseFacade.doSaveResponse(fsrq, userId, orgId, roleId);
      return Response.ok(res).build();
   } catch (ParseException ex) {
      logger.log(Level.SEVERE, null, ex);
      return Response.serverError().entity(new StatusMessage(ex)).build();
   }
}

@POST
@Path("saveRespApp")
@Consumes({MediaType.APPLICATION_JSON})
@Produces({MediaType.APPLICATION_JSON})
public Response saveResponseApp(
      FormSubmitRQ fsrq,
      @HeaderParam("user") UUID userId,
      @HeaderParam("org") UUID orgId,
      @HeaderParam("role") UUID roleId) {
   try {
      FormResponse formResp = formResponseFacade.doSaveResponseApp(fsrq, userId, orgId, roleId);
      return Response.ok(formResp).build();
   } catch (ParseException ex) {
      logger.log(Level.SEVERE, null, ex);
```

```
            return Response.serverError().entity(new StatusMessage(ex)).build();
        }
    }

    @POST
    @Consumes("application/json")
    public Response createFormRes(
            FormSubmitRQ fsrq,
            @HeaderParam("user") UUID userId,
            @HeaderParam("org") UUID orgId,
            @HeaderParam("role") UUID roleId) {
        try {
            String res = formResponseFacade.doCreateFormRes(fsrq, userId, orgId, roleId);
            return Response.ok(res).build();
        } catch (ParseException ex) {
            logger.log(Level.SEVERE, null, ex);
            return Response.serverError().entity(new StatusMessage(ex)).build();
        }
    }

}
```