МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

На правах рукописи

Гуськов Глеб Юрьевич

МЕТОДЫ И СРЕДСТВА ФОРМИРОВАНИЯ ПРЕДМЕТНЫХ ОНТОЛОГИЙ В АВТОМАТИЗИРОВАННОМ ПРОЕКТИРОВАНИИ ПРОГРАММНО-АППАРАТНЫХ КОМПЛЕКСОВ

Специальность 05.13.12 – Системы автоматизации проектирования

(промышленность)

Диссертация на соискание ученой степени

кандидата технических наук

НАУЧНЫЙ РУКОВОДИТЕЛЬ
кандидат технических наук,
доцент Наместников А.М.

Ульяновск – 2018

Оглавление

Список сокращений

АРМ – автоматизированное рабочее место;

АС – автоматизированная система;

ВР – временной ряд;

ИК – информационный критерий;

НИОКР – научно-исследовательские и опытно-конструкторские работы;

МЛВ – машина логического вывода;

ОП – онтологическое представление;

ПАК – программно-аппаратный комплекс;

СИПР – система интеллектуально проектирования и разработки;

УФХЗ – унифицированный формат хранения знаний;

OMG – Object Management Group;

UML – Unified modeling language;

XMI – XML Metadata Interchange.

Введение

Современный подход к автоматизированному проектированию программно-аппаратных комплексов (ПАК) предполагает использование интеллектуального проектного репозитория, позволяющего выполнять поиск и повторное использование архитектурных решений в программном обеспечении (ПО) на основе общего семантического представления предметных и проектных знаний. Знания в подобных репозиториях в настоящее время, как правило, представляются в форме онтологий.

Разработка онтологий - долгий и ресурсоемкий процесс, требующий привлечения специалистов, обладающих компетенциями в онтологическом инжиниринге, разработке программного обеспечения и предметной области программного обеспечения. Поиск специалиста с таким набором компетенций представляет из себя сложную, довольно часто не имеющую решения задачу.

Разработчики ПО зачастую не имеют достаточных знаний о предметной области, в которой решается задача автоматизации. Подобная проблема появляется из-за того, что разработчики могут переходить с проекта на проект и менять место работы. В документах, регламентирующих предметную область, не всегда зафиксированы все принятые при разработке программных продуктов семантические значения сущностей и отношений. Создание общей онтологии предметной области, учитывающей проектные решения, принятые в программных продуктах, позволяет повторно использовать найденные решения и сократить количество смысловых ошибок. Основной проблемой использования онтологий в разработке программного обеспечения остаются высокие требования к знаниям внутреннего устройства онтологий и их возможностей, предъявляемые к разработчикам.

Важность формализации концептов проблемной области для разработки ПО привела к появлению специальных языков проектирования, которые включают в себя формализацию концептов (сущностей) проблемной

области. Самые распространенные средства проектирования основываются на языке UML.

Средства, построенные на языке UML, позволяют построить проект создаваемой системы, состоящий из концептуальных элементов. Диаграммы на языке UML применимы к описанию конкретных особенностей системы, например, классов, составляющих архитектуру системы, таблиц и связей, описывающих схему базы данных, АРМ-операторов и серверов для создания схемы физического размещения.

На данный момент сформировалось достаточно большое сообщество отечественных и зарубежных исследователей в области онтологического проектирования и создания прикладных систем, основанных на онтологиях.

В работе Боргеста Н.М [7] описывается состояние дел в области онтологического проектирования на 2017 год. В работе Бениаминова Е.М. [6] рассмотрены существующие прикладные системы, которые в той или иной мере используют онтологии для хранения знаний. Научная проблематика, посвященная описанию и дальнейшему применению знаний в виде онтологий, представлена несколькими обобщающими трудами, например, Гавриловой Т.А., Хорошевского В.Ф. [10], Лукашевич Н.В [26]. В данной литературе обозначены основные проблемы и направления развития применения онтологий в информационных системах, как на уровне разработчиков программного обеспечения, так и на уровне пользователей.

Научная группа под руководством Голенкова В.В [11] создала технологию OSTIS. Ежегодно проводится одноименная конференция, посвященная созданию интеллектуальных систем. Одним из постоянных направлений является проблема поиска оптимальных форм представления знаний.

Fernando Bobillo и Umberto Straccia предложили формат описания нечеткой онтологии для представления сущностей [44], о которых невозможно судить однозначно. Предложенный формат основан на базовом синтаксисе OWL2.

Также можно отметить работы следующих ученых в онтологическом проектировании: В.А. Виттих., Грибова В.В., Добров Б.В., Загорулько Ю.А., Ландэ Д.В., Курейчик В.М, Соловьев В.Д., Смирнов С.В., Gruber T.R., Uschold M., и т.д. В работах перечисленных исследователей отмечается эффективность использования онтологического представления знаний в процессе проектирования, описываются попытки решения частных прикладных инженерных задач и фундаментальные решения на основе расширения онтологического представления знаний с помощью аппарата нечеткой логики.

На данном этапе развития промышленности создано большое количество программных систем с открытым исходным кодом в различных предметных областях. Повторное использование модулей открытых программных систем позволит существенно уменьшить затраты времени на разработку программного обеспечения. Онтологический подход к индексированию и поиску по программным решениям считается весьма перспективным, что находит подтверждение среди актуальных исследований.

За время проведения диссертационного исследования было сформировано несколько промежуточных результатов, опубликованных в рецензируемых журналах, журналах из перечня ВАК [3], [14], [15], [16], [19], [33] и на международных и всероссийских конференциях [61], [62], [63] , [101], [20], [21].

Исходя из анализа состояния исследований в сфере онтологического проектирования можно сделать вывод, что создание моделей, методов и алгоритмов, позволяющих осуществлять поиск программных продуктов, схожих по архитектуре и предметной области, представляет из себя актуальную задачу.

Предмет исследования

Модели, методы и средства формирования предметных онтологий в автоматизированном проектировании программно-аппаратных комплексов на основе проектов схожей архитектуры в рамках единой предметной области.

Объект исследования

Объектом исследования является онтологическое представление информационного обеспечения систем автоматизированного проектирования программно-аппаратных комплексов.

Актуальность темы

В настоящее время имеет место постоянный рост требований к эффективности и качеству информационного обеспечения автоматизированного проектирования технических систем в условиях слабой формализации поставленных задач, особенно на ранних этапах проектирования, который предполагает необходимость системного решения ряда научных задач:

- формирование семантического базиса представления проектов ПАК в интеллектуальном репозитории проектов;
- разработку концептуальных моделей, использующих распространенные методологии разработки программно-аппаратных комплексов, например, UML-методологии для представления знаний о предметной области проекта с использованием стандарта OWL;
- повышение эффективности автоматизированного проектирования ПАК за счет повторного использования прототипов, выбираемых из проектного репозитория на основе мер структурного подобия;
- предоставление инструментов для управления концептуальным развитием проекта и предотвращения появления смысловых ошибок.

Исходя из вышеперечисленных задач тема диссертации, посвященная разработке методов и средств представления проекта ПАК в форме OWL-

онтологии в интеллектуальном проектном репозитории, включающем в себя инструменты поиска прототипа на основе структурного подобия и управления концептуальным развитием проекта ПАК, является актуальной.

Методы исследования

В диссертационной работе применяются методы онтологического анализа, дескрипционной логики, теории нечетких систем, а также объектно-ориентированного программирования при построении программного комплекса.

Научная новизна

Научная новизна результатов исследования заключается в следующем:

1. Предложены новые модели:
   a. Онтологическая модель языка диаграмм UML, отличающаяся от известных тем, что она основана на мета-схеме языка UML построенной в виде концептов и утверждений онтологии. Данная модель позволяет внести в онтологию информацию, извлеченную из любой диаграммы на языке UML и обеспечить гибкую поддержку новых спецификаций языка UML.
   b. Онтологическая модель представления шаблона проектирования, отличающаяся от известных тем, что шаблон проектирования добавляется в онтологию в виде набора элементов, принадлежащих к классам онтологической модели языка UML, набора семантических ограничении и свойств шаблонов проектирования, описываемых в онтологии с помощью объектных свойств и свойств типов данных.
2. Разработаны новые меры архитектурного подобия программных проектов, отличающиеся новым способом вычисления степени выраженности шаблонов проектирования в рассматриваемых программных продуктах на основе их онтологических представлений.
3. Разработана методика переноса знаний из концептуальных моделей в онтологию OWL, отличающаяся тем, что отношения между элементами

UML, такие как обобщения, ассоциации, реализации и зависимости представлены набором элементов и объектных свойств с возможностью восстановления концептуальной модели по онтологическом представлению.

4. Разработан алгоритм трансформации UML диаграммы классов в онтологию формата OWL, отличающийся подходом к формированию иерархии классов UML-диаграммы и отношений между ними в OWL онтологию при помощи специфических наборов A-Box утверждений.

5. Предложен новый алгоритм агрегации методов прогнозирования временных рядов показателей состояния проекта разработки программно-аппаратного комплекса, который отличается способом интеграции результатов применения четких и нечетких методов экспоненциального сглаживания.

Теоретическая значимость работы

Теоретическая значимость работы заключается в разработке и реализации новых эффективных моделей, алгоритмов и методики представления предметной области программно-аппаратного комплекса в информационном обеспечении САПР на основе формирования онтологий из UML-диаграмм и прогнозирования динамики концептуального развития проекта, обеспечивающих снижение количества смысловых ошибок проектировщика.

Практическая значимость работы

Разработанная система интеллектуального проектирования и разработки, реализующая предложенные модели и алгоритмы, была использована:

1. В рамках проекта Автоматизированной системы балансировки мощностей системы управления АО «Авиастар-СП»;

2. В рамках проекта «Разработка ПС интеллектуальной системы анализа данных» ФНПЦ АО «НПО «Марс».

3. Подсистема анализа данных используется при формировании прогнозов в ООО «Эверест Ресерч».

Основания для выполнения работы

Результаты диссертационной работы использовались в ряде НИОКР, выполненных в Ульяновском государственном техническом университете, направленных на решение научно-технических задач.

К наиболее важным результатам следует отнести:

1. Участие в выполнении гранта РФФИ №13-01-00324 «Исследование формальных методов грануляции слабоструктурированных информационных ресурсов на основе онтологии предметной области»;
2. Участие в выполнении гранта РФФИ №15-01-03000 А «Исследование формальных методов идентификации последовательности коротких сообщений на основе нечетких онтологических моделей»;
3. Участие в выполнении гранта РФФИ №15-41-02413 «Интеллектуальный анализ временных рядов на основе нечетких онтологий, извлекаемых из Интернет-ресурсов»;
4. Участие в выполнении гранта РФФИ №16-47-730715 р_а «Исследование и разработка метода нечеткого моделирования метрик для предикативной аналитики в задачах управления разработкой программного обеспечения»;
5. Участие в выполнении гранта РФФИ №16-47-730742 р_а «Интеграция онтологических моделей и проектных диаграмм при концептуальном проектировании сложных информационных систем»;
6. Участие в выполнении гранта РФФИ №16-47-732033 р_офи_м «Разработка моделей и средств онтологического анализа проектных диаграмм на основе методов машинного обучения»;
7. Участие в выполнении гранта РФФИ №16-47-732112 р_офи_м «Исследование и разработка методов прогнозирования временных рядов на основе многомодельного подхода»;
8. Участие в выполнении гранта РФФИ №17-07-00973 А «Исследование моделей и методов нечетких онтологий в задачах анализа программно-аппаратных комплексов»;

9. Участие в реализации государственного задания №2.1182.2017/4.6 «Разработка методов и средств автоматизации производственно-технологической подготовки агрегатно-сборочного самолетостроительного производства в условиях мультипродуктовой производственной программы»;

10. Участие в реализации государственного задания № 2.4760.2017/8.9 «Исследование и разработка моделей, методов и алгоритмов гибридизации нечетких предметных онтологий, логического вывода и интеллектуального анализа временных рядов»;

11. Участие в выполнении НИР на основе хозяйственного договора с ООО «Эверест Ресерч» № «Д346» от 23.09.2014 г. «Разработка нечетких моделей и реализация нечетких методов прогнозирования временных рядов»;

12. Участие в выполнении НИР на основе хозяйственного договора с ФНПЦ АО «НПО «Марс» № «72/17-УлГТУ» от 01.02.2017.

Достоверность результатов диссертационной работы

Достоверность научных положений, выводов и рекомендаций подтверждена результатами вычислительных экспериментов, а также результатами использования материалов диссертации в работе компаний ООО «Эверест-ресерч», ФНПЦ АО «НПО «Марс» и АО «Авиастар – СП».

Основные положения, выносимые на защиту

1. Модель онтологии языка UML, содержащая описание следующих элементов: класс, абстрактный класс, интерфейс, объект, пакет, метод и атрибут, позволяет сохранить знания о архитектуре ПАК в интеллектуальное хранилище проектов проектной организации;

2. Онтологическая модель шаблона проектирования, построенная на основе объектных свойств и свойств типов данных, адекватно представляет структурные особенности программного проекта;

3. Меры архитектурного подобия программных продуктов на основе вычисления степени выраженности шаблонов проектирования

позволяют эффективно выделить прототипы среди существующих программных продуктов из репозитория;

4. Методика и алгоритм переноса знаний из концептуальных моделей и исходного кода программных продуктов в онтологию OWL эффективно транслируют проектные знания в интеллектуальное хранилище проектов проектной организации;

5. Алгоритм агрегации методов, отличающийся от существующих подходом к объединению результатов работы четких и нечетких методов экспоненциального сглаживания прогнозирования временных рядов, позволяет управлять динамикой концептуального развития проекта программно-аппаратного комплекса.

Апробация работы

Основные положения и результаты диссертационной работы докладывались, обсуждались и получили одобрение на следующих конференциях, семинарах и симпозиумах: VIII Международной научно-практической конференции «Интегрированные модели и мягкие вычисления в искусственном интеллекте» (г. Коломна, 2015 г.); Всероссийской научно-практической конференции «Нечеткие системы, мягкие вычисления и интеллектуальные технологии» (г. Санкт-Петербург, 2017 г.); XIV национальной конференции по искусственному интеллекту с международным участием «КИИ-2014» (г. Казань, 2014 г.); VI и VII Международных научно-технических конференциях «Открытые семантические технологии проектирования интеллектуальных систем» (г. Минск, 2016, 2017 гг.); I Международной научной конференции «Интеллектуальные информационные технологии в технике и на производстве» (г. Сочи, 2016 г.), XV национальной конференции по искусственному интеллекту с международным участием «КИИ-2016» (г. Смоленск, 2016 г), 4-й Всероссийской научно-технической конференции аспирантов, студентов и молодых ученых ИВТ-2012 (Ульяновск), Всероссийской научно-технической конференции «Теоретические и

практические аспекты развития отечественного авиастроения» (Ульяновск, 2012 г.), IX Всероссийской школе-семинаре аспирантов, студентов и молодых ученых «Информатика, моделирование, автоматизация проектирования» (г. Ульяновск, 2017 г.); Международной конференции «Interactive systems: Problems of Human-Computer Interaction collection of scientific papers.» (г. Ульяновск, 2017 г.); .Международной конференции «Creativity in intelligent technologies & Data Science» (г. Волгоград, 2015 г.); 2-ом международном научно-практическом семинаре «Soft computing applications and knowledge discovery (SCAKD 2016)» в рамках 13-ой международной конференции «Concept lattices and their applications (CLA 2016)» (г. Москва, 2016 г.); II Международной научной конференции «Интеллектуальные информационные технологии в технике и на производстве» (г. Варна, Болгария, 2017 г.); 12-ой Международной конференции «Uncertainty Modelling in Knowledge Engineering and Decision Making» FLINS 2016 (г. Рубе, Франция, 2016 г.), Мировой конференции по мягким вычислениям (г. Баку, Азербайджан, 2018 г.), 17-ой Международной конференции по искусственному интеллекту и мягким вычислениям (г. Закопане, Польша, 2018 г.)

Научные публикации

По результатам работы было опубликовано 32 статьи, 10 из которых в журналах из перечня ВАК, а также 6 статей в издании, индексируемом в Scopus. Получено 1 свидетельство о государственной регистрации программ для ЭВМ.

Личный вклад

Все результаты, составляющие содержание диссертации, получены автором самостоятельно. Подготовка к публикации некоторых результатов проводилась совместно с соавторами, вклад соискателя был определяющим.

Структура и объем диссертации

Диссертация состоит из введения, четырех глав, заключения, списка использованной литературы и приложений. Общий объем диссертации - 220 страниц, из них 172 страницы текста, включая 38 рисунков и 14 таблиц. Библиография включает 103 наименования на 11 страницах.

Глава 1 Анализ подходов и методов к формированию предметных онтологий для автоматизированного проектирования программно-аппаратных комплексов

1.1. Анализ отрасли разработки программного обеспечения

Программные продукты и программно-аппаратные комплексы производятся предприятиями и компаниями различного уровня и масштаба. С 1969 года фирма IBM начала разделять программную часть продукта от аппаратной. Несмотря на то, что программы могут быть абсолютно разными с точки зрения пользовательского интерфейса, внутренней логики, деления на модули, использованию аппаратных ресурсов и т.д., технологии разработки программного обеспечения при этом могут быть абсолютно одинаковыми. Отрасль производства программных продуктов относительно молода, вследствие чего стандарты проектирования и разработки часто подвергаются переосмыслению.

При разработке программного обеспечения в долгосрочной заказной разработке и разработке программного продукта появляются проблемы, связанные со знаниями предметной области. При проектировании сложных систем имеет место неполнота проектной информации [83], [88], [5]. Как правило, разработчики не имеют достаточных знаний о предметной области, в которой решаются задачи автоматизации [25]. Подобная проблема появляется из-за того, что разработчики могут переходить с проекта на проект или вовсе менять место работы.

Для того, чтобы иметь полное представление о производственном процессе, разработчикам необходимо изучить предметную область заказчика и ее отражение в программном обеспечении. Технические документы, создаваемые при разработке программных продуктов, часто не фиксируют условности и ограничения накладываемые в ходе автоматизации на предметную область.

В статье [8] отмечено, что поддержка актуальных знаний на протяжении всего жизненного цикла изделия наиболее актуальна в опытно-конструкторских и проектных организациях. Принцип поддержания единого источника знаний об изделии на протяжении жизненного цикла является одним из основных принципов разработки автоматизированных систем [9], [12], [29].

1.1.1. Анализ методологий и моделей разработки программного обеспечения

Под методологией разработки ПО принято понимать набор понятий и методов, применяемых на каждом этапе жизненного цикла разработки ПО для выполнения какого-либо базового принципа, лежащего в основе конкретной методологии. Существуют различные трактовки одной и той же методологии и бесчисленное количество модификаций, характерных для конкретных предприятий. Опыт внедрения методологий показывает необходимость учитывать индивидуальные особенности производства [40].

Необходимо отметить, что все методологии разработки ПО строятся на понятии жизненного цикла программного продукта. В рамках промышленной разработки любого программного обеспечения, разработчики занимаются проектированием, конструированием, тестированием и внедрением. Данные этапы жизненного цикла вполне естественны, а потому могут быть выделены в любой методологии. Этап поддержки программы, следующий за этапом внедрения, требует наличия пользователя, активно использующего программу и указывающего на ее недостатки.

Каскадная модель

Каскадная модель разработки программного обеспечения - итеративная модель была предложена Ройсом в 1970 году [90] и разделяла процесс разработки программного обеспечения на семь, следующих друг за другом этапов: формирование требований, проектирование, кодирование, воплощение, тестирование, поставка заказчику, поддержка ПО. Каскадная модель разработки (waterfall model) являлась единственной рекомендуемой

моделью в «РУКОВОДСТВО К СВОДУ ЗНАНИЙ ПО УПРАВЛЕНИЮ ПРОЕКТАМИ» [67], выпущенном Project Management Institute.

Основной особенностью каскадной модели является строго заданная последовательность этапов разработки. Данная особенность позволяет эффективно контролировать формальные показатели, но при этом теряется гибкость разработки. Исправление ошибок производится на стадии тестирования, все предшествующие этапы лишь накапливают ошибки и формируют временный, не структурированный код, не имеющий отражения в предметной области. Разработчики продолжительное время вынуждены работать с несовершенной реализацией системы.

Заказчик программного обеспечения, выполняемого по каскадной модели разработки, практически лишен возможности вносить изменения в требования в процессе разработки. Невозможность внесения изменений в требования порождает ситуацию, при которой по завершению разработки программное обеспечение может морально устареть и стать непригодным.

Итеративная модель

Продолжением развития идеи каскадной модели стала итеративная модель разработки, которая подразумевает большую гибкость за счет разделения проекта на части. Проект реализуется по частям инкрементно. Каждая часть проекта создается за отдельный цикл итераций, которые могут выполняться параллельно для разных частей проекта.

Итеративная модель содержит существенно меньше ограничений, чем каскадная модель и позволяет, не срывая формальные показатели, существенно чаще обеспечивать поставку новой версии программного обеспечения пользователям. Итеративная модель хорошо подходит для больших проектов, выполняемых несколькими десятками разработчиков в течении нескольких лет с четко определенными целями.

Инкрементная модель

Инкрементная модель так же является развитием идей каскадной разработки и подходит для проектов, состоящих из нескольких относительно

обособленных частей. Основным отличием от итеративной разработки являются результаты на промежуточных этапах работы. При использовании итерационной модели проектирования по завершению первого этапа все модули системы разработаны, но в каждом из них реализована лишь часть функционала, а при инкрементной модели разработки результатом каждого этапа является относительно законченный и работоспособный модуль. Полностью система будет разработана только в момент разработки всех модулей и последующей интеграции, на которую также, как правило выделяется отдельный этап.

В статье [75] описаны основные принципы итеративной модели и ее сравнение с другими наиболее популярными моделями.

Гибкая методология

Гибкие методологии разработки - серия подходов к разработке программного обеспечения, ориентированных на использование итеративной разработки, динамическое формирование требований и обеспечение их реализации в результате постоянного взаимодействия внутри самоорганизующихся рабочих групп, состоящих из специалистов различного профиля [50]. Существует несколько методик, относящихся к классу гибких методологий разработки, в частности экстремальное программирование, DSDM, Scrum, FDD. Подобные подходы применяются как эффективные практики организации труда небольших групп для решения творческих задач с часто меняющимися требованиями.

Большинство гибких методологий нацелены на минимизацию рисков путем сведения разработки к серии коротких циклов, называемых итерациями, которые обычно длятся две-три недели. Каждая итерация сама по себе выглядит как программный проект в миниатюре и включает все задачи, необходимые для выдачи минимального законченного прироста функциональности и состоит из всех основных стадий каскадной модели: планирование, анализ требований, проектирование, программирование, тестирование и документирование.

Манифест гибкой разработки [36] был составлен в 2001 году и содержал следующие основные положения:

- Сотрудники и взаимодействие важнее процессов и инструментов;
- Работающий продукт важнее исчерпывающей документации;
- Сотрудничество с заказчиком важнее согласования условий контракта;
- Готовность к изменениям важнее следования первоначальному плану.

Таблица 1.1

Сравнение методологий разработки и моделей жизненного цикла программного обеспечения

	Гибкие методологии разработки	Итерационная модель/ инкрементная модель	Каскадная модель/ итерационная модель
Масштаб применения методологии	Отдел, группы не более 7 человек	Предприятие, крупная проектная организация	Государственная структура, крупная проектная организация
Периодичность обновления продукта	От 2 до 4 недель	От 3 до 12 месяцев	От 12 месяцев
Качество проектирования	Низкое	Высокое	Исчерпывающее, избыточное
Качество документации	Низкое	Среднее	Очень высокое
Скорость модификации продукта	Высокая	Низкая	Очень низкая
Тип программного обеспечения	Программное обеспечение, обрабатывающее информацию в рамках сессии: Web-сервисы, Web-сайты, Системы управления информацией, Интернет-магазины, Игры, Экспериментальное ПО	Программное обеспечение, хранящее персональную информацию, влияющую на качество жизни пользователей: Программные инструменты, Web-сервисы, Системы обработки персональной информации, Исследовательские платформы.	Программное обеспечение, от которого напрямую зависит жизнь пользователей: Встраиваемое ПО, Авиационное ПО, Навигационное ПО, Медицинское ПО Операционные системы, Пакеты моделирования.

Программное обеспечение различается по предметной области, рискам, трудоемкости, сложности и приоритетам, в зависимости от этих параметров выбирается методология разработки и модель жизненного цикла ПО.

Зачастую целесообразно комбинировать подходы и выделять время на стадии жизненного цикла динамически. Наименее изученный модуль программного обеспечения может быть экспериментальным и для его реализации подойдет гибкий подход к разработке. Конкретная версия гибкой методологии может быть выбрана исходя из особенностей команды разработчиков.

Программный продукт скорее всего будет включать модули для импорта и экспорта данных в систему. Такие модули будут базироваться на стандартных форматах, чтобы система была совместима с уже существующим программным обеспечением. Необходимо реализовать обработку тривиального набора данных, не содержащего ошибки, а затем оптимизировать программную систему. Затем необходимо выявить набор предсказуемых ошибок и добавить логику для их обработки. Такого рода проектирование и разработку можно реализовать, используя итеративную модель [42], [75], [77].

Система может содержать множество объектно-ориентированных сущностей, представленных: классом, обработчиками, логикой управления репозитория базы данных, сущностью базы данных и т.д. Несмотря на большое количество деталей модуль, отвечающий за классы, выделенные в предметной области, как правило, содержит большое количество дублирующего и содержащего небольшие изменения исходного кода. Для проектируемого модуля дублирование исходного кода распространенное явление, позволяющее избежать ошибок на уровне интерпретации предметной области. Для такого модуля лучшим выбором будет инкрементная модель.

Можно сделать определенный выбор методологии, и задача будет решаться полностью только с помощью жестко фиксированной методологии.

В таком случае в ходе разработки будет появляться большое количество ошибок, которые непременно будут сдерживать развитие проекта на дальнейших стадиях [35], [75].

Любое программное обеспечение несовершенно и представляет из себя отображение предметной области, накапливает неопределенность модели, порожденной в ходе проектирования. Доля энтропии в программном обеспечении постоянно растет по мере его усложнения. Для решения проблемы роста сложности программного обеспечения разработчики регулярно производят рефакторинг, используют системы контроля версий, оставляют комментарии в исходном коде, используют стандартные архитектурные шаблоны, проводят повторное проектирование. Все эти операции призваны повысить уровень осознанности целей среди разработчиков и зафиксировать знания о системе в технических документах.

Подход к описанию предметной области в виде онтологического представления и ее отображения на программное обеспечение является многообещающим. Современный формат хранения знаний в виде онтологии позволит хранить семантику предметной области, ее отображение на модель, полученную в результате проектирования ПО и обеспечивать связь между синтаксическим уровнем представления знания и исходным кодом.

Инструмент, позволяющий проиндексировать технические документы в виде онтологической модели, будет чрезвычайно полезен для крупных проектов с большим количеством нюансов предметной области - второй и третий столбец Таблицы 1.1.

Гибкая разработка может быть эффективно использована для экспериментальных проектов, результатом которых является прототип системы. В таком случае использование онтологического представления технических документов может генерировать слишком много накладных расходов, заключающихся в актуализации онтологии предметной области. Помимо экспериментального ПО, гибкая разработка подходит для этапа сопровождения программного обеспечения. Для внесения изменений в

сложные проекты требуется понимание всех процессов, происходящих в программном продукте, иначе высока вероятность появления ошибок в неизмененных частях проекта. Программисты прибегают к разработке через тестирование, которая сводится к многочисленным автоматизированным проверкам работоспособности проекта до и после внесения изменений. Инструменты разработки через тестирование можно существенно модифицировать с помощью онтологического представления. В результате тестирование станет представлять из себя не только набор тестов каждого модуля ПО, но и обеспечивать проверку на целостность логической модели, написанной на языке предикатов дескрипционной логики в формате OWL. Подобная онтология может быть полностью проверена на непротиворечивость с помощью одной из машин логического вывода. Таким образом, в рамках гибкой разработки ПО онтологическое представление также может быть безусловно полезно.

Методология разработки через тестирование и разработка на основе предметной области

Онтологическое представление предметной области и проектной модели может обеспечивать существенные возможности для проектировщиков и разработчиков вне зависимости от выбранной методологии разработки с некоторыми ограничениями для экспериментального программного обеспечения и прототипирования.

Стоит отметить, что существует потребность в способах формализации предметных областей крупных программных продуктов. В ходе работы над крупным проектом, ни один разработчик не способен учитывать все особенности предметной области. Отчасти данная проблема решается с помощью методологии разработки через тестирование (Test Driven Development - TDD), которая позволяет избежать наложенных ошибок при разработке.

Тем не менее, все популярнее становится методология разработки, именуемая разработкой на основе предметной области (Domain Driven

Development - DDD) и ее сочетание с TDD. Данные подходы хорошо комбинируются между собой на основе объектно-ориентированной парадигмы программирования. При подобном подходе TDD позволяет выполнять функцию проверки качества программного продукта, но не привязана к построенной модели. В результате TDD не дает гарантии отсутствия глобальных ошибок в восприятии предметной области, контроль ошибок такого рода остается на менеджерах проекта и ведущих разработчиках.

В рамках научных исследований по формализации знаний сформулировано более полноценное решение проблемы представления предметной области, предлагающее создать отдельное ее онтологическое представление, чем принято в сложившейся практике разработки ПАК.

1.1.2. Источники знаний о состоянии программного продукта

Онтологическое представление можно разделить на ОП предметной области и ОП проектной модели. Данное разделение удобно для разделения знаний и фиксации соответствия между знаниями о предметной области и знаниями, на которых построена проектная модель. Оба типа знаний фиксируются в онтологии в формате OWL с заранее определенным T-box [86].

Разработка программного обеспечения вне зависимости от избранной методологии представляет из себя длительный итеративный процесс, состоящий из повторяющихся этапов:

- сбор требований;
- проектирование;
- разработка;
- тестирование;
- внедрение;
- сопровождение.

Сбор требований

На этапе сбора требований проектная модель еще не построена. Знания могут быть собраны из документов, описывающих предметную область: инструкции, регламенты, описания техпроцессов, описания бизнес-процессов, схемы, руководства. Основной задачей на этапе формирования требований к программной системе является сбор и формализация знаний о предметной области. Знания должны быть сразу собраны в онтологическом формате, что позволит исключить множество ошибок, связанных с неточностями, многозначностью, множественностью вариантов развития процессов, структурой используемых объектов предметной области.

Подобный исчерпывающий анализ предметной области занимает гораздо больше времени, чем оформление традиционного документа, регламентирующего требования к ПО, и позволяет:

- фиксировать результаты работы;
- оценивать оставшийся объем работ;
- порождает онтологическую структуру, поиск знаний по которой осуществляется с помощью языка запросов;
- порождает логическую модель, предусматривающую эксперименты с помощью дополнительных утверждений и машины логического вывода;
- распространяет знания о системе среди сотрудников.

Эксперименты по определению времени, затрачиваемого на анализ предметной области, представлены в 4 Главе данного диссертационного исследования. Данные эксперименты проводились на основе работы по Государственному заданию №2.1182.2017/4.6 «Разработка методов и средств автоматизации производственно-технологической подготовки агрегатно-сборочного самолетостроительного производства в условиях мультипродуктовой производственной программы» в сотрудничестве с предприятием АО «АВИАСТАР-СП».

Проектирование

На этапе проектирования создается проектная модель программного обеспечения. Существуют различные подходы к проектированию программного обеспечения.

Подходы к проектированию, основанные лишь на абстрактных рекомендациях, не могут быть рассмотрены в качестве источника знаний, так как результаты проектирования в данном случае не форматизированы. Подобные подходы к проектированию не являются бесполезными и могут успешно применяться в разработке программного обеспечения, но автоматическое и даже автоматизированное извлечение знаний и дальнейший перенос в онтологическое представление невозможны ввиду слабой формализации.

Наиболее распространенные подходы к проектированию подчиняются стандартам. Стандартизированные подходы делятся на две группы:

- подходы к проектированию, разработанные и продвигаемые корпорациями. Такие подходы могут быть очень эффективны, но как правило распространяются по лицензии и используют специфическое ПО. Например, *Rational Unified Process* [43], предлагаемый компанией IBM или *Microsoft Solutions Framework* от компании Microsoft [94];
- подходы, основанные на стандартах, созданных объединениями специалистов такие как UML, IDEF, DFD, ДРАКОН и т.д.

Методология проектирования IDEF

Методология моделирования сложных систем IDEF предоставляет широкий инструментарий для описания предметной области моделируемой системы. IDEF позволяет представить модель системы в виде набора диаграмм, часть из которых со временем была нивелирована (IDEF 2, 7, 10, 11), а оставшиеся позволяют сформировать целостное представление о системе на базе:

- Функциональной модели на диаграмме IDEF0, состоящей из функциональных блоков, связанных с помощью входных данных, выходных данных, объектов, осуществляющих преобразование, и управляющих команд;
- Диаграммы IDEF1X, отображающей данные системы как реляционной структуры, при необходимости, приведенной к требуемой нормальной форме;
- IDEF3 диаграмма определяет технический процесс и является схемой для каждого функционального блока диаграммы IDEF0. Изначально диаграмма IDEF3 создавалась для описания заводских технических процессов, поэтому она имеет несколько специфический набор элементов;
- IDEF4 диаграмма представляет диаграмму поведения объектов системы и использует классическое объектно-ориентированное представление модели;
- IDEF5 состоит из набора диаграмм, формирующих онтологическое представление системы. Диаграммы IDEF5 строятся на основе языков Schematic Language и Elaboration Language.

В списке перечислены не все возможные диаграммы, регламентируемые стандартом IDEF, а лишь те диаграммы, которые непосредственно связаны с разработкой программного обеспечения и ПАК.

Набор диаграмм IDEF 5 представлял особый интерес для исследования, но так как не существует инструмента для редактирования диаграмм данного типа, их невозможно автоматизировано обработать. На данный момент не существует универсального формата описания диаграмм IDEF5, и как следствие, инструмента преобразования файлов подобного формата в общепринятый формат описания онтологий OWL.

Диаграммы IDEF0 и IDEF 1X возможно использовать для формирования онтологического представления проектов, но ввиду слабой

распространенности и формализации было принято решение использовать более удачный формат описания системы. Диаграмма IDEF1X может быть сведена к реляционной схеме базы данных, для формализации которой гораздо удобнее использовать представление на языке SQL. Работ, посвященных преобразованию SQL в OWL довольно много. В основном, они сводятся к наборам правил, позволяющих получить соответствие между утверждением из A-box онтологии и элементом реляционной таблицы с использованием статичного T-box онтологии [39], [41], [72], [99].

Основными недостатками стандарта IDEF являются его высокий уровень абстракции, отсутствие общепринятого формата описания диаграмм, отсутствия редакторов, низкий уровень распространенности.

Методология проектирования DFD

Методология проектирования DFD является иерархичным набором диаграмм потоков данных. Диаграммы методологии DFD схожи по элементной базе и семантике с диаграммами нотации IDEF0. Основным отличием DFD-диаграммы от IDFE0 является ее специализация для проектирования автоматизированных информационных систем. Правила построения DFD диаграммы гораздо менее строгие в сравнении с IDEF0, за счет чего DFD диаграммы обладают большей гибкостью.

Первым и наиболее существенным недостатком методологии DFD является ограниченность набора диаграмм всего одним типом. Есть возможность построить диаграммы потоков данных на разных уровнях разрабатываемой ИС, но для исчерпывающего описания ИС возможностей методологии DFD явно недостаточно.

Формат хранения диаграммы четко не определен, из-за чего каждый из представленных редакторов взаимодействует с проектом в своем внутреннем формате. Исходный код программной части ИС не может быть однозначно сопоставлен с DFD-диаграммой, что влечет устаревание информации, полученной на этапе проектирования. Немаловажным параметром для включения источника знаний в исследование так же является

распространенность формата среди профессионального сообщества. DFD-диаграммы практически полностью вытеснены более новым и совершенным стандартом проектирования UML.

Методология проектирования UML

Унифицированный язык моделирования UML – является графическим языком для проектирования, описания спецификации, конструирования, документирования и визуализации ПО в виде набора определенных диаграмм. [46]. Язык UML был создан группой авторов в которую входили Гради Буч, Джеймс Рамбо и Ивар Якобсон. Все разработчики языка UML ранее независимо работали над задачей создания графического языка проектирования, объектно-ориентированного ПО. В языке UML были собраны все наиболее актуальные и эффективные подходы к визуальному проектированию. Стандарт языка UML формируется консорциумом OMG (Object Management Group), в который входят университеты, крупные компании и корпорации, правительственные организации и т.д. Язык UML постоянно актуализируется. Последняя версия спецификации языка UML 2.5.1 [85] вышла в декабре 2017 года.

Отдельно стоит выделить унифицированный формат описания диаграмм XMI, который так же обновляется вместе с каждой новой спецификацией языка UML. Данный формат позволяет гарантировать соответствие диаграммы, созданной в одном из многих редакторов, спецификации консорциума OMG. Формат XMI предполагает описание документов не только уровня диаграммы, но и документов мета-уровня. Каждая спецификация языка UML поставляется вместе с описанием в виде XMI документа и нескольких, сопутствующих XMI документов: описание примитивных типов, модель обмена метаданными и др.

Язык UML позволяет создавать различные диаграммы (Рисунок 1.1), построенные на одном и том же множестве элементов, и принципах их взаимодействия.

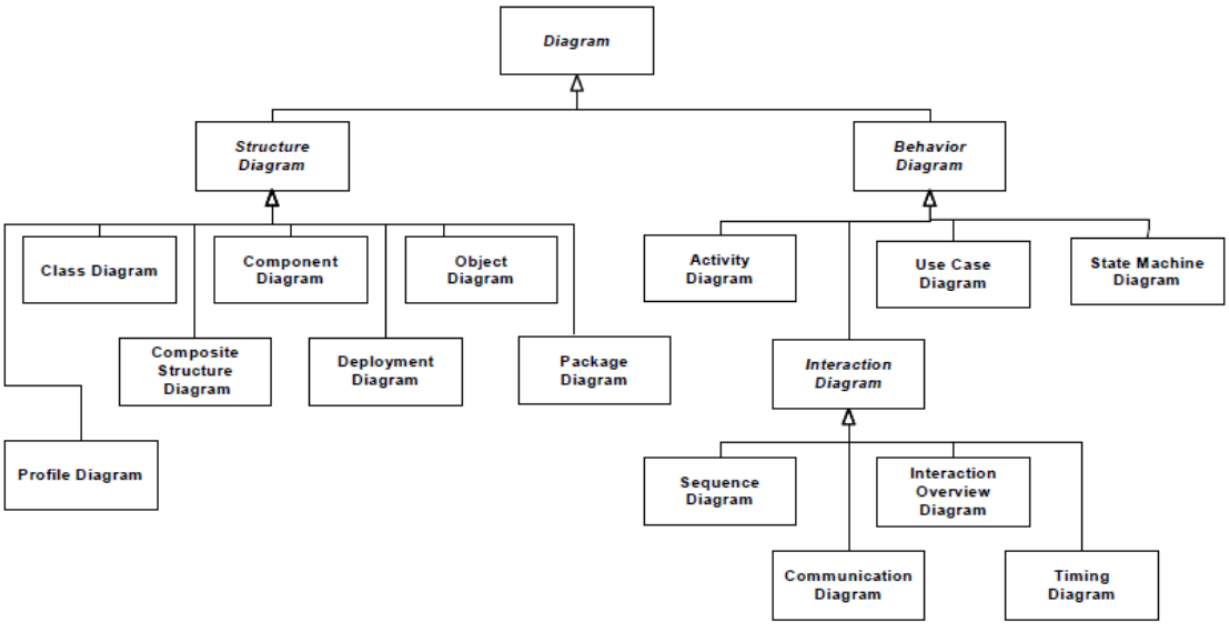

Рис 1.1 Классификация диаграмм UML.

Диаграммы UML делятся на диаграммы, описывающие структуру ПО, и диаграммы, описывающие поведение. Для описания ПО не обязательно создавать все 14 диаграмм. Как правило проектировщик выбирает диаграммы, наиболее подходящие для описания особенностей конкретной программной системы. Не все UML диаграммы одинаково востребованы и распространены.

К структурным UML диаграммам относятся:

- Диаграмма классов (Class diagram) – описывает внутреннее устройство ПО, разработанного с помощью объектно-ориентированного подхода к описанию предметной области. Диаграмма классов является самой базовой и распространенной из UML диаграмм, создаваемых при проектировании. Диаграмма классов может быть построена для уже существующей системы с помощью автоматического обратного проектирования;
- Диаграмма пакетов (Package Diagram) и диаграмма компонентов (Component Diagram) позволяют спроектировать более высокий уровень структуры системы, чем диаграмма классов, но используют элементы диаграммы классов. Четкого разделения между данными диаграммы пакетов и диаграммой классов нет.

Основные аспекты программы касающиеся взаимодействия пакетов и компонент могут быть показаны на диаграмме классов, поэтому диаграммы пакетов и компонент используются при проектировании гораздо реже;

- Диаграмма объектов (Object Diagram) и диаграмма композитной структуры (Composite Structure Diagram) являются вспомогательными для диаграммы классов. Данные диаграммы создаются в случае, если синтаксических средств диаграммы классов недостаточно для описания взаимодействия объектов или классов. Диаграмма объектов хоть и относиться к структурным диаграммам, но описывает не структуру системы, а взаимодействие элементов программы во время работы;
- Profile Diagram – расширенная диаграмма классов, позволяющая устанавливать ограничения на элементы, добавлять закрепленные значения и стереотипы. В наиболее распространенных UML редакторах возможности Profile Diagram включены в диаграмму классов;
- Диаграмма развертывания (Deployment Diagram) имеет существенное семантическое отличие от остальных структурных диаграмм. Диаграмма развертывания создается для того, чтобы спроектировать физическое размещение компонентов программного комплекса.

Среди структурных UML диаграмм наибольший интерес для диссертационного исследования представляет диаграмма классов ввиду широкой распространенности и достаточно выразительного синтаксиса.

Остальные структурные UML диаграммы в той или иной мере являются расширением возможностей диаграммы классов. Анализ редакторов UML диаграмм показал, что альтернативные структурные диаграммы могут отсутствовать. Возможность создания диаграммы классов присутствует во всех рассмотренных средствах проектирования. За счет иерархической

структуры языка UML элементы одной диаграммы могут быть успешно использованы в другой диаграмме.

Диаграммы поведения системы представляют меньший интерес для диссертационного исследования. Разработанное в результате диссертационного исследования ПО может быть впоследствии расширено информацией, извлекаемой из диаграмм поведения, так как представление информации из диаграммы классов основывается на расширяемой метамодели языка UML. Наиболее интересной диаграммой поведения для исследования является диаграмма вариантов использования (Use case Diagram), которая позволит определить пользователей и их функциональные возможности.

Использовать остальные диаграммы поведения для формирования индекса проекта представляется спорным решением. Данные диаграммы абсолютно уникальны, как по элементам, так и по их взаимодействию. При поиске архитектурно похожих решений не стоит цель обнаружить проект, уже решающий необходимые задачи. Отобранные проекты должны помочь оптимально сформировать основные модули, выбрать библиотеки, определить процессы, учесть ограничения и т.д.

Сравнение методологий проектирования представлено в таблице 1.2.

Таблица 1.2

Сравнение методологий проектирования

	IDEF	DFD	UML
Формат описания файла	отсутствует	отсутствует	XMI – универсальный формат представления диаграмм
Визуальные редакторы для диаграмм	Erwin: IDEF0, IDEF3, Ramus	ERWin, Visual Paradigm,	Visual Paradigm, StarUML, Visio и т.д.
Представление внутренней структуры проекта	IDEF1X, IDEF3	Data flow diagram	Class diagram, Deployment diagram, Component Diagram, Package Diagram, Composite Structure Diagram, Class diagram,
Внутренняя структура проекта в терминах ООП	отсутствует	отсутствует	Class diagram, Class diagram, Composite Structure Diagram,
Представление поведения программы	IDEF0, IDEF3, IDEF4, IDEF 5	Data flow diagram	Activity Diagram, Communication Diagram, Interaction Overview Diagram, State Machine Diagram, Sequence Diagram, Timing Diagram, Use Case Diagram
Дата последнего обновления нотации	Разные для каждой из диаграмм 1993-1997 годы	Отсутствует официальная спецификация	2.5.1 – декабрь 2017
Интеграция с исходным кодом	Интерпретация специалистами	Интерпретация специалистами	Автоматическая генерация иерархии классов, связей, полей и методов без реализации. Автоматическое обратное проектирование для получения диаграммы классов.

Исходя из цели и задач диссертационного исследования наибольший интерес представляют языки проектирования, которые позволяют извлекать максимально формализованные знания, схожие со знаниями, извлекаемыми из исходного кода.

Из анализа языков и средств проектирования можно сделать вывод об оптимальности использования в исследовании диаграмм, разработанных на языке UML. Язык UML имеет четко определенную спецификацию, и разрабатывающий ее коллектив OMG активен. Язык UML имеет самое широкое распространение среди сообщества разработчиков. Спецификация языка UML обновляется редко, но регулярно, существуют более частые минорные обновления отдельных документов спецификации. UML диаграммы можно создавать практически в любом средстве проектирования.

Язык UML содержит наиболее широкий набор диаграмм, содержащий как структурные диаграммы, так и диаграммы поведения. Наибольший интерес для исследования представляет именно диаграмма классов за счет полной и однозначной совместимости элементов с исходным кодом.

Метамодель языка UML обеспечивает в дальнейшем расширение функциональности диссертационного программного продукта за счет добавления большего числа диаграмм и элементов, так как при реализации поддержки дерева элементов языка UML добавление новых элементов представляет из себя тривиальную задачу.

Следует отметить, что достоинством языка UML является наличие четко формализованного языка описания и импорта/экспорта диаграмм – XMI. Наличие данного формата позволяет автоматизировано обрабатывать диаграммы, созданные в разных редакторах.

Знания из диаграмм, построенных с помощью других нотаций, так же можно извлекать и использовать с помощью программного продукта, разработанного в ходе диссертационного исследования. Альтернативные диаграммы можно экспортировать в XMI, но с помощью некоторых редакторов.

Конструирование

Результатом конструирования или разработки программы является исходный код, эволюционирующий от версии к версии, документация, создаваемая на каждом этапе, ряд внутренних текстовых документов, не

поддающихся формализации, но содержащих необходимые знания для разработчиков и пользователей, наборы данных для тестирования узких случаев и т.д.

Глобально можно разделить все приведенные выше документы на три категории:

- неструктурированные и не формализуемые текстовые документы: документация, комментарии, внутренние документы;
- исходный код, который строится по определенным формальным правилам и может быть обработан автоматизировано;
- временные ряды, порожденные версиями исходного кода, версиями текстовых документов и статистикой, накопленной по данному программному обеспечению в системе отслеживания ошибок и в системе контроля версий.

Документы первого типа могут быть обработаны интеллектуальными методами обработки [31] текстовой информации для индексации проекта в рамках предметной области.

Исходный код системы анализируется с помощью специального программного обеспечения, позволяющего представить знания в виде онтологии в формате OWL, идентичном формату знаний, извлеченных на этапе проектирования из документов в формате XMI.

Так как существует множество языков программирования, подходов к программированию, форматов приложений, платформ и фреймворков, решено было ограничить возможное многообразие рассматриваемых проектов до подмножества проектов, реализованных на объектно-ориентированном языке Java.

Данные, извлекаемые из систем контроля версий, удобно представлять в виде временных рядов. Такой вид представления информации позволяет использовать широкий аппарат обработки, анализа и прогнозирования временных рядов. Представление данных в виде временных рядов позволяет учитывать семантику внутренних связей между значениями показателей и

строить рекомендации на основе состояния проекта, описываемого последовательностью комбинаций показателей.

Тестирование, внедрение и сопровождение

На этапах тестирования, внедрения и сопровождения специфические источники знаний как правило не выделяются. Работа на каждом из перечисленных этапов состоит, в основном, из вербальных коммуникаций между специалистами, представителями заказчика и пользователями.

Источники знаний могут быть определены как текстовые документы, проектные диаграммы, исходный код и временные ряды, построенные в ходе жизненного цикла ПО на основе системы контроля версий.

Сформированное на ранних стадиях жизненного цикла онтологическое представление знаний позволит точно и однозначно определить значение терминов и связей между ними, и тем самым оптимизировать время, затрачиваемое на переговоры и уточнение информации.

1.2. Обзор основных форм представления знаний

Для решения сформулированных задач диссертационного исследования необходимо определить оптимальный подход к представлению знаний. Из цели исследования можно определить некоторые требования к способу представления знаний.

Представление знаний должно позволять описать обособленную предметную область. Выбранное представление не должно зависеть от знаний, которые не требуются для непосредственного решения производственных задач.

Современные крупные проекты ПАК преимущественно создаются в объектно-ориентированной парадигме. Формат представления знаний должен позволять максимально эффективно описывать иерархическую систему взаимосвязанных классов для выражения знаний о структуре проекта и взаимодействия объектов этих классов для представления знаний о поведении системы.

Формирование представления знаний в рамках производственного процесса никогда не прекращается. Выбранное представление знаний должно позволять быстро вносить изменения. Изменения будут вноситься не только на уровне утверждений о концептах, но и на уровне самих концептов.

При рассмотрении существующих производственных подходов к обособленному описанию предметной области, большинство разработчиков сходятся к использованию систем автоматизированного тестирования в рамках TDD. Автоматизированное тестирование знаний о системе должно так же быть обеспечено, например, за счет проверки хранилища знаний на согласованность с помощью машины логического вывода.

1.2.1. Логическая модель

Логическая модель, как вид представления знаний, рассматривает знания как высказывания и пропозиции о существующих в системе объектах и предлагает процедурный подход к выведению новых истинных знаний на основе заранее заданных аксиом.

Логическая модель представления знаний может быть сформулирована как [30]:

$$M =< T, П, A, B >, \text{где}$$

Т– множество базовых элементов модели, извлеченных из предметной области. Элемент принадлежит модели, только если некоторая процедура $Пр_1(t)$ выдаст положительный результат.

П – множество синтаксических правил, позволяющих формировать корректные утверждения из элементов Т. Модель содержит некоторую процедуру $Пр_2$ ($п(т_1,т_2…т_i)$), которая позволяет за конечное число шагов дать ответ на вопрос, является ли утверждение синтаксически правильным.

А – множество утверждений, являющихся аксиомами при конкретной интерпретации прикладной задачи.

В – множество правил логического вывода, применимых к утверждениям из множества А. Результаты применения правил логического вывода так же являются синтаксически верными утверждениями.

Если для модели имеется процедура определения выводимости любой корректной синтаксической последовательности на основе предложенных аксиом, то модель считается разрешимой.

Логическая модель представления знаний представляет собой попытку представить предметную область в виде набора утверждений, основанных на элементах предметной области. Подобный подход позволяет проверять новые полученные в ходе проектирования утверждения на корректность относительно базовых аксиом и утверждений, уже прошедших проверку на истинность, что позволяет следовать принципам DDD на уровне проектирования и представления знаний. Возможность контролировать непротиворечивость знаний, на основе которых ведется проектирование, позволит снизить количество ошибок, допускаемых при неверной трактовке предметной области.

К отрицательным аспектам применения логической модели можно отнести дополнительные затраты времени на формализацию знаний о системе и наличие в команде разработчиков инженера по знаниям, имеющего компетенцию по работе с машиной логического вывода. Наличие ограничения по компетенциям и накладные расходы по времени являются общими для всех моделей представления знаний.

Недостатком именно логической модели представления знаний является слабое представление структуры отдельного элемента предметной области. Элементы предметной области не могут быть объединены в иерархию, внутреннее строение элемента никак не учитывается, все взаимосвязи между элементами указываются в виде логических высказываний, для которых не применимы свойства рефлексивности, транзитивности, симметричности и т.д. Кроме того, логическая модель не позволяет реализовать разделение на классы и объекты характерные объектно-ориентированному стилю разработки ПО.

1.2.2. Семантические сети

Семантическая сеть представляет собой математическую абстракцию, передающую семантику предметной области, в виде графа, вершинами которого являются термины предметной области, связанные отношениями. Моделирование предметной области с помощью семантической сети позволяет отобразить гораздо большее количество разнообразных отношений над элементами предметной области. С помощью семантической сети можно задать иерархические, функциональные, логические, количественные отношения и т.д.

Модель семантической сети может быть представлена в виде [30]:

$$C =< T, O, B >, \textit{где}$$

T – множество терминов предметной области.

O – множество отношений, заданных в рамках сети.

B – множество экземпляров отношений, связывающих конкретные термины.

Семантические сети позволяют решать прикладные задачи посредством формирования и дальнейшего обхода сети по конкретному запросу. Семантическая сеть также может быть обработана с помощью машины логического вывода, что представляет собой продукционную модель.

Семантические сети создавались для решения конкретных прикладных задач и хорошо себя зарекомендовали как для формирования сложной системы отношений, так и способа визуализации представления знаний.

С помощью семантических сетей невозможно обособленно представить внутреннюю структуру сложного объекта и разнести статический и динамический уровни описания предметной области.

1.2.3. Продукционные модели

В продукционных моделях представления знаний объединяются идеи логических моделей и семантических сетей. Знания в продукционных моделях представляются в виде семантических сетей, над которыми

построена система логического вывода на основе продукций. Продукцию можно представить в виде [30]:

$$i =< Q; P; A \rightarrow B; N >, \text{где}$$

i – название или порядковый номер продукции, позволяющий производить обращение в системе продукций для последующего исполнения.

Q – параметр, определяющий область применения продукции. Данный параметр выполняет роль классификатора продукции и позволяет влиять на ход ее применения.

P – условие применимости продукции. Продукция может быть исполнена только в случае истинности логического выражения P.

$A \rightarrow B$ – ядро продукции, являющееся правилом ЕСЛИ A, ТО B в общем случае. Ядро продукции может быть расширено дополнением альтернативного действия в случае НЕ A или обозначать логическое следование. Особенности организации ядра продукции определяются разработчиками ПАК.

N – множество событий, наступающих в результате выполнения ядра продукции.

Продукционная модель представления знаний предполагает наличие множества продукций, семантической сети, служащей для хранения и визуализации знаний и системы исполнения продукций.

Продукционные модели являются первым этапом агрегации представления знаний в виде единой системы. Продукционные модели совмещают детальное выражение знаний с помощью семантической сети и представляют логический вывод в виде множества разветвленных правил, именуемых продукциями.

К недостаткам можно отнести нетривиальную уникальную систему управления выводом по продукциям, составляемую для каждой предметной области. Недостатки визуализации семантических сетей остаются также присущи и продукционным моделям. Результатом выполнения череды продукций является видоизмененная семантическая сеть, которую как

правило, довольно сложно интерпретировать и анализировать из-за уникальной системы управления продукциями.

1.2.4. Фреймы

Основные идеи фреймовой модели представления знаний были заимствованы при исследовании структуризации лингвистических и естественных систем структурирования знаний. Главной идеей фреймовой модели представления знаний является формирование сложной многоуровневой структуры фрейма. Фрейм может быть представлен с помощью следующей модели [80]:

$$F =< N, \{S_{st1}, S_{st2}, \dots, S_{sti}\}, \{S_{l1}, S_{l2}, \dots, S_{li}\}, \{S_{t1}, S_{s2}, \dots, S_{ti}\} >, \text{где}$$

N – наименование слота, позволяет выделять фрейм из множества фреймов в представлении знаний.

S_{sti} – слот, описывающий внутреннюю структуру фрейма. Данный слот в свою очередь содержит:

- название слота;
- указатель наследования. Определяет поведение слота при наследовании фреймов;
- указатель типа данных - содержит информацию о типе данных, размещаемых в слоте;
- значение слота;
- действия, выполняемые при изменении значения слота.

S_{li} – слот, определяющий связь с другими фреймами.

S_{ti} – слот, определяющий преобразования фреймовой модели.

Фреймовая модель представления знаний аналитически проработана и содержит множество различных модификаций. Фреймовая модель основана на естественной структуризации знаний человеком в виде системы классов и объектов. Представление знаний в виде фреймов позволяет выделить основные объекты предметной области и формализовать их внутреннюю структуру и поведение.

Фреймы чаще всего представляются в виде таблиц, состоящих из слотов, напоминающих таблицы реляционной базы данных, а слоты типа S_{li}, определяют связи между фреймами. Машины логического вывода могут быть применены к представлению знаний в виде фреймов лишь в случае построения дополнительной логики трансформации фреймовой модели в логические высказывания.

1.2.5. Онтологии

Развитием идеи представления знаний в виде семантических сетей послужила их глобализация и попытка объединить как можно больше знаний в одну сеть. Глобализация семантической сети невозможна за счет научной группы или даже организации, поэтому появился формат rdf. Формат rdf основан на xml, позволяет описывать знания в виде триплетов и был предложен для передачи семантики ресурса, доступного по глобальной сети Интернет. Основным преимуществом rdf была возможность обрабатывать rdf-документы автоматически, что делало семантику доступной для программного агента.

Прообразы онтологий, как средств описания предметной области, появились достаточно давно. Любая попытка структурировать знания о предметной области и аккумулировать их в виде книги или технологического документа, связанного гиперссылками, может рассматриваться как попытка формализации предметной области.

Основное определение онтологии дал Т.Груббер в 1993 году. Онтология – это точная спецификация концептуализации. Попытки сформулировать или уточнить понятие онтологии постоянно продолжаются. Для разработки объектно-ориентированного ПО лучше всего подходит определение представления знаний является данное A Gomez-Perez в 2004 году. Онтология – это формальная спецификация согласованной концептуализации.

Стремление уточнить понятие онтологии объясняется непрекращающимся процессом поиска модификации стандартов и

технологий, окружающих данный вид представления знаний. Каждый исследователь понимает под онтологией свое субъективное представление и выделяет важные для себя особенности науки и технологии, связанные с данным видом представления знаний.

Согласно работам [24] и [10], онтология представляет собой формальное представление структуры исследуемой предметной области:

$$O = \langle T, R, D \rangle,$$

где T – термины отражающие основные понятия предметной области, R - отношения между терминами, D определения этих понятий и отношений.

В онтологиях каждому понятию предметной области ставится в соответствие большое количество различных слов и выражений. Создаваемые таким образом языковые конструкции называются терминами онтологии [22].

Онтология, как правило, состоит из классов сущностей, соответствующих набору терминов предметной области, внутреннего устройства классов, заданного через свойства данных, связей между классами и высказываний [28], [58], [95].

Если попытаться сравнить онтологию с остальными наиболее распространенными способами представления знаний, то под онтологией можно понимать агрегацию всех удачных идей из предшествующих подходов.

Онтология позволяет:

- формировать иерархическую систему концептов и их индивидуалов (данное свойство изначально предлагалось в рамках семантических сетей и в объектно-ориентированном проектировании);
- создавать отношения между концептами и индивидуалами (семантические сети и тезаурусы позволяют осуществлять схожие действия над своими элементами);

- описывать внутреннюю структуру концептов и индивидуалов (возможность детально описывать внутреннюю структуру и поведение элементов впервые появилась на таком уровне в представлении знаний с помощью фреймов и в классах объектно-ориентированного проектирования);
- осуществлять логический вывод с помощью обособленных машин логического вывода (данная особенность была заимствована у логических моделей и модифицирована путем повышения гибкости за счет разделения зон ответственности между представлением знаний и выводом новых знаний).

Нельзя не отметить, что онтологическое представление знаний получило не только аналитическое развитие модели и методологии, но и было воплощено в виде многочисленных технических решений, к которым можно отнести:

- стандарты на языки описания онтологий rdf, а позднее owl;
- редакторы онтологий, позволяющие оперировать элементами пользовательского интерфейса без воздействия напрямую с owl или rdf файлами;
- машины логического вывода, адаптированные специально для онтологий форматов, указанных в стандартах;
- разнообразие диалектов описания онтологий на основе языка OWL;
- программные библиотеки, позволяющие реализовывать взаимодействие с онтологиями и машинами логического вывода (МЛВ) на уровне исходного кода.

Исходя из анализа моделей представления знаний, изложенного выше, было принято решение использовать в рамках диссертационного исследования онтологическое представление знаний. Подход к представлению знаний в виде онтологии позволяет агрегировать

преимущества всех существующих моделей представления знаний и использовать широкие инфраструктурные возможности модели.

1.3. Анализ и прогнозирование временных рядов, как инструмент управления формированием онтологий и ходом разработки ПАК

1.3.1. Подход к разработке инструмента управления ходом разработки ПАК на основе анализа и прогнозирования временных рядов

Отображение динамической информации в виде временных рядов является полезным, поскольку временные ряды с помощью накопленных методов и алгоритмов обработки позволяют моделировать большое количество самых разнообразных процессов. Лучше всего представлять с помощью временных рядов показатели, которые меняются с течением времени. Вследствие широкого распространения применения временных рядов существует и большое разнообразие методов и моделей для их представления, и что самое важное, их прогнозирования.

Для повышения качества автоматизированного проектирования ПАК, предлагается использовать прогнозирование временных рядов показателей в ходе работы над проектом за счет развития методов нечеткого моделирования и создания коллективов прогнозирующих методов.

В рамках задач исследования временные ряды рассматриваются как форма представления показателей процесса проектирования ПАК. В исследовании предлагается новый алгоритм, интегрирующий результаты прогнозирования коллектива методов, состоящего из классических методов экспоненциального сглаживания временных рядов и методов нечеткого экспоненциального сглаживания.

1.3.2. Формирование массива методов моделирования временных рядов

Классические методы экспоненциального сглаживания

Экспоненциальное сглаживание относится к адаптивным методам прогнозирования временных рядов. За счет наличия адаптивного механизма, простоты реализации и легкой интерпретации результата, экспоненциальное сглаживание играет важную роль в прогнозировании развития проектов разработки ПО. Высокое распространение получил метод автоматического прогнозирования с использованием набора моделей экспоненциального сглаживания. Экспоненциальное сглаживание относится к методам краткосрочного прогнозирования. В качестве набора моделей используется расширенная классификация моделей [87], [93].

В классификации представлены 5 вариантов тренда и 3 варианта сезонности. Определения методов приведены в таблице 1.3, в которой на пересечении строки тренда и столбца сезонности получается одна из моделей экспоненциального сглаживания. В практических задачах прогнозирования данные методы показали очень высокую эффективность.

Таблица 1.3

Расширенная классификация методов экспоненциального сглаживания

Тренд	Сезонность		
	N Отсутствует	**A** Аддитивная	**M** Мультипликативная
N Отсутствует	$S_t = \alpha X_t + (1-\alpha)S_{t-1}$ $\hat{X}_t(m) = S_t$	$S_t = \alpha(X_t - I_{t-p}) + (1-\alpha)S_{t-1}$ $I_t = \delta(X_t - S_t) + (1-\delta)I_{t-p}$ $\hat{X}_t(m) = S_t + I_{t-p+m}$	$S_t = \alpha(X_t / I_{t-p}) + (1-\alpha)S_{t-1}$ $I_t = \delta(X_t / S_t) + (1-\delta)I_{t-p}$ $\hat{X}_t(m) = S_t I_{t-p+m}$
	$S_t = S_{t-1} + \alpha e_t$ $\hat{X}_t(m) = S_t$	$S_t = S_{t-1} + \alpha e_t$ $I_t = I_{t-p} + \delta(1-\alpha)e_t$ $\hat{X}_t(m) = S_t + I_{t-p+m}$	$S_t = S_{t-1} + \alpha e_t / I_{t-p}$ $I_t = I_{t-p} + \delta(1-\alpha)e_t / S_t$ $\hat{X}_t(m) = S_t I_{t-p+m}$
A Аддитивный	$S_t = \alpha X_t + (1-\alpha)(S_{t-1} + T_{t-1})$ $T_t = \gamma(S_t - S_{t-1}) + (1-\gamma)T_{t-1}$ $\hat{X}_t(m) = S_t + mT_t$	$S_t = \alpha(X_t - I_{t-p}) + (1-\alpha)(S_{t-1} + T_{t-1})$ $T_t = \gamma(S_t - S_{t-1}) + (1-\gamma)T_{t-1}$ $I_t = \delta(X_t - S_t) + (1-\delta)I_{t-p}$ $\hat{X}_t(m) = S_t + mT_t + I_{t-p+m}$	$S_t = \alpha(X_t / I_{t-p}) + (1-\alpha)(S_{t-1} + T_{t-1})$ $T_t = \gamma(S_t - S_{t-1}) + (1-\gamma)T_{t-1}$ $I_t = \delta(X_t / S_t) + (1-\delta)I_{t-p}$ $\hat{X}_t(m) = (S_t + mT_t)I_{t-p+m}$
	$S_t = S_{t-1} + T_{t-1} + \alpha e_t$ $T_t = T_{t-1} + \alpha\gamma e_t$ $\hat{X}_t(m) = S_t + mT_t$	$S_t = S_{t-1} + T_{t-1} + \alpha e_t$ $T_t = T_{t-1} + \alpha\gamma e_t$ $I_t = I_{t-p} + \delta(1-\alpha)e_t$ $\hat{X}_t(m) = S_t + mT_t + I_{t-p+m}$	$S_t = S_{t-1} + T_{t-1} + \alpha e_t / I_{t-p}$ $T_t = T_{t-1} + \alpha\gamma e_t / I_{t-p}$ $I_t = I_{t-p} + \delta(1-\alpha)e_t / S_t$ $\hat{X}_t(m) = (S_t + mT_t)I_{t-p+m}$
DA Аддитивный с демпингом	$S_t = \alpha X_t + (1-\alpha)(S_{t-1} + \phi T_{t-1})$ $T_t = \gamma(S_t - S_{t-1}) + (1-\gamma)\phi T_{t-1}$ $\hat{X}_t(m) = S_t + \sum_{i=1}^{m} \phi^i T_t$	$S_t = \alpha(X_t - I_{t-p}) + (1-\alpha)(S_{t-1} + \phi T_{t-1})$ $T_t = \gamma(S_t - S_{t-1}) + (1-\gamma)\phi T_{t-1}$ $I_t = \delta(X_t - S_t) + (1-\delta)I_{t-p}$ $\hat{X}_t(m) = S_t + \sum_{i=1}^{m} \phi^i T_t + I_{t-p+m}$	$S_t = \alpha(X_t / I_{t-p}) + (1-\alpha)(S_{t-1} + \phi T_{t-1})$ $T_t = \gamma(S_t - S_{t-1}) + (1-\gamma)\phi T_{t-1}$ $I_t = \delta(X_t / S_t) + (1-\delta)I_{t-p}$ $\hat{X}_t(m) = (S_t + \sum_{i=1}^{m} \phi^i T_t)I_{t-p+m}$
	$S_t = S_{t-1} + \phi T_{t-1} + \alpha e_t$ $T_t = \phi T_{t-1} + \alpha\gamma e_t$ $\hat{X}_t(m) = S_t + \sum_{i=1}^{m} \phi^i T_t$	$S_t = S_{t-1} + \phi T_{t-1} + \alpha e_t$ $T_t = \phi T_{t-1} + \alpha\gamma e_t$ $I_t = I_{t-p} + \delta(1-\alpha)e_t$ $\hat{X}_t(m) = S_t + \sum_{i=1}^{m} \phi^i T_t + I_{t-p+m}$	$S_t = S_{t-1} + \phi T_{t-1} + \alpha e_t / I_{t-p}$ $T_t = \phi T_{t-1} + \alpha\gamma e_t / I_{t-p}$ $I_t = I_{t-p} + \delta(1-\alpha)e_t / S_t$ $\hat{X}_t(m) = (S_t + \sum_{i=1}^{m} \phi^i T_t)I_{t-p+m}$
M Мультипли-кативный	$S_t = \alpha X_t + (1-\alpha)(S_{t-1}R_{t-1})$ $R_t = \gamma(S_t / S_{t-1}) + (1-\gamma)R_{t-1}$ $\hat{X}_t(m) = S_t R_t^m$	$S_t = \alpha(X_t - I_{t-p}) + (1-\alpha)S_{t-1}R_{t-1}$ $R_t = \gamma(S_t / S_{t-1}) + (1-\gamma)R_{t-1}$ $I_t = \delta(X_t - S_t) + (1-\delta)I_{t-p}$ $\hat{X}_t(m) = S_t R_t^m + I_{t-p+m}$	$S_t = \alpha(X_t / I_{t-p}) + (1-\alpha)S_{t-1}R_{t-1}$ $R_t = \gamma(S_t / S_{t-1}) + (1-\gamma)R_{t-1}$ $I_t = \delta(X_t / S_t) + (1-\delta)I_{t-p}$ $\hat{X}_t(m) = (S_t R_t^m)I_{t-p+m}$
	$S_t = S_{t-1}R_{t-1} + \alpha e_t$ $R_t = R_{t-1} + \alpha\gamma e_t / S_{t-1}$ $\hat{X}_t(m) = S_t R_t^m$	$S_t = S_{t-1}R_{t-1} + \alpha e_t$ $R_t = R_{t+1} + \alpha\gamma e_t / S_{t-1}$ $I_t = I_{t-p} + \delta(1-\alpha)e_t$ $\hat{X}_t(m) = S_t R_t^m + I_{t-p+m}$	$S_t = S_{t-1}R_{t-1} + \alpha e_t / I_{t-p}$ $R_t = R_{t-1} + (\alpha\gamma e_t / S_{t-1}) / I_{t-p}$ $I_t = I_{t-p} + \delta(1-\alpha)e_t / S_t$ $\hat{X}_t(m) = (S_t R_t^m)I_{t-p+m}$
DM Мультипли-кативный с демпингом	$S_t = \alpha X_t + (1-\alpha)(S_{t-1}R_{t-1}^{\phi})$ $R_t = \gamma(S_t / S_{t-1}) + (1-\gamma)R_{t-1}^{\phi}$ $\hat{X}_t(m) = S_t R_t^{\sum_{i=1}^{m}\phi^i}$	$S_t = \alpha(X_t - I_{t-p}) + (1-\alpha)S_{t-1}R_{t-1}^{\phi}$ $R_t = \gamma(S_t / S_{t-1}) + (1-\gamma)R_{t-1}^{\phi}$ $I_t = \delta(X_t - S_t) + (1-\delta)I_{t-p}$ $\hat{X}_t(m) = S_t R_t^{\sum_{i=1}^{m}\phi^i} + I_{t-p-m}$	$S_t = \alpha(X_t / I_{t-p}) + (1-\alpha)(S_{t-1}R_{t-1}^{\phi})$ $R_t = \gamma(S_t / S_{t-1}) + (1-\gamma)R_{t-1}^{\phi}$ $I_t = \delta(X_t / S_t) + (1-\delta)I_{t-1}$ $\hat{X}_t(m) = (S_t R_t^{\sum_{i=1}^{m}\phi^i})I_{t-p+m}$
	$S_t = S_{t-1}R_{t-1}^{\phi} + \alpha e_t$ $R_t = R_{t-1}^{\phi} + \alpha\gamma e_t / S_{t-1}$ $\hat{X}_t(m) = S_t R_t^{\sum_{i=1}^{m}\phi^i}$	$S_t = S_{t-1}R_{t-1}^{\phi} + \alpha e_t$ $R_t = R_{t-1}^{\phi} + \alpha\gamma e_t / S_{t-1}$ $I_t = I_{t-p} + \delta(1-\alpha)e_t$ $\hat{X}_t(m) = S_t R_t^{\sum_{i=1}^{m}\phi^i} + I_{t-p+m}$	$S_t = S_{t-1}R_{t-1}^{\phi} + \alpha e_t / I_{t-p}$ $R_t = R_{t-1}^{\phi} + (\alpha\gamma e_t / S_{t-1}) / I_{t-p}$ $I_t = I_{t-p} + \delta(1-\alpha)e_t / S_t$ $\hat{X}_t(m) = (S_t R_t^{\sum_{i=1}^{m}\phi^i})I_{t-p+m}$

В приведенной классификации:

α - параметр сглаживания временного ряда;

γ - параметр сглаживания для тренда;

δ - параметр сглаживания сезонной компоненты;

ϕ - параметр, определяющий величину демпинга;

S_t – значение сглаженного ряда в момент времени t;

T_t - значение сглаженного аддитивного тренда в момент времени t;

R_t - значение сглаженного мультипликативного тренда в момент времени t;

I_t - значение сглаженной сезонной компоненты в момент времени t;

X_t - значение временного ряда в момент времени t с учетом компонента тренда и сезонности;

m - количество периодов прогнозирования;

p - количество периодов в сезонном цикле;

$\hat{X}_t(m)$ - прогноз на m периодов от t;

e_t - ошибка одношагового прогноза: $e_t = X_t - \hat{X}_{t-1}$.

В соответствии с данной классификацией были реализованы 15 методов экспоненциального сглаживания в рамках модуля интеллектуального анализа. Оптимизация параметров осуществляется с помощью подбора значения из промежутка, определяемого пользователем. Промежуток значения параметра выбирается отдельно для каждого метода, поскольку с увеличением количества параметров нелинейно растет время подбора оптимального набора параметров.

Интеграция нечеткого моделирования и экспоненциального сглаживания

В статье [56] авторы развивают подход Сонга и Чиссома (1993) [92] по нечеткому моделированию временных рядов. Как и в оригинальном методе моделирование выполняется следующим образом:

Шаг 1.

Определяется универсум для значений временного ряда и задаются интервалы для нечеткого разбиения. Универсум как правило может быть определен как:

$$U = [U_{start}, U_{end}],$$

$$U_{start} = U_{min} - D_{min},$$

$$U_{end} = U_{max} - D_{max},$$

где U_{min} и U_{max} - минимальное и максимальное значение временного ряда соответственно, а D_{min}и D_{max} - два целых положительных числа, задаваемые экспертом предметной области. После того как определена длина интервала разбиения, универсум может быть разделен на части равной длины.

Шаг 2.

Определяются нечеткие множества, основанные на универсуме и исторических данных временного ряда.

Шаг 3.

Производится фаззификация исторических данных по следующему правилу: если значение временного ряда принадлежит нечеткому множеству A_j, и максимальная степень принадлежности этого значения относится к нечеткому множеству A_j, то нечеткое значение A_j.

Шаг 4.

Устанавливаются логические отношения, и выполняется их группировка по текущим состояниям фаззифицированных данных.

Шаг 5.

Строится прогноз. Пусть $F(t-1) = A_i$.

Вариант 1: Существует только одно нечеткое логическое отношение. Если $A_i \rightarrow A_j$то F(t) равно A_j.

Вариант 2: Если $A_i \rightarrow A_i, A_j, \dots, A_k$, то F(f) равно$A_i, A_j, \dots, A_k$.

Шаг 6.

Дефаззификация. Применяется центроидный метод для получения числовых результатов.

Хуанг и др. [66] предложили метод прогнозирования, который агрегирует глобальную информацию нечетких отношений и локальную информацию последних колебаний для расчета прогнозируемого значения.

$$Forecasted\_value = w_g \times Glob\_info + w_l \times Local\_info,$$

где w_gи w_l – адаптивные веса, и $w_g + w_l = 1, 0 \leq w_{g,} w_l \leq 1$.

Glob_info - это глобальная информация. Если существует система нечетких логических отношений $A_{t-1} \rightarrow A_{t1}, A_{t2}, \dots, A_{tk}$, и $m_{t1}, m_{t2}, \dots, m_{tk}$– средние значения лингвистических переменных $A_{t1}, A_{t2}, \dots, A_{tk}$ соответственно, то:

$$Glob\_info = \frac{m_{t1}+m_{t2}+\cdots+m_{tk}}{k},$$

В свою очередь Local_info определяется как:

$$Local\_info = bs_t + \frac{be_t - bs_t}{2} + \frac{m_t - m_{t-1}}{m_t + m_{t-1}},$$

где be_tи bs_tесть начальная и конечная точки интервала $u_t = [bs_t, be_t]$, m_tи m_{t-1} – средние значения лингвистических переменных A_t и A_{t-1}.

По аналогии с методами экспоненциального сглаживания было предложено в качестве весов w_gи w_l использовать параметр w.

$$Forecasted\_value = w \times Glob\_info + (1 - w) \times Local\_info.$$

Для применения данного подхода к классическим методам экспоненциального сглаживания, необходимо определить, как выделять сезонную и трендовую составляющие. Было предложено поставить в соответствие трендовой составляющей модели величину Glob_info. Сезонную компоненту было предложено определять по системе нечеткого логического вывода, предварительно задав период p.

Таким образом, стало возможным осуществить реализацию 4-х классических методов экспоненциального сглаживания через сочетания этих методов с комбинациями трендовой и сезонной компонентой.

Модель без тренда и сезонности

$$Forecasted\_value = Local\_info.$$

В данной модели подразумевается, что операция фаззификации заменяет собой сглаживание.

Модель без тренда с аддитивной сезонной компонентой периода p

$$Forecasted\_value = \gamma * Local\_info \ + (1 - \gamma) * Season\_info.$$

Модель с аддитивным трендом без сезонной компоненты

$$Forecasted_{value} = \delta * Local_{info} + (1 - \delta) * Glob_{info}.$$

Модель с аддитивным трендом и аддитивной сезонной компонентой периода p

$$Forecasted\_value = \delta * \gamma * Local\_info \ + (1 - \gamma) * Season\_info +$$
$$+(1 - \delta) * Glob\_info.$$

Мультипликативные методы не были рассмотрены так как большинство задач из наборов для тестирования и отладки моделей не предполагают подобного поведения временных рядов. Мультипликативные методы ориентированы в первую очередь на ряды, характеризующимися быстрым и постоянным ростом. Подобные процессы редко можно эффективно прогнозировать с помощью моделей, не учитывающих особенности предметной области.

Нечеткое моделирование ВР

По аналогии с классическим представлением временного ряда, для нечеткого временного ряда Reinhard Viertel [96] представляет экспоненциальное сглаживание в виде:

$$\tilde{y}_t = \beta \cdot \tilde{y}_{t-1} \oplus (1 - \beta) \ \cdot \tilde{x}_t,$$

где - β параметр сглаживания, $\tilde{y}_t$-сглаженный ряд, $\tilde{x}_t$- нечеткий ряд.

В рамках исследования рассматриваются нечеткие компоненты, формирующие нечеткий временной ряд:

$$\tilde{x}_t = \tilde{m}_t \oplus \tilde{s}_t,$$

где $\tilde{x}_t$ - нечеткий ряд, $\tilde{m}_t$ – нечеткий тренд, $\tilde{s}_t$–нечеткая сезонная компонента. В таком случае становится возможным построение моделей, аналогичных описанным классическим методам из классификации с той

лишь разницей, что все операции будут производиться над нечеткими множествами.

Тогда, формальное представление методов будет следующим.

- Модель без тренда и сезонности: $\tilde{x}_t = \tilde{y}_t$;
- Аддитивная сезонность: $\tilde{x}_t = \tilde{y}_t \oplus \tilde{s}_t$;
- Мультипликативная сезонность: $\tilde{x}_t = \tilde{y}_t \odot \tilde{s}_t$.
- Аддитивный тренд: $\tilde{x}_t = \tilde{y}_t \oplus \tilde{m}_t$.
- Мультипликативный тренд: $\tilde{x}_t = \tilde{y}_t \odot \tilde{m}_t$.
- Аддитивный тренд, аддитивная сезонность: $\tilde{x}_t = \tilde{y}_t \oplus \tilde{m}_t \oplus \tilde{s}_t$.
- Аддитивный тренд, мультипликативная сезонность:

$$\tilde{x}_t = (\tilde{y}_t \oplus \tilde{m}_t) \odot \tilde{s}_t;$$

- Мультипликативный тренд, аддитивная сезонность:

$$\tilde{x}_t = \tilde{y}_t \odot \tilde{m}_t \oplus \tilde{s}_t;$$

- Мультипликативный тренд, мультипликативная сезонность:

$$\tilde{x}_t = \tilde{y}_t \odot \tilde{m}_t \odot \tilde{s}_t.$$

Для проведения сглаживания нечеткого временного ряда необходимо определить операции над нечеткими множествами. Чтобы реализовать нечеткие модификации всех методов, необходимы операции: сложения множеств, разности, умножения, деления, умножения на число.

Операции над нечеткими множествами представляют собой операции над границами и вершинами функций принадлежности. В данной работе были выбраны треугольные симметричные функции принадлежности. Производить операции необходимо над границами этих множеств, поскольку вершину можно будет вычислить.

Реализация нечеткого моделирования ВР по Мамдани

Метод экстраполяционного прогнозирования DSA [69] создан с помощью нечеткой логики на основе методов экстраполяции, введенных Сонгом и Чиссомом (Song-Chissom) [92]. Форматом входных значений DSA алгоритма являются хронологические значения в виде временного ряда некой

характеристики исследуемого объекта. Для корректной работы модели необходимо составить репрезентативный обучающий набор значений.

В результате разработки фаззификацированного метода DSA были разработаны две новых процедуры, описывающие функцию принадлежности и вычисление степени перекрытия между нечеткими множествами. Данные процедуры являются развитием методов Song-Chissom. По средствам данных процедур можно управлять дополнительными параметрами модели метода DSA для улучшения точности прогноза. В модели DSA треугольные функции принадлежности используются для всех нечетких множеств, причем, степени перекрытия для последовательных наборов данных конкретной модели идентичны.

Основным аспектом в процедуре фаззификации метода DSA является универсальное преобразование, которое отражает расширение диапазона исторических значений временного ряда. В ходе процедуры фаззификации метода DSA минимальное значение диапазона сократилось, а максимальное увеличилось. Следовательно, изменилось и среднее значение абсолютной разности между значениями последовательных периодов временного ряда. Это обеспечивает метод DSA способностью производить внутренние, а также внешние прогнозы, которые отражают рост, либо спад показателя временного ряда за рамками обучающего набора значений показателя.

В методе DSA нечеткие множества определяются на универсальном множестве, которое служит областью входных и выходных значений. В то время как в большинстве нечетких методов результатом является набор лингвистических меток. В методе DSA достаточно одной индексированной метки. Низкие значения индекса связаны с низкими значениями переменной, в то время как индексы с высокими значениями связаны с низкими значениями переменной. Поэтому нечеткие множества были обозначены как $A_i, i \in [1; n]$, где n - количество нечетких множеств для отобранных моделей. Минимальное количество нечетких множеств равно двум, а одно нечеткое множество порождает прогноз в виде горизонтальной линии.

Нечеткая модификация метода DSA позволяет определить бесконечное количество нечетких множеств. В методе DSA, в отличие от более ранних нечетких методов, количество нечетких множеств, определенных на универсуме, считается параметром модели, который может быть использован для улучшения точности прогноза. Экспериментально в ходе диссертационного исследования было найдено оптимальное количество нечетких множеств, равное семи, оно соответствует оптимальному количеству нечетких множеств, предлагаемому в исследовании [42]. Данное количество нечетких множеств позволяет интерпретировать состояние проекта на основе обработанных данных, извлекаемых из системы контроля версий проекта, таких как исходный код, проектные диаграммы и онтологии в формате owl.

Границы интервалов нечетких множеств определяются экспертом. Установление границ происходит посредством присваивания нечеткой метки множеству значений из обучающей выборки. В классическом DSA методе интервалы принадлежности определяются для каждого нечеткого множества, и каждый интервал связан с определенной степенью принадлежности в диапазоне [0,1].

На заключительном этапе фаззификации используется набор правил Если-То, позволяющий присвоить метку одному из нечетких множеств каждого выделенного участка ВР из обучающей выборки.

Пример правила фаззицикации №1:

ЕСЛИ среди значений функций принадлежности для текущего, наблюдаемого участка временного ряда только одно больше 0, ТО для данного участка нечеткая лингвистическая метка определяется однозначно.

Пример правила фаззицикации №2:

ЕСЛИ среди значений функций принадлежности для текущего, наблюдаемого участка временного ряда несколько больше 0, ТО для данному участку ставится в соответствие нечеткая лингвистическая метка нечеткого множества с максимальным значением функции принадлежности.

В модуле вывода DSA метода тоже используется набор правил Если-То, чтобы сделать выводы о взаимосвязи между входными и выходными нечеткими лингвистическими переменными. Условие и следствие каждого правила является сопоставлением текущего и будущего состояния характеристики, описываемой временным рядом. Совокупность пар множеств представляет собой нечеткую модель временного ряда.

При дефаззификации значений прогноза используется модуль вывода. Для преобразования нечеткого прогноза, состоящего из набора нечетких лингвистических меток в скалярный прогноз, используется метод центра тяжести. Процедура дефаззификации может быть описана следующим набором правил:

Правило дефаззификации №1:

ЕСЛИ нечеткий прогноз построен на основе одного и только одного нечеткого множества, ТО скалярный прогноз есть одна из точек, в которой функция принадлежности нечеткого множества достигает своего максимального значения. В случае, если невозможно однозначно за конечное число шагов определить точку, в которой функция принадлежности нечеткого множества достигает максимального значения, то данная точка определяется по методу, определяемому конкретной функцией принадлежности, например, по методу центра тяжести.

Правило дефаззификации №2:

Если существует два или более нечетких множеств при построении нечеткого прогноза, ТО скалярный прогноз является центром среди всех максимальных значений функций принадлежности нечетких множеств.

1.3.3. Оценка методов прогнозирования

Для оценивания качества результата прогнозирования временного ряда используются информационные критерии. Информационные критерии позволяют в численном виде выразить степень корректности прогноза временного ряда. В ходе диссертационного исследования был разработан

модуль оценки качества результатов прогнозирования, в котором реализован расчет оценок прогноза по следующим информационным критериям:

- информационный критерий Акаике (AIC) [37];
- Байесовский информационный критерий (BIC) [91];
- модифицированный критерий Акаике (AICc) [48], для временных рядов с малым объемом обучающей выборки;
- Symmetric mean absolute percentage error *SMAPE* [78].

Каждый из перечисленных критериев по-разному интерпретирует значение функции правдоподобия, вычисляющей ошибку прогнозирования. Значение функции правдоподобия определяется как отклонение прогнозного значения от эталонного значения с учетом количества параметров, определяющих модель.

Ошибка прогнозирования является основной характеристикой прогноза. Чем меньше значение ошибки, тем выше оценка качества прогнозирования предлагаемой модели.

Количество параметров модели характеризует ее сложность. Все реализованные в модуле оценки качества прогноза критерии определяют обратную зависимость качества прогноза от количества параметров модели. Чем большим количеством параметров определяется модель, тем сложнее и дольше производить процесс настройки значении набора параметров. Результат прогнозирования может сильно изменятся при изменении значения любого из параметров, из чего можно сделать вывод о существовании сложных нелинейных зависимостей между параметрами модели, построенной для каждого конкретного ряда. Подобные зависимости приводят к необходимости перебора возможных наборов параметров, что существенно увеличивает время, затрачиваемое на создание модели и уменьшает стабильность результатов прогнозирования, как для временного ряда, из которого была отобрана обучающая выборка, так и для временных рядов, имеющих схожее поведение.

В сложной модели как правило невозможно выделить трендовую и сезонную составляющие. Связь функции расчета прогноза модели с семантикой прогнозируемого процесса невозможно отследить. Слишком сложные функции, лучше остальных подходящие под известные значения временного ряда, часто ведут себя непредсказуемо за пределами известного отрезка временного ряда.

Информационный критерий Акаике

Информационный критерий Акаике [37] позволяет оценивать модели временных рядов и рассчитывается по следующей формуле:

$$AIC = NLog\left(\sum_{i=1}^{N}(y_i - f_i)^2\right) + 2 * K,$$

где N - длина временного ряда, y_i - значение временного ряда в i-тый момент времени, f_i - результат прогнозирования модели в i-тый момент времени, K -количество параметров модели.

Модифицированный критерий Акаике

Модификация критерия Акаике [48] используется для рядов малой длины, с соотношением $\frac{N}{K} < 40$.

$$AICc = NLog\left(\sum_{i=1}^{N}(y_i - f_i)^2\right) + \frac{2KN}{N - K - 1}.$$

Байесовский информационный критерий

Байесовский информационный критерий [91], отличается от информационного критерия Акаике тем, что качество прогноза сильнее зависит от количества параметров модели.

$$AIC = NLog\left(\sum_{i=1}^{N}(y_i - f_i)^2\right) + Log(N) * K.$$

Средняя абсолютная процентная ошибка (MAPE информационный критерий)

Данный критерий принят за стандарт в сообществе исследователей, занимающихся прогнозированием временных рядов и используется в качестве основного при определении качества прогнозирования каждой новой предлагаемой модели. Отличительной особенностью данного критерия является отсутствие ухудшения оценки качества прогноза от количества параметров, определяющих модель.

$$MAPE = \frac{\left(\sum_{i=1}^{N} \frac{|y_i - f_i|}{|y_i + f_i|/_2} \right)}{N} * 100\%.$$

1.4. Интеграция онтологического представления знаний в процессе создания ПАК

Разработка онтологии весьма долгий и ресурсоемкий процесс, требующий привлечения специалистов, обладающих компетенциями в следующих направлениях:

- предметной области;
- области онтологического инжиниринга;
- программном обеспечении.

Компетенция в области онтологического инжиниринга предполагает применение:

- инструментов редактирования онтологий;
- языка интерпретации онтологий;
- машины логического вывода.

Есть много работ, посвященных интеграции онтологий в процесс разработки программного обеспечения. Существует целый подход к разработке, основанный на предметной области, известной как domain driven development [51], [52], [84], [98]. Попытки интегрировать онтологии в

разработку программного обеспечения осуществлялись на разных уровнях: технические документы [57] [73], [74], [82], поддержка и тестирование исходного кода [64], [65], [79], [89], диаграммы UML [38], [45], [76], [103].

Существуют программные системы, решающие схожие задачи для передачи концептуальных моделей в базу знаний о производстве [23] с использованием собственной нотации представления продуктивного знания [13]. Альтернативные подходы к преобразованию диаграмм в формате UML в онтологии в формате OWL представлены в статьях [38], [55], [103]. Авторы позиционируют трансформацию как систему для передачи знаний из диаграмм в онтологии. Основным отличием похода данного диссертационного исследования является постановка задачи, предполагающая обратную совместимость результата трансформации знаний. Проекты на крупных предприятиях имеют долгую историю и могут поддерживаться и расширяться годами. Поэтому необходимо обеспечить постоянную связь между концептуальными моделями и формой представления знаний. Такой подход позволит избежать наиболее дорогостоящих ошибок при разработке проектов и обеспечить их семантическую согласованность. Настоящее исследование представляет собой попытку произвести интеграцию прикладной онтологии в процесс разработки программного обеспечения на уровне концептуальных моделей. Таким образом, нормативная документация [27], [32] и исходный код должны быть связаны с онтологией. Это позволит автоматически обнаруживать конфликты и несоответствия между проектами в одной предметной области.

Создание общей онтологии предметной области в отрыве от задач автоматизации и разработки программных продуктов менее эффективно, чем создание онтологии с учетом аспектов автоматизации конкретной предметной области. Специалисты по онтологическому инжинирингу создают онтологию конкретной предметной области, используя знания, полученные от экспертов. Разработчики программного обеспечения

заинтересованы в быстром освоении знаний из предметной области и в проверке применяемых ими решений.

Основной проблемой использования онтологий в разработке программного обеспечения остаются высокие требования к знаниям внутреннего устройства онтологий и их возможностей. Онтологии составляются на основании знаний из предметных областей без учета особенностей архитектуры и возможностей программного обеспечения.

Важность формализации концептов проблемной области для разработки программных продуктов и программно-аппаратных комплексов привела к появлению многочисленных методологий разработки, языков проектирования, правил и стилей оформления исходного кода, систем контроля версий.

Исходя из анализа современного состояния сферы информационных технологий в рамках автоматизированного проектирования, приведенного в пункте 1.1., в качестве основных источников знаний были выделены:

- UML-диаграммы, создаваемые при проектировании;
- исходный код, модифицируемый на протяжении всего жизненного цикла проекта;
- временные ряды, формируемые на основе информации, получаемой из системы контроля версий.

В каждом источнике знаний формат представления первичной информации сильно разнится, вследствие чего невозможно работать со знаниями с помощью единого инструмента или общей методологии напрямую.

В качестве наиболее подходящего способа хранения знаний в пункте 1.2 было выбрано онтологическое представление, а именно формат OWL. Одной из задач диссертационного исследования является формирование T-box онтологии, позволяющей работать со знаниями, полученными из каждого источника знаний единообразно.

Средства UML позволяют разработать проект создаваемой системы, содержащий концептуальные элементы, такие как системные функции и производственные процессы. Диаграммы на языке UML применимы к описанию конкретных особенностей системы: классов, составляющих архитектуру системы, таблиц и связей, описывающих схему базы данных, АРМ-операторов и серверов для создания схемы физического размещения. Язык UML позволяет описать разные аспекты системы, абстрагируясь от реализации, но при этом сохраняя индивидуальные черты. Именно поэтому в основу T-box разрабатываемой онтологии была положена мета-схема языка UML.

Представление версий исходного кода в системе контроля версий позволяет организовать управление разработкой на основе анализа показателей (метрик) проекта, представленных в виде временных рядов.

Наряду с временными рядами показателей, характеризующих работу над исходным кодом, возможно анализировать временные ряды формирования проекта: количества сущностей (концептов), отношений, свойств, диаграмм. Перечисленные временные ряды позволяют расширить возможности анализа состояния проекта с помощью знаний, извлеченных на стадии проектирования и предотвратить смысловые ошибки проектирования.

Так как для крупных программных и программно-аппаратных комплексов прогнозирование является повторяющимся этапом жизненного цикла, анализ знаний, извлекаемых на данном этапе, также представляет интерес в ходе автоматизированного проектирования. Целесообразно осуществить модификацию известных методов анализа и прогнозирования временных рядов и предложить методы, подходящие к задаче автоматизированного проектирования программных и программно-аппаратных комплексов.

1.5. Постановка цели и задач исследования

Цель диссертационной работы

Целью диссертационной работы является снижение временных затрат на этапах проектирования и разработки программных и программно-аппаратных комплексов (ПАК).

Предполагается достичь цель за счет повторного использования проектных решений с помощью разработки и реализации моделей и алгоритмов поиска структурно-семантически схожих проектов по унифицированному онтологическому представлению знаний о предметной области, составленному на основе знаний, извлекаемых из проектных документов и формирования рекомендаций по управлению ходом развития проекта ПАК на основе интеграции классических и нечетких моделей прогнозирования временных рядов.

Задачи исследования

В соответствии с целью работы актуальными являются следующие задачи диссертационного исследования.

- Провести сравнительный анализ современных методов, алгоритмов и систем поиска программных проектов схожих по архитектуре и предметной области.
- Провести сравнительный анализ современных средств представления проектных диаграмм в интеллектуальном проектном репозитории.
- Разработать формальную модель программного комплекса, определяемую прикладной онтологией, основанной на элементах объектно-ориентированных языков программирования с учетом жизненного цикла.
- Разработать методику извлечения, формализации и использования знаний, извлеченных из систем контроля версий в рамках онтологического представления знаний.

- Разработать методику и алгоритм трансляции проектных диаграмм и исходного кода в онтологическое представление проекта в рамках информационного обеспечения САПР.
- Разработать способ управления автоматизированным проектированием проектных диаграмм проекта на основе прогнозирования временных рядов, выражающих развитие концептуальных диаграмм.
- Разработать программные средства, позволяющие структурировать проекты в рамках экспертной выборки или репозитория крупного предприятия.
- Исследовать возможности мер структурного подобия для эффективного отбора прототипа проекта.
- Провести вычислительные эксперименты, позволяющие оценить эффективность предложенных моделей и алгоритмов в процессе проведения концептуального проектирования ПАК.
- Внедрить результаты исследований в практику процесса проектирования технических систем предприятий.

Глава 2 Модели и методы обработки электронных документов и показателей программно-аппаратных комплексов в автоматизированном проектировании

2.1. Онтологическая модель автоматизированного проектирования ПАК

Программный или программно-аппаратный комплекс, позволяющий автоматизировать задачи в рамках определенной предметной области, создается итерационно. С каждой новой итерацией выделяются новые требования, что приводит к росту количества знаний о предметной области и знаний об использованных инженерных решениях.

Наиболее распространенным способом сохранения знаний о системе является распространение и дублирование знаний среди разработчиков. Подобный подход позволяет обезопасить систему от потери знаний при непредвиденных обстоятельствах, но не решить проблему полностью. Распространение знаний среди работников происходит во время обсуждения и взаимодействия, что существенно расходует время наиболее эффективных работников.

В случае, если по той или иной причине нельзя получить знания по средствам очного общения, происходит процесс получения нужного знания за счет изучения технической документации или непосредственной работы с системой. Получение знаний о комплексе из технической документации и опыта работы с системой приводит к неточностям и ошибкам в его восприятии.

Все вышеперечисленные проблемы можно решить с помощью создания хранилища знаний. Хранилище должно позволять:

- добавлять новые знания;
- редактировать существующие знания;
- предоставлять одновременный доступ множеству работников;

- представлять знания в удобной форме;
- осуществлять проверку на непротиворечивость знаний.

Информационное содержание хранилища концептуально может быть представлено онтологией или несколькими онтологиями. Системы хранения знаний и, в частности, онтологии предоставляют широкие возможности по управлению знаниями, вполне достаточные для управления знаниями о программном продукте, но требуют специфических навыков. Онтологии хранят знания с помощью утверждений двух типов:

T-Box – множество высказываний, определяющих отношения между классами индивидуалов;

A-Box – множество высказываний, определяющих отношения между индивидуалами.

В форме онтологии необходимо хранить знания непосредственно о предметной области системы и о программном или программно-аппаратном комплексе, построенном на базе модели предметной области, содержащей в том числе ограничения и условности. Общие знания о предметной области заносятся в систему экспертами при помощи специалистов по онтологическому инжинирингу.

Знания о реализованных предприятиями ПАК также могут храниться в виде онтологии. Формализация принятых решений по каждому из ПАК позволит существенно ускорить процесс проектирования нового продукта в предметной области, описанной ранее в виде онтологии предметной области. Подобное разделение позволит поддерживать знания о программной реализации каждого ПАК с учетом предметной области, к которой он относится.

Онтология в формате OWL [86] была выбрана для хранения UML диаграмм классов, так как формат OWL обладает на данный момент наибольшей выразительной, синтаксической силой для представления знаний в сложных предметных областях. Элементы диаграммы классов переносятся в онтологию в виде концептов с учетом их семантики. Семантика всех

диаграмм складывается не только из семантики входящих в нее элементов, но и из семантики их взаимодействия. Ввиду сложной организации знаний в UML диаграммах формального переноса элементов недостаточно, именно поэтому было принято решение перенести часть метамодели языка UML в формат онтологии OWL.

Для решения задачи интеллектуального анализа UML диаграмм проекта необходимо разработать хранилище знаний, позволяющее описать знания, извлеченные из UML диаграмм.

Разработанная онтология построена из концептов, которые являются базовыми элементами диаграммы классов. Данный подход к построению онтологии позволяет вносить изменения и расширять множество элементов онтологии в случае необходимости. В ходе построения онтологии на основе метамодели языка UML была предложена онтология [63], описываемая следующим выражением:

$$O^{prj} = \langle C^{prj}, R^{prj}, F^{prj} \rangle,$$

где $C^{prj} = \{c_1^{prj}, \dots, c_i^{prj}\}$ – множество концептов онтологии, построенных на основе элементов языка UML: "Class", "Object", "Interface", "Relationship" и т.д.;

R^{prj} – множество связей между концептами онтологии, описывающих отношения между элементами языка UML;

F^{prj} – множество функций интерпретации, определенных на множестве R^{prj} ;

Иерархия концептов разработанной онтологии представлена на рисунке 2.1.

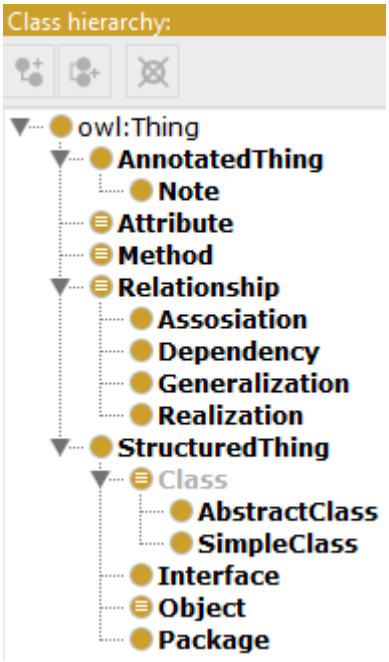

Рис. 2.1. Иерархия концептов разработанной онтологии в редакторе *Protégé*.

Иерархия DataTypeProperty и ObjectProperty разработанной онтологии представлена на рисунке 2.2.

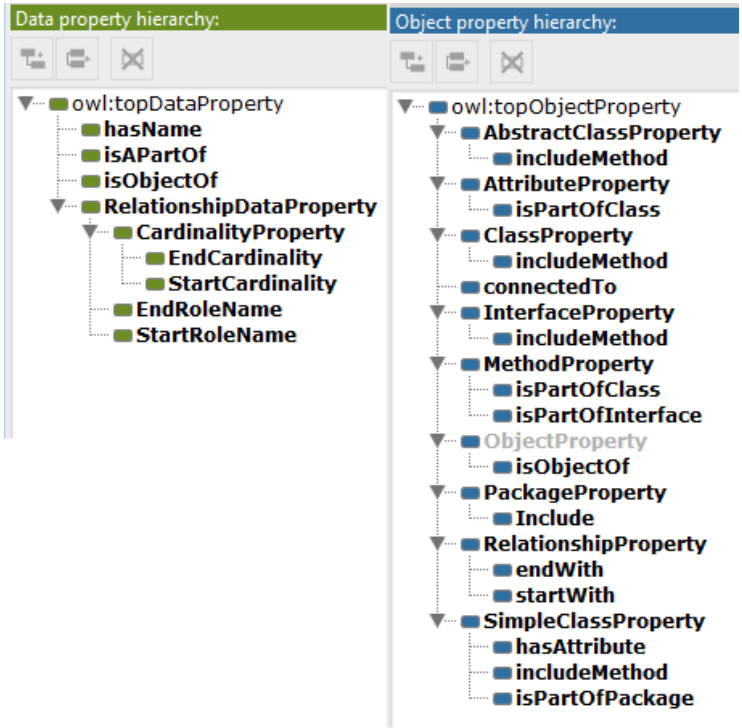

Рис. 2.2 Иерархия *DataTypeProperty* и *ObjectProperty* разработанной онтологии в редакторе *Protégé*.

2.2. Онтологическое представление шаблонов проектирования

Шаблоны проектирования заносятся в онтологию как множество экземпляров концептов разработанной онтологии. Семантика, ограничения и свойства шаблонов проектирования определяются с помощью *ObjectProperties* и *DatatypeProperties* онтологии.

Элементы шаблонов проектирования часто имеют одинаковые названия, это приводит к проблеме однозначной идентификации экземпляров концептов онтологии. Для решения данной проблемы необходимо ввести правило именования экземпляров онтологии. Название экземпляра онтологии начинается с названия шаблона проектирования и дополняется изначальным названием элемента. Для экземпляров онтологии, представляющих отношения в шаблонах проектирования, необходимо ввести аналогичное правило. Название такого экземпляра начинается с названия шаблона проектирования, далее записывается название элемента, из которого исходит отношение и завершается название экземпляра названием элемента, к которому направлено отношение.

Формально онтологическое представление шаблона проектирования можно представить следующим образом [62]:

$$O_{tmp_i}^{prj} = \{inst(C_1^{prj}), \dots inst(\mathrm{C}_{rel1}^{prj}), \dots r_{sameAs}\},$$

где $inst(C_1^{prj})$ – экземпляр концепта в онтологии, построенной на основе мета-схемы языка UML;

$inst(\mathrm{C}_{rel1}^{prj})$ – отношение между элементами шаблона проектирования, представленное в виде экземпляра концепта;

r_{sameAs} – онтологическое отношение, обозначающее эквивалентность входящих в него экземпляров.

Онтологическое представление шаблона проектирования состоит из набора экземпляров концептов и экземпляров отношений онтологии, определенных на базе метамодели языка UML.

Рассмотрим описание шаблона проектирования Builder в формате разработанной онтологии. Шаблон проектирования Builder является одним из наиболее часто используемых в промышленной разработке программного обеспечения шаблонов. Шаблон проектирования Builder представляет из себя порождающий шаблон проектирования и позволяет создавать сложные составные объекты. На рисунке 2.3 изображена UML-диаграмма классов шаблона проектирования Builder, выполненная с помощью инструмента Visual Paradigm.

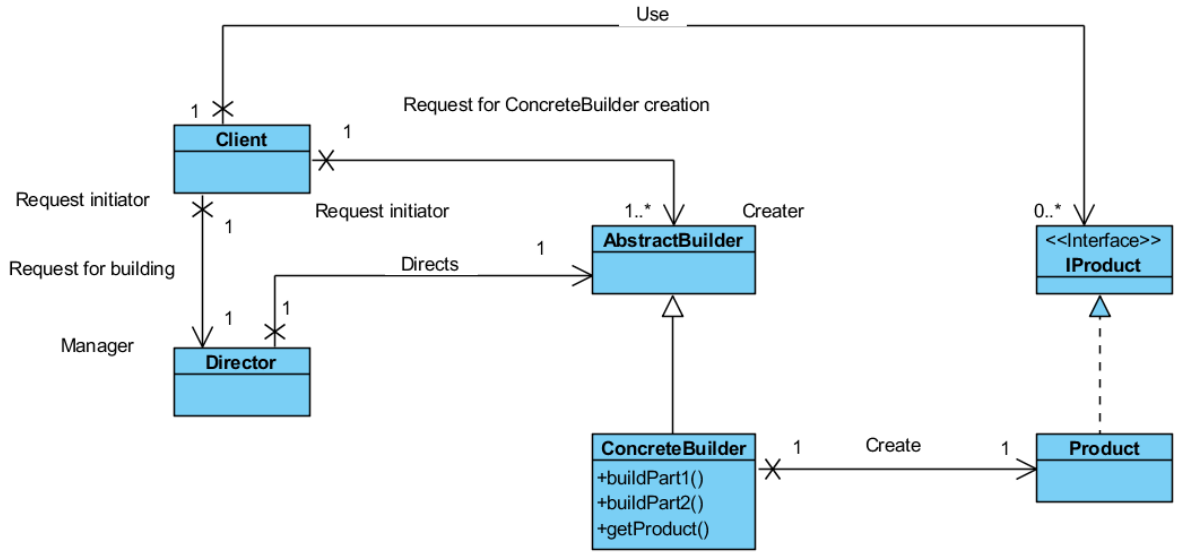

Рис. 2.3 Диаграмма классов шаблона проектирования Builder построенная в редакторе Visual Paradigm.

Для представления шаблона проектирования Builder в формате разработанной онтологии необходимо определить следующие экземпляры концептов, представленные в Таблице Builder.

Таблица 2.1

Экземпляры концептов онтологии, построенной по шаблону проектирования *Builder*

Концепт	**Экземпляры**
SimpleClass	*Builder_Client,* *Builder_Director,* *Builder_ConcreteBuilder, Builder_Product*
AbstractClass	*Builder_AbstractBuilder*
Association	*Builder_Client_AbstractBuilder, Builder_Client_Director, Builder_Client_IProduct, Builder_ConcreteBuilder_Product*
Generalization	*Builder_ConcreteBuilder_AbstractBuilder*
Realization	*Builder_Product_IProduct*

По элементам онтологии, содержащимся в Таблице Builder можно построить онтограф Builder с помощью редактора Protégé, представленный на рисунке 2.4.

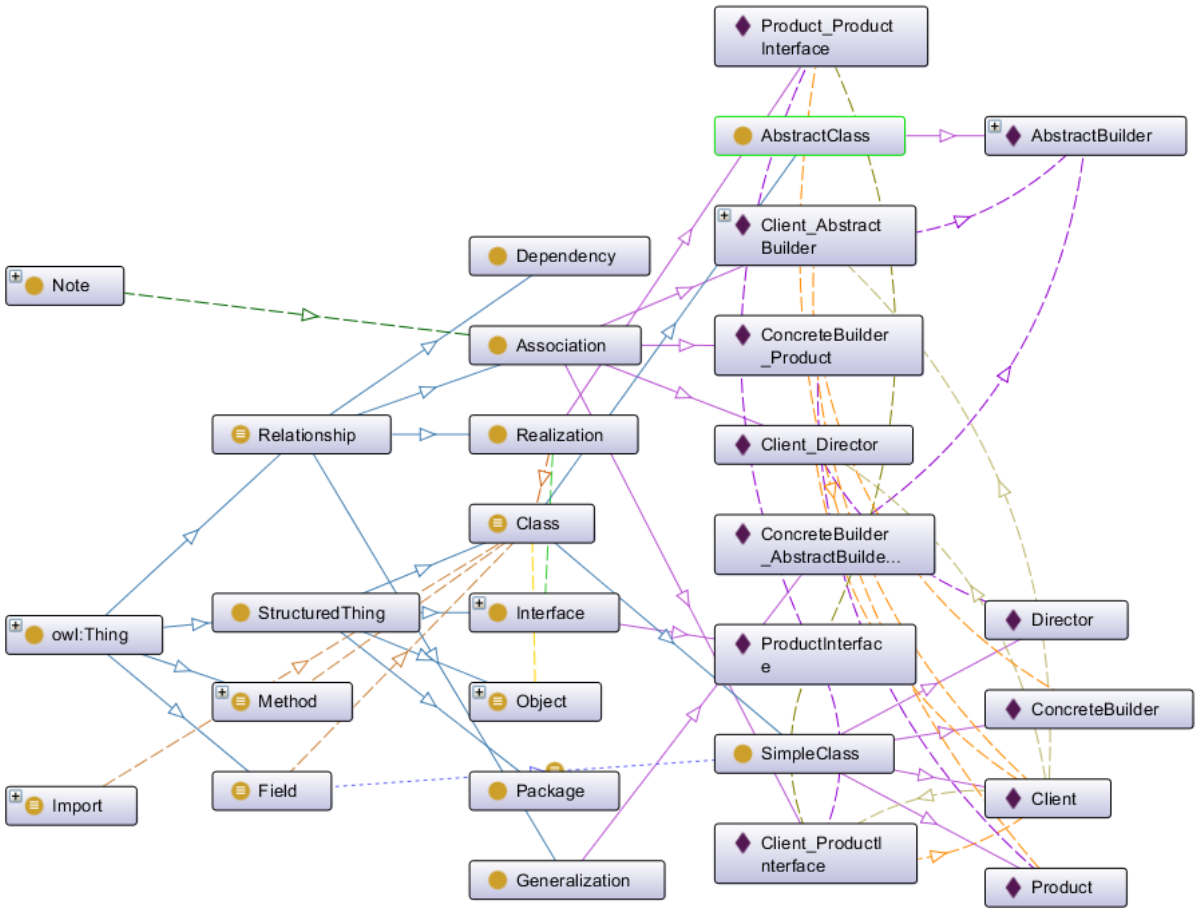

Рис. 2.4. Онтограф шаблона проектирования Builder построенный в редакторе Protege.

Так как основной целью диссертационной работы является сокращение времени на разработку программного обеспечения при разработке ПАК путем повторного использования архитектурных программных решений, необходимо построить метрику для вычисления подобия между проектируемыми и уже реализованными программными проектами.

2.3. Меры архитектурного подобия программных проектов

Для вычисления меры архитектурного подобия программных проектов предложены новая формула вычисления степени выраженности шаблона проектирования в программном проекте [17]:

$$\mu_{prj,tmp} = \frac{|C_{prj} \cap C_{tmp}| + |R_{prj} \cap R_{tmp}|}{|C_{tmp}| + R_{tmp}|},$$

где $\mu_{prj,tmp}$ – мера выраженности шаблона проектирования в программном продукте,

C_{prj} и C_{tmp} – экземпляры концепта онтологии программного проекта или шаблона проектирования,

R_{prj} и R_{tmp} – экземпляры отношения онтологии программного проекта или шаблона проектирования.

Для вычисления архитектурного подобия между программными продуктами необходимо вычислить меру выраженности для каждого шаблона проектирования из составленного множества шаблонов.

Множество шаблонов проектирования составляется ведущими разработчиками предприятия и экспертами в предметной области. Ведущие разработчики должны подбирать шаблоны проектирования, основываясь на принятом на предприятии корпоративном стиле разработки. Эксперты предметной области отбирают характерные для предметной области понятия, термины, отношения и процессы, определяющие архитектуру программного обеспечения.

2.4. Меры подобия между программными проектами

После вычисления меры выраженности каждого отобранного шаблона проектирования в каждом из рассматриваемых программных продуктов, становиться возможным вычисление меры подобия между программными проектами по одной из трех метрик [62].

Первая метрика позволяет рассчитать меру подобия с помощью выделения наиболее выраженного шаблона проектирования в каждом из проектов:

$$\mu_{dc_\gamma, dc_\delta} = \bigvee_{tmp \epsilon (dc_\gamma \cap dc_\delta)} \mu_{dc_\gamma \cap dc_\delta (tmp)},$$

где dc_γ и dc_δ – UML диаграмма классов проекта, представленная в виде выражений A-box разработанной онтологии, $\mu_{dc_\gamma \cap dc_\delta (tmp)}$ – мера подобия проектов γ и δ.

Подобная метрика показывает хорошие результаты для сравнительно небольшого количества сложных комбинированных шаблонов проектирования. Подобные шаблоны проектирования основываются на предметной области и, в меньшей мере, соответствуют шаблонам проектирования в привычном понимании промышленного программирования.

Вторая метрика учитывает меру выраженности шаблонов проектирования, превышающую определенное пороговое значение. Экспериментальным путем было подобрано пороговое значение, равное 0,3. В случае, если мера выраженности шаблона проектирования ниже 0,3, можно сделать вывод об отсутствии в программном проекте данного шаблона проектирования, и как следствие, такой шаблон проектирования необходимо исключить из рассмотрения:

$$\mu_{dc_\gamma, dc_\delta} = (\bigvee_{tmp \epsilon (dc_\gamma \cap dc_\delta) \geq 0{,}3} \mu_{dc_\gamma \cap dc_\delta}) / N,$$

где N – число шаблонов проектирования с мерой выраженности выше 0,3 для каждого из проектов.

Третья, предлагаемая в работе, метрика схожа по семантике со второй метрикой, но накладывает дополнительное условие на учет вклада меры выраженности шаблона проектирования в меру архитектурного подобия проектов:

$$\mu_{dc_\gamma, dc_\delta} = \Big(\bigvee_{tmp \epsilon (dc_\gamma \cap dc_\delta) \geq 0{,}3} \tilde{\mu}_{dc_\gamma \cap dc_\delta} \Big) / N,$$

где $\tilde{\mu}_{dc_\gamma \cap dc_\delta}$ – взвешенная мера выраженности шаблона проектирования в программном проекте [63].

Под взвешенной мерой выраженности шаблона проектирования в программном проекте будем понимать меру выраженности, нормализованную по максимальному числу элементов, входящих в состав шаблона проектирования. Данная модификация позволяет учитывать сложность внутреннего строения шаблона проектирования при вычислениях меры подобия программных проектов. По данной метрике шаблон проектирования, состоящий из 20 элементов, полностью выраженный в каждом из сравниваемых программных продуктов, будет обладать в 4 раза большим весом, чем шаблон проектирования, состоящий из 5 элементов и тоже имеющий меру выраженности $\mu_{prj_2, tmp}$, равную 1.

2.5. Методика переноса знаний из концептуальных моделей в онтологию OWL

2.5.1. Структура онтологии на основе метамодели UML

Схема работы инструментального программного средства по поиску структурно и архитектурно подобных программных проектов представлена на рисунке 2.5. На диаграмме представлены основные компоненты системы, связи между ними, форматы передаваемых между компонентами данных и пользователи с учетом исполняемой ими роли. Предполагается, что на предприятии есть следующий персонал для работы с системой:

- программисты (разработчики);
- проектировщики (наиболее опытные программисты);
- эксперты в предметной области текущего проекта.

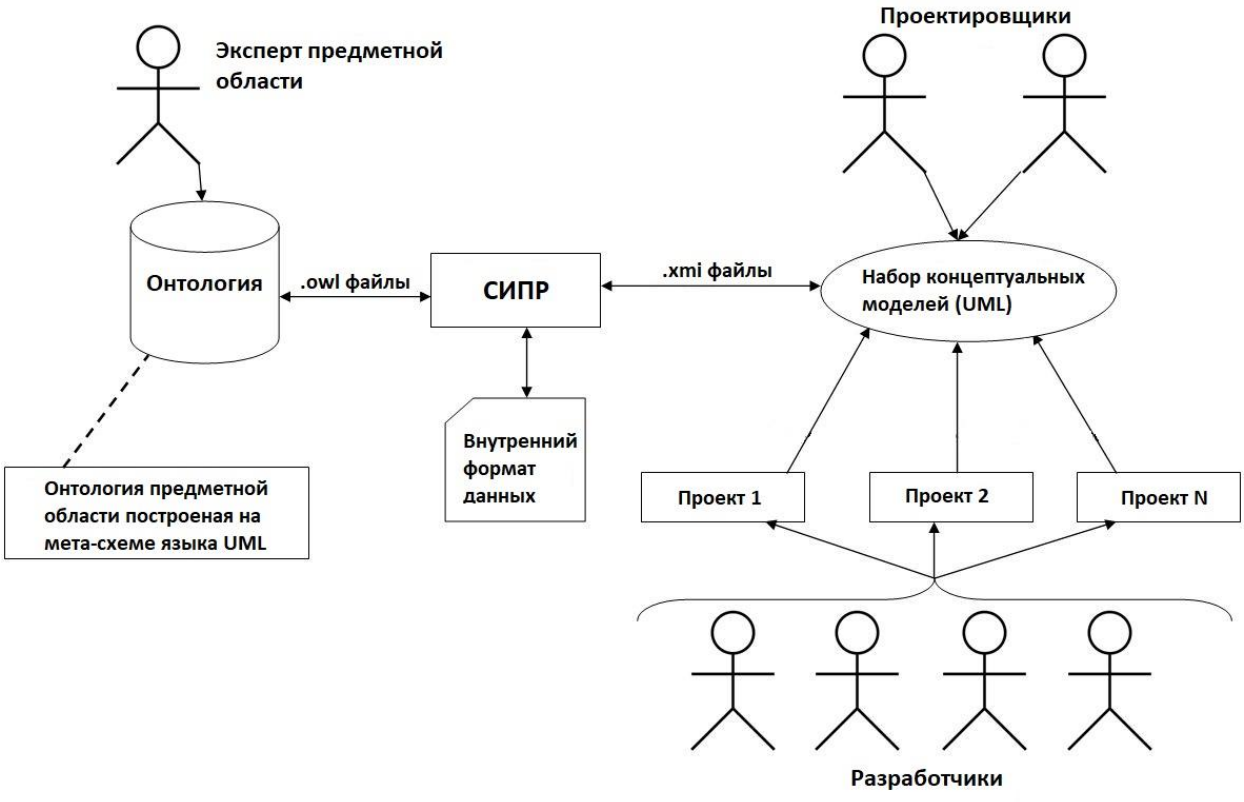

Рис. 2.5 Общая схема использования модуля извлечения знаний из проектных диаграмм.

На приведенной схеме приведен общий процесс работы разных пользователей с системой. На каждом предприятии данная схема различна, и зачастую, один и тот же специалист может выступать в нескольких ролях,

например, заносить знания в разрабатываемую онтологию и создавать проектные диаграммы.

Набор UML диаграмм, как правило, содержит информацию о процессах предметной области, используемых более чем в одном проекте. Серверное приложение, клиентские приложения для различных платформ, утилиты, сервисы и т.д. могут быть реализованы в виде отдельных приложений, но их UML диаграммы могут быть построены на общем множестве элементов. Программисты, работающие над разными приложениями одного и того же проекта, могут тратить меньше времени на синхронизацию приложений благодаря использованию проектной части, построенной на общих базовых элементах. Система позволяет связать диаграммы классов и онтологию с помощью онтологического представления. Эксперты предметной области смогут работать с онтологией напрямую через любой специализированный редактор.

2.5.2. Правила переноса знаний из концептуальных моделей в онтологию OWL

Элементы концептуальных моделей переносятся в онтологию в виде концептов с учетом семантики их интерпретации. Семантика всей диаграммы формируется из семантики отдельных элементов диаграммы и семантики их взаимодействия. Поэтому можно считать, что при подобной трансляции происходит не только формальный процесс изменения формата данных, но и перенос знаний. Это утверждение можно назвать справедливым, так как после процедуры трансляции можно использовать машину логического вывода, например, pellet или fact++, которые доступны непосредственно в наиболее популярном редакторе онтологий Protégé.

2.5.3. Алгоритм трансформации UML – диаграммы классов в онтологию формата OWL.

Рассмотрим правила трансляции для наиболее распространенных элементов UML диаграмм в онтологию. Входными данными для алгоритма системы интеграции [20] является UML диаграмма классов в форме XMI

документа. XMI документ представляет из себя XML файл определенной внутренней структуры. Подобная структура позволяет производить нетривиальные запросы к документу. Таким образом можно использовать любую, наиболее удобную библиотеку для работы с XML.

На рисунке 2.6 представлена блок-схема алгоритма переноса знаний из UML-диаграммы в OWL онтологию. На данной блок-схеме представлена последовательность вызовов отдельных функций, каждая из которых реализует техническую задачу. Особый значение имеет последовательность вызовов функций, представленная на рисунке 2.6, выполнение функций в другом порядке приведет к нарушению целостности данных.

Входными данными для алгоритма являются UML диаграммы, экспортированные из редактора в формате XMI. Помимо диаграмм алгоритм использует шаблон результирующей онтологии, содержащей только T-Box основанный на мета-схеме языка UML. Использование шаблона вместо генерации T-Box имеет преимущество в гибкости работы с разработанным ПО. Шаблон может быть расширен вне исходного кода проекта с помощью редактора онтологий.

Результатами выполнения алгоритма являются:

- заполненное онтологическое представление проекта;
- список ошибок, в случае сбоя преобразования;
- дополненное онтологическое представление, полученное с помощью МЛВ;
- список ошибочных утверждений, нарушающих непротиворечивость онтологии.

К ограничениям данного алгоритма стоит отнести:

- формат экспорта UML диаграмм XMI версии 2.1;
- формат онтологии OWL 2.0.

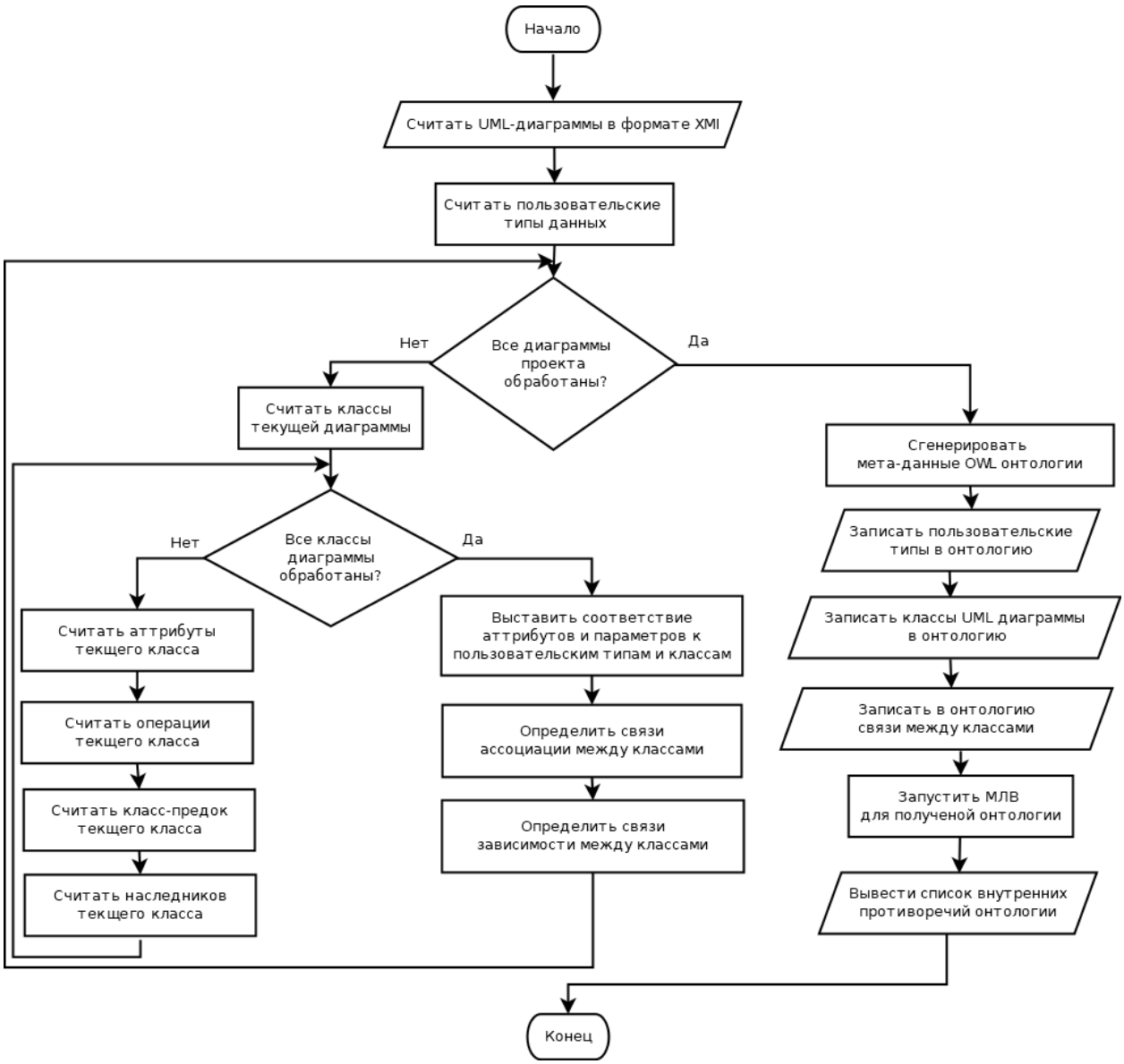

Рисунок 2.6 Блок-схема алгоритма переноса знаний из набора UML-диаграмм в онтологию формата OWL.

Ниже представлен с помощью псевдокода разработанный алгоритм переноса знаний из набора UML диаграммы в онтологию формата OWL.

1. *begin //Parsing UML in XMI format*
2. *classDiagram = getClassDiagramm(XMIroot);*
3. *if (classDiagramm == null) than*
4. *exit;*
5. *UserTypes [] = getTypes(classDiagram);*
6. *Classes [] = getClasses(ClassDiagram)*
7. *foreach(class in Classes[])*
8. *begin*

```
9.          class.Attributes = getAttributes(class);
10.         class.Operations = getOperations(class);
11.         class.Childs = getChilds(class, Classes[]);
12.         class.Parent = getParent(class, Classes[]);
13.   end
14.   MatchingDataTypes(Classes [], UserTypes[]);
// Replacing XMI ID on the type of names
15.   Associations [] = getAssociations(root, Classes []);
16.   Dependencies [] = getDependencies(root, Classes []);
17. //Generation of OWL
18.   owl = writeHeader(owl);
//Generation ontology header
19.   owl = writeDataTypes(owl, UserTypes);
//DataTypeProperties
20.   owl = writeClasses(owl, Classes[]);
//Classes
21.   owl = writeLinks(owl, Classes[],Associations [],
Dependencies [])//ObjectProperties;
22.   ErrorList [] = reasoner.Validate(owl);
23.   foreach(error in errorList)
24.   begin
25.         print(error);
26.   end
27. end
```

2.6. Формирование прогноза развития проекта на основе результатов прогнозирования с помощью комбинаций и коллективов моделей

Одним из наиболее успешных направлений в прогнозировании временных рядов является объединение моделей прогнозирования в

комбинации или коллективы. Как показывает практика, объединение моделей в комбинацию или коллектив позволяет получить лучший результат по сравнению с результатами каждой из моделей в отдельности.

В данной диссертационной работе были предложены и реализованы комбинация моделей с использованием весов, полученных с помощью информационных критериев [71], комбинация с использованием нечетких весов [70] и коллектив, полученный агрегирующим алгоритмом Вовка [97].

2.6.1. Формальное представление временного ряда управления проектом

Руководителю проекта необходимо выполнить прогнозирование возможных вариантов развития проекта и своевременно принять проектные и организационные решения, повышающие эффективность разработки средства автоматизации. Одним из наиболее выразительных способов представления знаний о ходе разработки являются временные ряды основных показателей проекта. В процессе автоматизированного проектирования руководитель проекта может эффективно отследить состояние и динамику реализации проекта на основе набора временных рядов, описывающих проект:

$P =< TS_{releases}, TS_{commits}, TS_{tasks}, TS_{concepts}, TS_{states}, F_{states} >$, где:

$TS_{releases}$ – временной ряд версии программного обеспечения, поставляемого клиентам (release);

$TS_{commits}$ – временной ряд изменений исходного кода (commits);

TS_{tasks} – временной ряд задач ($tasks$);

$TS_{concepts}$ – временной ряд измененных, добавленных и удаленных концептов онтологии предметной области, извлеченных из диаграммы классов проекта за последний этап проектирования (для классической разработки с моделью ЖЦ) или извлеченных из исходного кода программы за последний этап или спринт разработки;

TS_{states} – временной ряд состояний программного проекта, основанный на интеграции показателей временных рядов, характеризующий проект.

F_{states} – прогноз состояний программного продукта, полученный с помощью выбранного коллектива или комбинации методов.

$$TS_{releases} =< P_1^{rel}, \dots, P_i^{rel}, \dots, P_N^{rel} >, \text{где:}$$

N – количество точек временного ряда $TS_{releases}$,

P_1^{rel} – точка временного ряда $TS_{releases}$;

$$P_i^{rel} =< Label, Date_{plan}, Date_{fact} >, \text{где:}$$

$Label$ – метка или название сборки, позволяющее ее идентифицировать,

$Date_{plan}$- дата сборки по плану,

$Date_{fact}$- дата поставки сборки по факту.

$$TS_{commits} =< P_1^{com}, \dots, P_i^{com}, \dots, P_N^{com} >, \text{где:}$$

N – количество точек временного ряда $TS_{commits}$,

P_1^{com} – точка временного ряда $TS_{commits}$;

$$P_i^{com} =< Label, Date, Operation, Type, Author >, \text{где:}$$

$Label$ – уникальный номер и комментарий к изменению исходного кода, позволяющее ее идентифицировать,

Operation –изменений: {*Addition, Edit, Remove, All*}

Type – тип вносимых изменений: {*Gui, Business Logic, Subject area*}

$Date$ – дата внесения изменения в исходный код.

Autor – автор изменений.

$$TS_{tasks} =< P_1^{task}, \dots, P_i^{task}, \dots, P_N^{tasks} >, \text{где:}$$

N – количество точек временного ряда TS_{tasks},

P_1^{task} – точка временного ряда TS_{tasks};

$$P_i^{task} =< Label, Date_{plan}, Date_{fact} >, \text{где:}$$

$Label$ – уникальный номер и название задачи в программе, позволяющие ее идентифицировать,

$Date_{plan}$- дата реализации задачи по плану,

$Date_{fact}$- дата реализации задачу по факту.

$$TS_{concepts} =< P_1^{concept}, \dots, P_i^{concept}, \dots, P_N^{concept} >, \text{где:}$$

N – количество точек временного ряда $TS_{concepts}$,

$P_1^{concept}$ – точка временного ряда $TS_{concepts}$;

$$P_i^{concept} =< Label, Date >, \text{ где:}$$

$Label$ – уникальный номер версии проектного решения,

$Date$ – дата создания новой версии проектного решения,

$$TS_{states} =< P_1^{state}, \dots, P_i^{state}, \dots, P_N^{sate} >, \text{ где:}$$

N – количество точек временного ряда TS_{states},

P_1^{state} – точка временного ряда TS_{states};

$$P_i^{state} =< Date, State >, \text{ где:}$$

$Date$ – дата присвоения проектному решению данного состояния,

$State$ – одно из множества состояний проекта {*Start, Unexpected fast, fast, Normal, Stable, Chaotic, Danger, Fail*}

$$F_{states} =< P_1^{state}, \dots, P_i^{state}, \dots, P_N^{sate} >, \text{ где:}$$

N – количество точек временного ряда F_{states},

P_1^{state} – точка временного ряда F_{states};

$$P_i^{state} =< Date, State >, \text{ где:}$$

$Date$ – дата присвоения проектному решению данного состояния,

$State$ – одно из множества состояний проекта {*Start, Unexpected fast, fast, Normal, Stable, Chaotic, Danger, Fail*}

В пункте 2.6. описаны разработанные методы: комбинация моделей с использованием весов, полученных с помощью информационных критериев, комбинация с использованием нечетких весов и коллектив, полученный агрегирующим алгоритмом. В ходе исследования был реализован массив методов для прогнозирования временных рядов, состоящий из классических методов экспоненциального сглаживания и нечетких методов прогнозирования временных рядов. Часть нечетких методов были предложены в ходе работы по хозяйственному договору с ООО «Эверест ресерч» и основаны на интеграции нечеткой модели и традиционного экспоненциального сглаживания. Отличительной особенностью коллективов и комбинаций моделей прогнозирования, предложенных и реализованных в

рамках данного диссертационного исследования, является интеграция результатов работы четких и нечетких моделей.

Временные ряды выделенных показателей невозможно интерпретировать как состояние программного продукта, и как следствие, невозможно дать рекомендации исходя из их прогноза. Поэтому было решено выделять семантически значимые комбинации и последовательности показателей, которые получают семантическую метку, описывающую изменение состояния программного обеспечения.

Таблица 2.2

Соответствия последовательностей и комбинаций показателей, изменениям, вносимым в проект

Последовательность или комбинация показателей	**Изменение состояния**
ЕСЛИ P_{i+1} не существует	ТО P_i^{state} :{State - Start}
ЕСЛИ P_i^{rel}:$\{Date_{plan} > Date_{fact}\}$ && P_i^{state}:{State –Start, Unexpected fast, Fast}	ТО P_i^{state} :{State - Normal}
ЕСЛИ P_i^{rel}:$\{Date_{plan} > Date_{fact}\}$	ТО P_i^{state} :{State - Danger}
ЕСЛИ >70% P_i^{task} :$\{Date_{fact} > Date_{plan}\}$	ТО P_i^{state} :{State - Unexpected fast}
ЕСЛИ >30% P_i^{task} :$\{Date_{fact} > Date_{plan}\}$	ТО P_i^{state} :{State - Fast}
ЕСЛИ >30% P_i^{commit} :$\{Operation \ - \ \text{All}\}$	ТО P_i^{state} :{State - Chaotic}
ЕСЛИ >50% P_i^{commit} :$\{Operation \ - \ \text{All}\}$ && P_i^{state}:{State != Start, Unexpected fast, Fast} && P_i^{rel}:$\{Date_{plan} > Date_{fact}\}$	ТО P_i^{state} :{State - Fail}

2.6.2. Комбинация моделей на основе ИК

Основной идеей данного подхода к комбинации методов является вычисление весов для моделей с использованием информационного критерия. В проекте реализован способ расчета весов, рассмотренный в [71]. Для расчета весов вычисляется разница между наименьшим значением критерия и текущим, например, критерий Акаике:

$$\Delta_{AIC}(M) = \ AIC(M) - MIN_{N \in \mu}\big(AIC(N)\big).$$

Далее рассчитывается функция правдоподобия:

$$Likehood = \exp\left(-\frac{1}{2}\Delta_{AIC}(M)\right).$$

После этого рассчитаем веса, нормализовав функцию правдоподобия:

$$\omega_{AIC}(M) = \frac{\exp\left(-\frac{1}{2}\Delta_{AIC}(M)\right)}{\sum_{N\in\mu}\exp\left(-\frac{1}{2}\Delta_{AIC}(N)\right)}.$$

Далее при расчете прогноза будем использовать значения модели с учетом веса модели. Подобным образом можно рассчитывать веса, используя и другой критерий.

2.6.3. Комбинация моделей с нечеткими весами на основе ИК

Другой подход к построению комбинаций моделей предложен в [70]. Нечеткий вес модели рассчитывается как:

$$l_j(i) = \beta_i r\left(e_j(i)\right) + (1 - \beta_i)s\left(c_j(i)\right),$$

где β_i - адаптивный коэффициент, обычно 0<β_i< 1; r и s - серые веса, e - относительная ошибка, c - серая тенденция;

$$\beta_i = 1 - \frac{i-1^N}{i},$$

где 0 < i < K,K-длина временного ряда, N - константа, обычно 0,5.

$$r\left(e_j(i)\right) = \frac{\alpha + \beta\delta}{|e_j(i)| + \beta\delta};$$

$$s\left(\acute{c}_j(i)\right) = \frac{\alpha + \beta\delta}{|\acute{c}_j(i)| + \beta\delta};$$

где $\alpha = \min\min|y(i) - fj(i)|, fj(i) = g[y(i-k), y(i-k+1), \dots, y(i-1)]$, $\beta = max_j max_i|y(i) - fj(i)|$, δ обычно выбирается 0.5

$$e_j(i) = \frac{y(i) - f_j(i)}{y(i)};$$

$$c_j(i) = \frac{y(i) - \frac{1}{k}\sum_{m=i-k+1}^{i} y(m)}{y(i)},$$

где j =1,2..., n; i = 1,2,...k , y(i) - значение ряда, f(i) - прогнозное значение.

2.6.4 Коллектив моделей на основе Агрегирующего алгоритма Вовка

Агрегирующий алгоритм Вовка рассматривает процесс прогнозирования в виде тройки <Ω, Г, λ>, где Ω — множество исходов, Г — множество допустимых предсказаний, λ: Ω×Г→R+— функция потерь.

Алгоритм прогнозирования представляется в виде функции A: Ω∗ →Г, которая выдает прогноз γ(T+1) для следующего элемента последовательности. При этом потери при прогнозировании определяются как

$$\text{LossA}(\omega 1, \ldots, \omega T)=\sum_{t=1}^{T} \lambda(\omega t, \gamma t).$$

Агрегирующий алгоритм оперирует с несколькими моделями, взвешивая в экспоненциальном пространстве ошибки прогнозов экспертов, а затем порождает прогноз с помощью решающего правила:

$$\gamma_{T+1} = \sum \log\left(\sum_{j=1}^{N} \exp\left(-\mu\, Loss_{A_j}(\overleftrightarrow{\omega})\right) 4.\, p_0^{(j)}\right);$$

где $p_0 -$ вес эксперта j в начальный момент времени, μ>0-параметр алгоритма.

2.8. Подход к агрегации методов прогнозирования временных рядов

В ходе исследования был предложен и реализован массив методов для прогнозирования временных рядов, состоящий из следующих методов:

1. Классических методов экспоненциального сглаживания;
2. Нечетких методов, основанных на интеграции нечеткой модели и традиционного экспоненциального сглаживания, в том числе [56], [84], [102];
3. Подхода, описанного в статье [96], в котором строится модель временного ряда в виде нечетких множеств, над которыми совершаются операции, аналогичные классическим экспоненциальным методам сглаживания;

4. Нечеткий метод *Direct Set Assignment* [69] для прогнозирования временных рядов, объединенный с экспоненциальным сглаживанием.

Новизна части исследования, посвященной прогнозированию временных рядов, заключается в нечеткой модификации классических методов экспоненциального сглаживания с помощью второго и третьего подходов и создании коллективов и комбинаций методов [59], [60], [1], [2], [3], [4]. Отличительной особенностью коллективов и комбинаций является интеграция результатов работы четких и нечетких методов экспоненциального сглаживания.

Глава 3 Разработка программного комплекса автоматизации проектирования и разработки программных и программно-аппаратных комплексов

3.1 Структурно-функциональное решение программного комплекса автоматизации проектирования и разработки

В рамках решения задачи оценки эффективности предложенных моделей, методик и алгоритмов поддержки интеллектуального проектирования и разработки программных и программно-аппаратных комплексов был разработан модульный программный комплекс, предназначенный для автоматизации работы с проектными документами, исходным кодом, данными из систем контроля версий и отслеживания ошибок. В качестве унифицированного формата хранения знаний (УФХЗ) в данном программном комплексе используется онтология в формате OWL, T-box которой, построен на основе мета-схемы нотации языка UML

Основные функции программного комплекса:

1. Индексирование проектных документов – формирование индекса в виде документа УФХЗ;
2. Индексирование исходного кода программных и программно-аппаратных комплексов – формирование индекса в виде документа УФХЗ;
3. Хранение и обновление индексов ранее спроектированных и разработанных проектов в виде документа УФХЗ;
4. Хранение и обновление знаний предметной области в виде шаблонов проектирования, представленных документами УФХЗ;
5. Вычисление меры структурно-семантического подобия проектов на основе предметной области;
6. Проверка онтологии проекта в формате УФХЗ на согласованность с помощью машины логического вывода.

Индексирование проекта можно произвести на этапе проектирования, что позволяет рассмотреть схожие проекты до начала этапа разработки.

Комплекс состоит из двух подсистем:

1. Подсистема поддержки интеллектуального проектирования (ПИП) программных продуктов на основе знаний, хранящихся в онтологии УФХЗ;

2. Подсистема анализа данных (ПАД). Данная подсистема позволяет на основании знаний, полученных из прикладных систем разработки программного обеспечения, генерировать рекомендации по дальнейшему развитию проекта. В основе ПАД лежит подход к представлению данных в виде временных рядов с последующей обработкой.

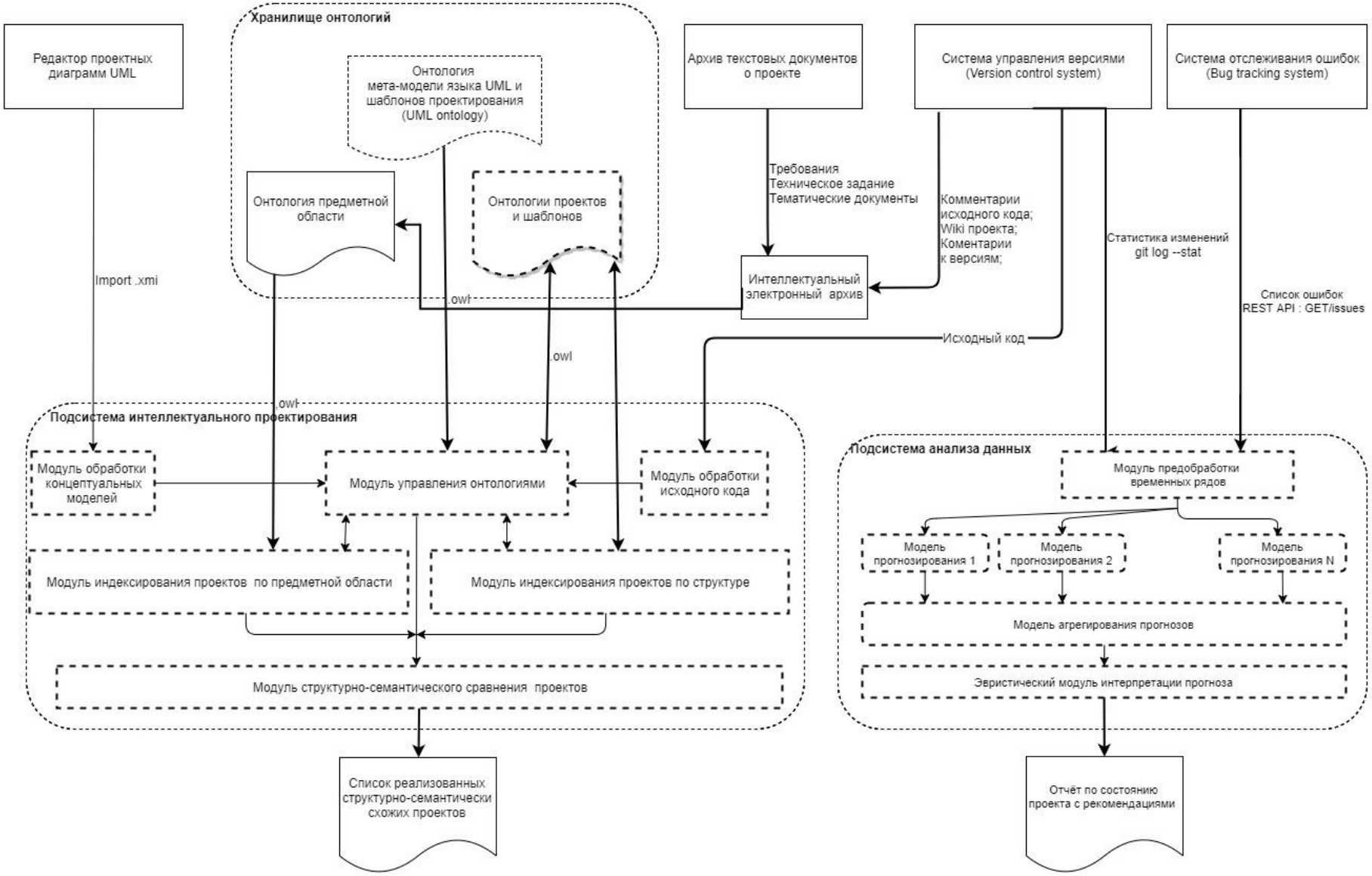

Рис 3.1. Общая схема программной системы.

3.2 Описание подсистемы интеллектуального проектирования и разработки

Процесс проектирования программного продукта представляет собой нетривиальную творческую задачу. Проектировщик должен сформировать архитектуру программного проекта из ряда технических решений, принимаемых по каждому модулю. Процесс проектирования предполагает прогнозирование, но с высокой вероятностью некоторые технические решения будут со временем модифицированы. Чем больше архитектурных решений приняты правильно, тем меньше будет происходить повторной работы во время реализации программного продукта, что приведет к уменьшению времени на реализацию проекта. Оптимизация использования времени специалистов позволит ускорить время реализации проекта и, тем самым, снизить себестоимость разработки для предприятия.

Одним из результатов проектирования программного продукта являются проектные диаграммы. При внесении изменений в архитектуру программной системы наиболее верным решением является корректировка технического задания и дальнейшая доработка приложения. Изменение или дополнение технического задания весьма трудоемкий процесс, который влечет корректировку всех документов, определяющих разработку проекта. Временные рамки проекта зачастую не позволяют вносить изменения в проектные диаграммы, вместо этого изменения вносятся на уровне исходного кода.

В связи с различным представлением проекта на разных этапах и цикличностью разработки необходимо получить универсальное представление проекта. Для формализации знаний о предметной области и примененных архитектурных решениях в программном продукте предлагается использовать онтологию, разработанную на языке OWL.

3.2.1 Требования к подсистеме поддержки интеллектуального проектирования

Подсистема поддержки интеллектуального проектирования должна обеспечить:

- извлечение знаний из диаграмм классов на языке UML;
- извлечение знаний из исходного кода проекта;
- формирование онтологий специального формата;
- сохранение извлеченных знаний в форме специализированной онтологии;
- формат хранения знаний должен быть совместимым на уровне проектных решений.

Язык UML накладывает на подсистему ограничения по формату используемой онтологии. Специализированный формат онтологии определяется метамоделью языка UML, так как в этом случае в онтологию можно будет записать любое знание, описанное с помощью языка UML. Разработанная онтология поддерживает большинство наиболее часто используемых элементов UML. Элементы языка UML, не отраженные в онтологии изначально, могут быть добавлены позднее, так как в качестве T-Box онтологии используются утверждения, полученные из метамодели UML. Подобный подход позволяет встраивать поддержку новых элементов UML, которые будут добавлены в новых спецификациях языка UML.

3.2.2. Диаграмма последовательностей подсистемы поддержки интеллектуального проектирования

Ключевым моментом в эксплуатации разработанной системы является возможность цикличного проектирования. Предполагается, что разработчики перед внесением любого концептуального изменения в проект на уровне исходного кода обновляют проектную часть – UML-диаграммы. Возможно, внесение изменений в проектную часть произойдет после изменения исходного кода. Подобный подход к разработке не может считаться верным, так как вносить изменения в проект без анализа изменения программного продукта в

целом не профессионально и допустимо лишь в сравнительно небольших проектах с малым временем разработки и сопровождения.

Необходимо стремиться к уменьшению трудозатрат на поддержание проектной части программного продукта в актуальном состоянии. Образцом могут служить системы контроля версий, вошедшие в практику разработки программного обеспечения.

Разработанная программа трансляции позволяет синхронизировать изменения между онтологией, проектными диаграммами и исходным кодом проекта. Ее задача состоит не только в поддержании истории изменений всех фрагментов проекта, но и в сохранении связей между элементами. Примерами полезных связей могут служить следующие: класс А в онтологии связан с классом А на диаграмме классов и диаграмме активностей и классом А в исходном коде программного продукта.

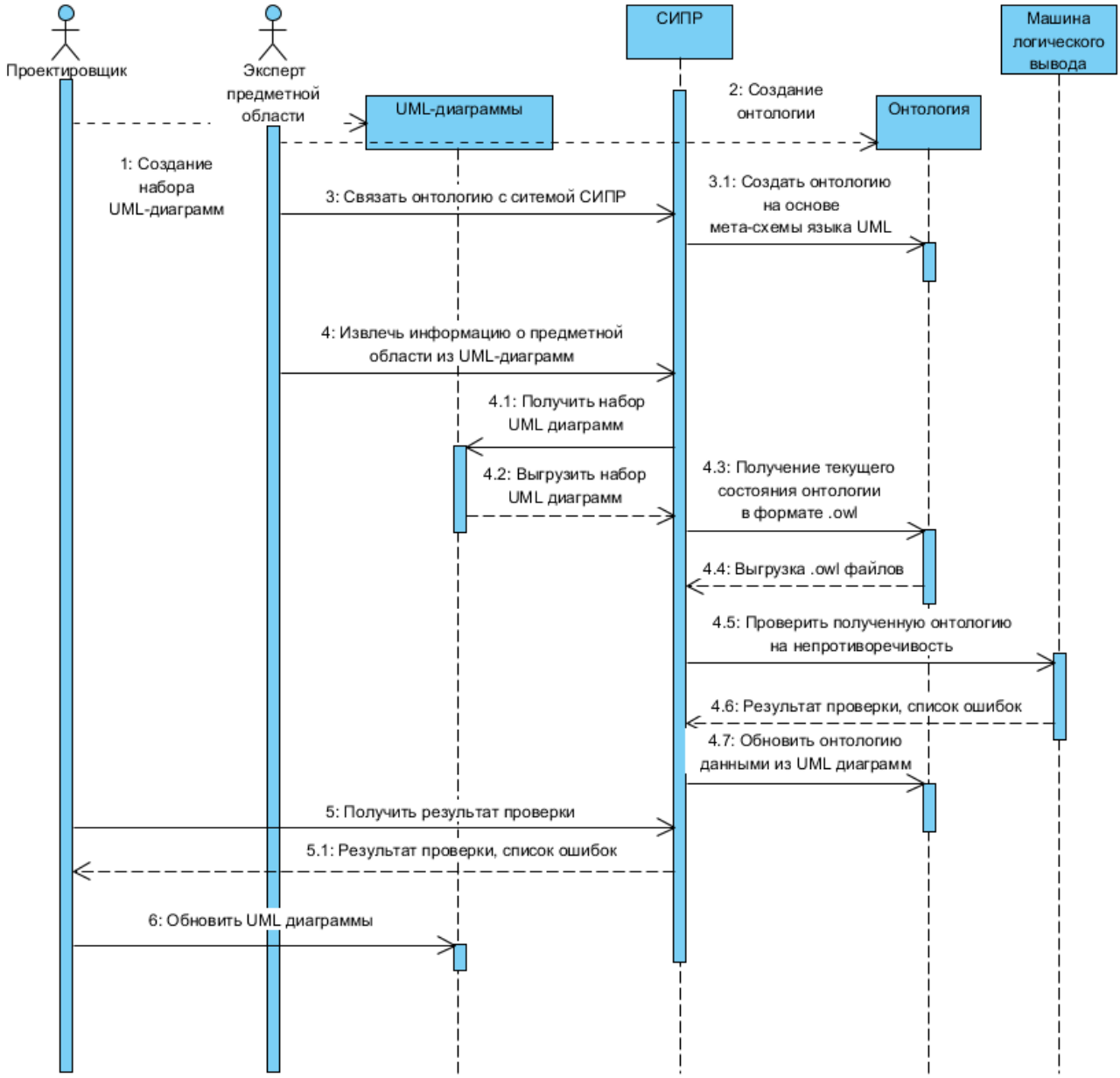

Рис. 3.2 Диаграмма последовательностей модуля извлечения знаний из проектных диаграмм.

На рисунке 3.2 представлена диаграмма последовательностей, определяющая процессы в работе системы интеграции на этапе проектирования жизненного цикла программного проекта:

1. Проектировщик создает UML диаграммы;
2. Эксперт предметной области создает онтологию;
3. Эксперт предметной области формулирует запрос на интеграцию онтологии UML диаграммами;
4. Система интеграции создает онтологию, основанную на метамодели UML:

4.1. Система интеграции выгружает данные из онтологии в формате OWL;

4.2. Система интеграции выгружает данные из UML диаграмм в формате XMI;

4.3. Система интеграции совмещает данные, полученные из UML диаграммы и онтологии, используя правила интеграции;

4.4. Система интеграции возвращает список ошибок, полученных при попытке согласовать онтологию и UML-диаграмму с помощью аксиом предметной области;

4.5. Система интеграции обновляет онтологию данными, полученными из UML диаграмм.

5. Проектировщик получает обновленные диаграммы с учетом результатов работы логического вывода;

6. Проектировщик обновляет UML -диаграммы.

3.2.3 Архитектура модуля извлечения знаний из UML диаграмм

Среди UML-диаграмм наиболее распространена диаграмма классов. Диаграмма классов может быть построена для каждого объектно-ориентированного проекта. Даже, если UML-диаграмма классов не была построена при проектировании, можно автоматически создать ее с помощью обратного проектирования.

За счет распространения технологии и культуры программирования проекты, написанные с учетом универсальных требований к стилю программирования, используют длинные мнемонические имена и общепринятые шаблоны проектирования. Данная унификация позволяет использовать UML-диаграмму классов автоматизировано.

UML-диаграмма классов модуля, отвечающего за трансляцию UML-диаграммы классов в онтологию, приведена на рисунке 3.3.

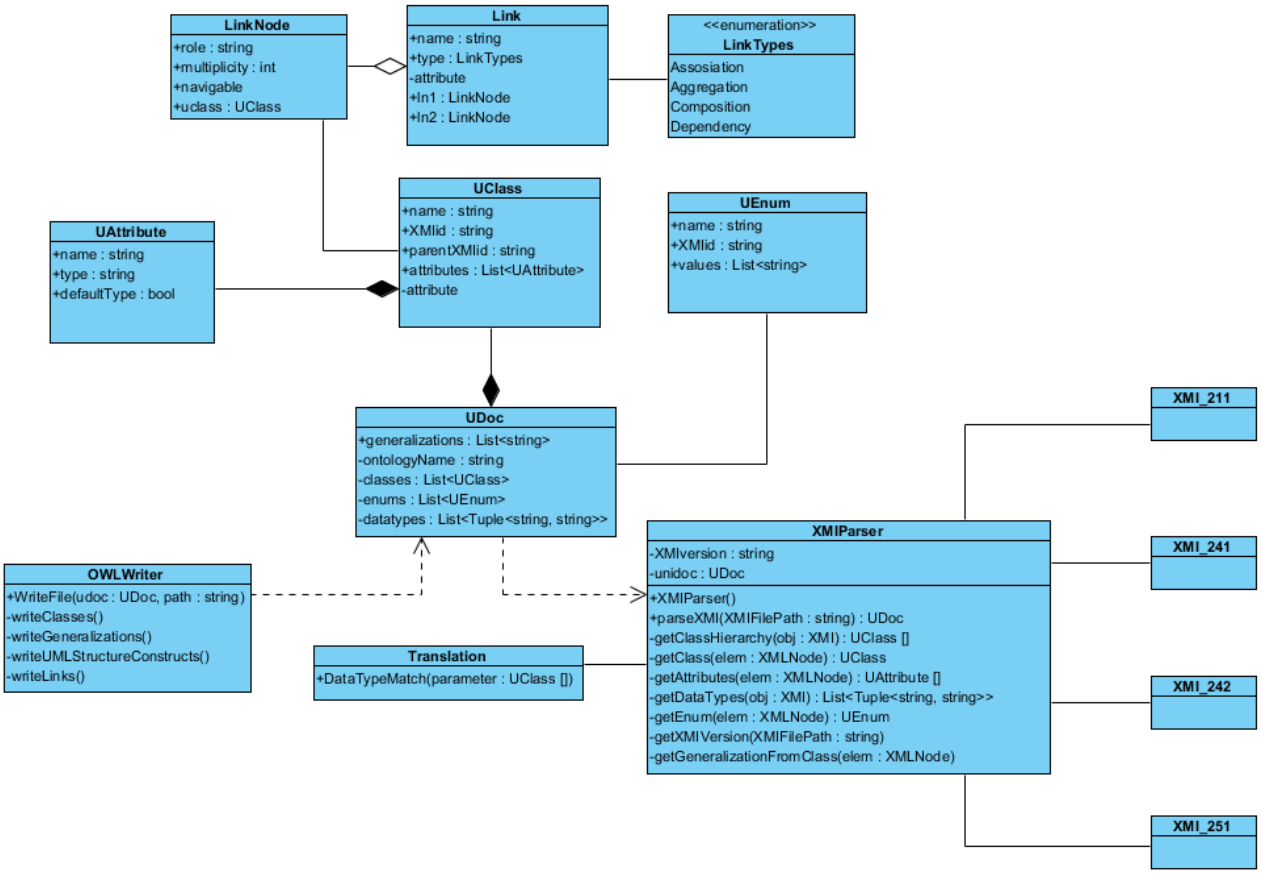

Рис.3.3 Диаграмма классов модуля трансляции UML-диаграмм в онтологию.

Модуль разработан для использования XMI формата, предложенного консорциумом OMG и являющимся единственным общепринятым форматом хранения UML-диаграмм. Не все редакторы UML-диаграмм поддерживают экспорт UML-диаграммы в формате XMI.

На данный момент модуль поддерживает версии XMI от 2.1 и выше. В таблице 3.1. приведены результаты тестирования по трансляции диаграмм классов, разработанных в разных редакторах.

Таблица.3.1

Результаты тестирования экспорта диаграмм в XMI формат

Редактор	Версия XMI	Результат транслирования	Комментарий
ArgoUML	1.2	Не транслировано	Слишком старая версия XMI
Astah	1.1	Не транслировано	Слишком старая версия XMI
Altova UModel	2.1, 2.4.1	Транслировано	Редактор может быть использован для экспорта диаграмм в XMI формат
Enterprise Architect	1.1, 2.1	Транслировано	Редактор может быть использован для экспорта диаграмм в XMI формат
MagicDraw	Есть	Не транслировано	Экспорт в XMI не доступен в бесплатной версии
Innovator Enterprise	Нет	Не транслировано	Экспорт в XMI не доступен в бесплатной версии
Modelio	Нет	Не транслировано	Экспорт в XMI не доступен в бесплатной версии
StarUML	Нет	Не транслировано	Экспорт в XMI не доступен в бесплатной версии
Visual Paradigm	2.1	Транслировано	Редактор используется как основное средство при разработке трансляции UML-диаграмм

По данным из таблицы 3.1 XMI, далеко не все редакторы поддерживают XMI экспорт. Тем не менее, XMI является единственным унифицированным форматом, поэтому модуль основан на данном формате.

Понятие элемента является фундаментальным в мета-схеме языка UML. Все UML-диаграммы состоят из элементов. Построенная онтология языка UML содержит описание следующих типов элементов: класс, абстрактный класс, интерфейс, объект, пакет, метод и атрибут. Такой набор элементов позволяет описать и сохранить в онтологии большинство наиболее известных шаблонов проектирования.

В мета-схеме языка UML отношение является элементом и описывает какой-либо вид взаимодействия между другими элементами. Следующие отношения языка UML были отобраны и перенесены в онтологию, как наиболее распространенные: обобщения, ассоциации, реализации и зависимости.

Для описания элементов диаграммы классов, таких как: класс, атрибут и метод, необходимо установить соответствие между типами данных, доступными в UML и в OWL. Типы данных в UML и OWL совпадают не полностью, но имеют общую часть. Предлагаемый вариант трансляции на уровне типов данных представлен в Таблице 3.2.

Таблица 3.2.

Сравнение типов данных

Тип данных	OWL	UML
Действительные, десятичные и целый числа	*xsd:decimal,xsd:integer, xsd:nonNegativeInteger, xsd:nonPositiveInteger, xsd:positiveInteger, xsd:long, xsd:int, xsd:short, xsd:byte, xsd:unsignedLong, xsd:unsignedInt, xsd:unsignedByte*	
	owl:real, owl:rational	
Числа с плавающей точкой	*xsd:double, xsd:float*	
Строки	*xsd:string, xsd:normalizedString, xsd:token, xsd:language, xsd:Name, xsd:NCName,*	
		xsd:NMTOKENS,xsd:ID,xsd:IDREF,xsd:IDREFS, xsd:Entity,xsd:Entities
Логические значения	*xsd:boolean*	
Двоичные данные	*xsd:hexBinary, xsd:base64Binary*	
Интернациональный идентификатор ресурса	*xsd:anyURI*	
Моменты времени	*xsd:dateTime*	
	xsd:dateTimeStamp	*xsd:date,xsd:time xsd:gYearMounth, xsd:gYear, xsd:gMounth, xsd:gDay*
Типы XML Shema не указанные в OWL 2 спецификации		*xsd:anySimpleType, xsd:anyType, xsd:duration, xsd:NOTATION, xsd:QName*

Базовые типы данных в OWL и UML основаны на XSD, это означает, что они могут быть приведены друг к другу напрямую, без дополнительных преобразований. Уникальные типы данных в OWL и UML были получены путем наложения ограничений на базовые типы данных. Это означает, что базовые типы данных также полностью совместимы.

Понятие класса существует как в UML, так и в OWL. Класс в OWL является множеством экземпляров. В качестве встроенных классов предлагаются множество всех экземпляров класса *Thing* и пустое множество экземпляров класса *Nothing*. Пользовательские классы связаны отношениями с помощью объектных свойств и литеральными значениями с помощью свойств типов данных.

UML диаграмма классов рассматривает класс как совокупность внутреннего устройства и поведения объектов. Правила нотации UML предоставляет аналитику широкие возможности для выражения диаграммы классов, которые тем или иным способом реализуются в конкретном инструменте. В UML класс определяется как структурная единица диаграммы. Для класса в UML-диаграмме правомерно использовать определение из ООП. Классы предметной области связаны отношениями с помощью объектных свойств и литеральными значениями с помощью свойств типов данных.

Концепты класса отличаются семантически, но возможна трансляция на уровне иерархии классов. Внутреннее устройство классов UML-диаграммы можно транслировать в OWL классы, используя объектные свойства и свойства типов данных.

В процессе проведенного исследования необходимо было решить задачу преобразования иерархии классов UML в иерархию классов OWL. Из определения наследования как в UML, так и в OWL следует, что каждый экземпляр (индивид) класса наследника является экземпляром базового класса. Ввиду совпадения семантики данного понятия в ходе реализации инструмента трансляции было принято решение транслировать наследование без использования дополнительных эвристик.

Большинство объектно-ориентированных языков программирования не поддерживают множественное наследование в связи с проблемой неоднозначности наследования отдельных членов класса. Тем не менее, при построении предметной области подобная конструкция допустима.

Множественное наследование влечет за собой проблему неоднозначного наследования одинаковых свойств и методов родительских классов. Данная проблема решается либо путем явного указания родителя при наследовании общих элементов классов, либо введением интерфейсов. Наибольшее распространение получило использование интерфейсов, поэтому стоит ввести правило ограничения наследования классов и добавить интерфейсы в процесс трансляции.

Поля класса по типам данных можно разделить на примитивные типы (*xsd shema*) и пользовательские (атрибут содержит объект класса, атрибут типа перечисление и т.д.).

Поля примитивного типа данных транслируются в онтологию как *DataTypeProperty* с *Domain*, равным классу, которому принадлежит атрибут и *Range*, равным примитивному типу. Перечисление транслируется в заданный пользователем тип данных, для которого определены возможные значения и *DataTypeProperty*. Атрибут типа класса транслируется в *ObjectProperty* с *Domain*, равным классу, которому принадлежит атрибут и *Range,* равным классу типа атрибута.

Шаблон проектирования добавляется в онтологию в виде набора индивидуалов, принадлежащих к классам, описанным выше. Семантические ограничения и свойства шаблонов проектирования описываются в онтологии с помощью объектных свойств и свойств типов данных. Так как в онтологии хранятся несколько шаблонов проектирования одновременно, необходимо ввести правило именования для их элементов, чтобы избежать дублирования имен.

Имя элемента шаблона проектирования начинается с названия шаблона, а затем через подчеркивание пишется название элемента. Для именования отношений, входящих в состав шаблона проектирования, действует похожее правило. Название отношения так же начинается с названия шаблона проектирования, затем идут названия элементов, которые связаны данным

отношением с учетом направленности. Первым в названии отношения идет название элемента, от которого идет отношение, а вторым, название элемента к которому идет отношение.

Для хранения знаний, извлеченных из шаблонов проектирования, можно использовать разработанную раннее онтологию на основе метамодели языка UML.

Трансляция интерфейсов предполагает создание в онтологии класса, на который наложен запрет на создание индивидуалов. При этом наследники класса, реализующие данный интерфейс, по логике ООП могут иметь индивидуалы. Предложено ввести свойства объекта (*ObjectProperty*) – *IsInterface* и *RealiseInterface*. Свойство *IsInterface* определяет, что класс онтологии, обладающий этим свойством, является интерфейсом. Свойство *RealiseInterface* утверждает, что класс онтологии, находящийся в области определения *Domain*, – это наследный класс, а класс онтологии в интервале значений *Range* является интерфейсом, и для него должно быть определено свойство *IsInterface*.

Практически все связи, применяемые при проектировании UML-диаграммы классов, не имеют прямого аналога в онтологическом представлении предметной области. Однако отношения между классами в OWL можно модифицировать, добавляя аксиомы, ограничения или свойства. Для трансляции UML-диаграмм в онтологию изначально создается набор отношений, характеризующих различные связи из диаграмм.

Исключением из правила является связь обобщения (*generalization*), которая имеет аналогичную по смыслу связь в онтологическом представлении (*subClassOf*).

Ассоциативная связь в UML не имеет прямого аналога в OWL. Данная связь означает, что объект одного класса связан с объектом другого класса. Агрегация и композиция по спецификации представляют собой разновидности ассоциации, реализующие отношения «часть-целое».

Ассоциации предлагается представлять в явном виде, как три отношения: два *ObjectProperty* для каждого атрибута связанных классов и *ObjectProperty* с названием, соответствующим наименованию ассоциации. Идея подобного способа выражения ассоциаций состоит в том, что такого рода связи выражают интерпретацию предметной области в виде программного продукта, но не являются частью этой предметной области.

Зависимость (*dependency*) – связь, которая означает, что при изменении класса зависимый от него класс также необходимо изменить с точки зрения нотации UML. Зависимость действует не на уровне объектов, а на уровне программной реализации. Именно результат анализа связи зависимости послужил основой для разделения в онтологии знаний о самой предметной области и о программном продукте, построенном на ее основе. Результат трансляции связи зависимости представляет собой одно свойство *ObjectPrope*rty, у которого ранг *Range* хранит зависимый класс, а *Domain* – класс, от которого существует зависимость.

Обратная трансляция при подобном способе трансляции осуществима за счет однозначной трактовки результата трансляции и уникальных значений элементов, полученных из XMI.

3.2.4 Экранные формы модуля извлечения знаний из UML диаграмм

Модуль извлечения знаний из UML диаграммы классов реализован на языке программирования C# в среде разработки *Visual Studio 2017*. Данный модуль позволяет осуществить выбор проектной диаграммы в экспортированной в формат XMI (рис 3.4).

Рис. 3.4. Загрузка проектной диаграммы в формате xmi.

После выбора диаграммы начнется процесс ее импорта, сначала во внутренний формат программы, а затем в вид онтологического представления. Иерархия классов представлена в виде древовидной структуры на рисунке 3.4:

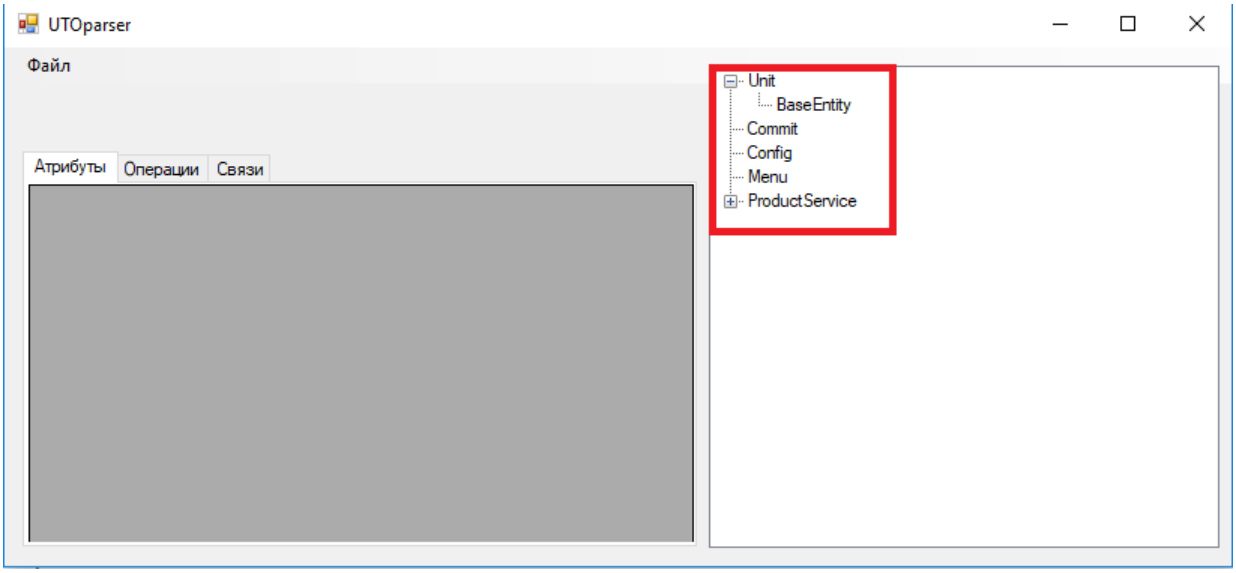

Рис. 3.5. Элементы, извлеченные из диаграммы классов, представленные в виде иерархии классов.

При выборе одного из узлов дерева, можно посмотреть элементы его внутреннего устройства, которые были выделены в результате анализа проектной диаграммы.

На рисунке 3.6 представлены все аттрибуты класса *BaseEntity* у учетом их типов.

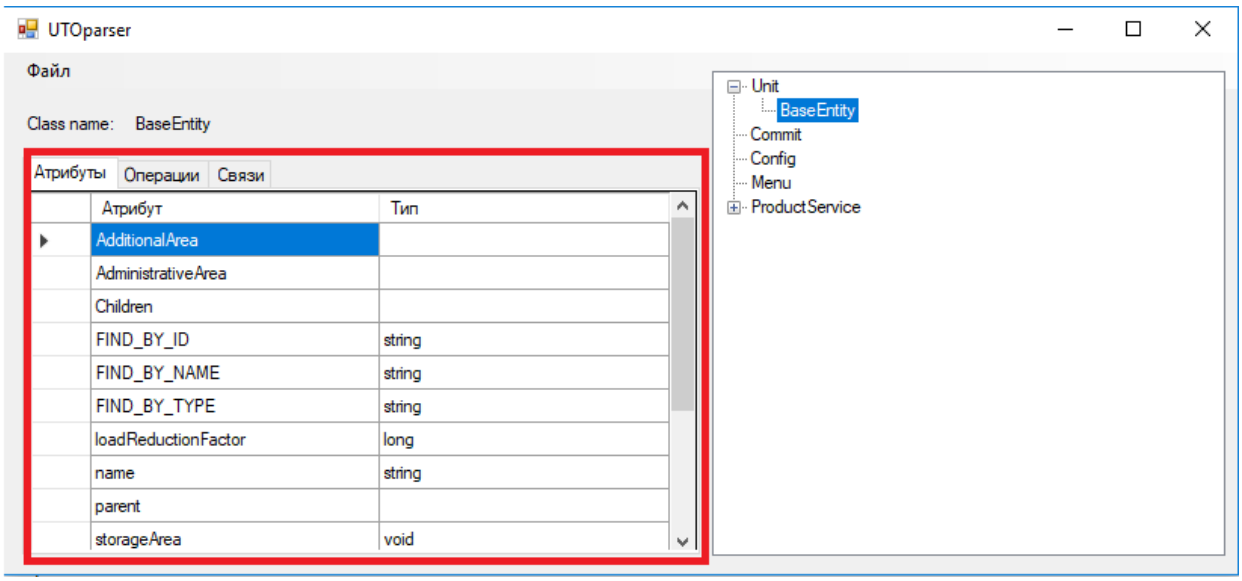

Рис. 3.6 Просмотр атрибутов класса *BaseEntity*.

На рисунке 3.7 представлены все операции класса *BaseEntity* у учетом их возвращаемых значений.

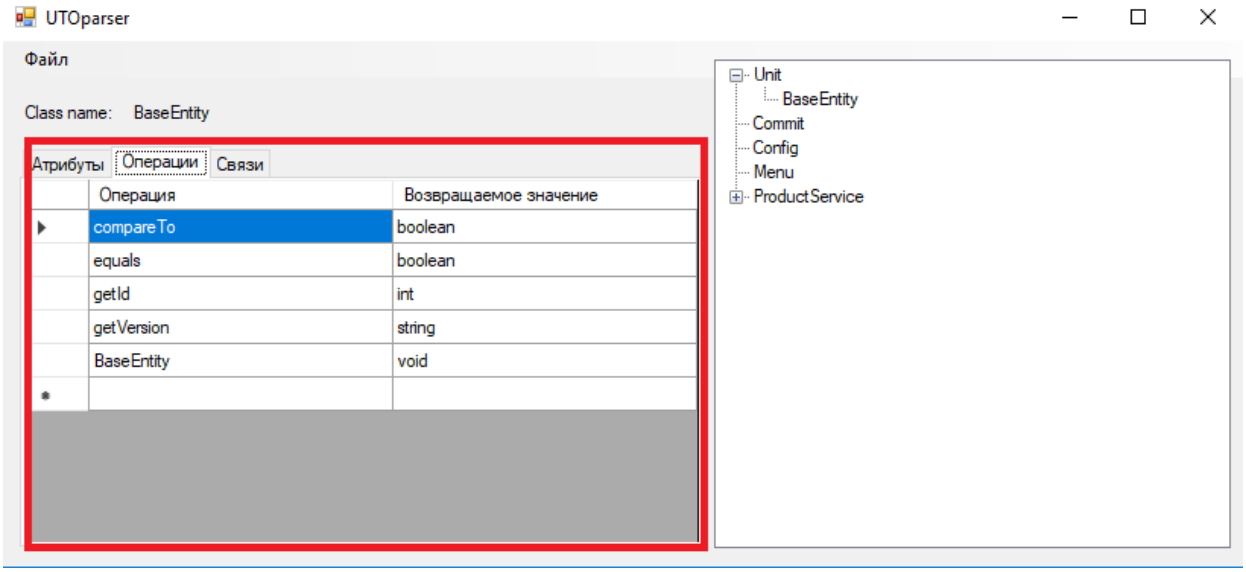

Рис. 3.7 Просмотр операции класса *BaseEntity*.

На рисунке 3.8 представлены все связей класса *BaseEntity* у учетом их типов.

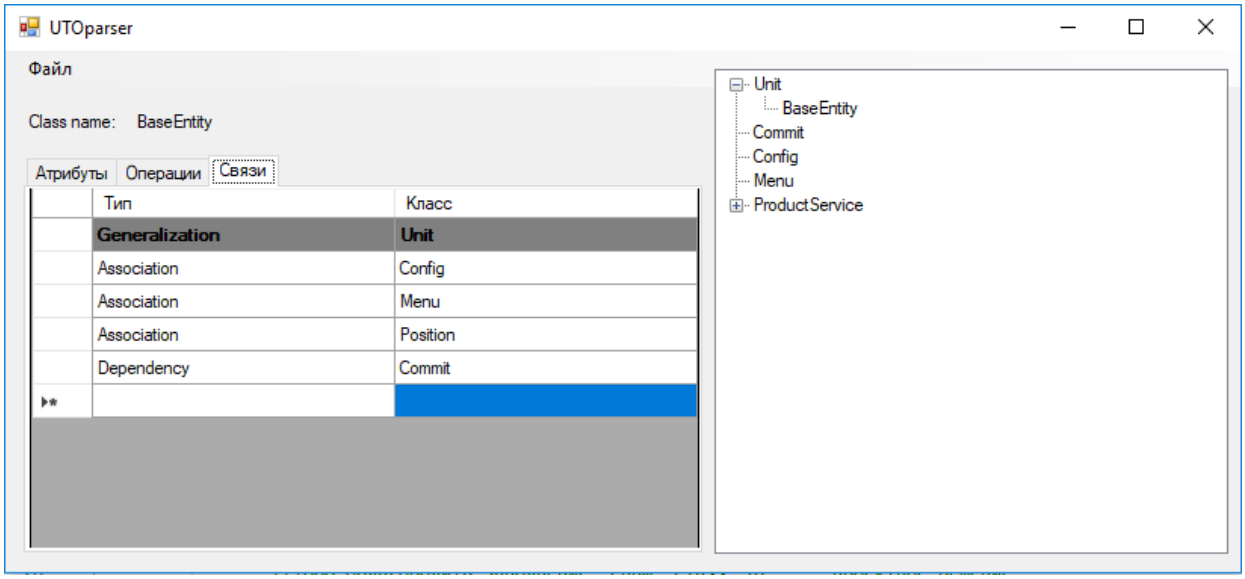

Рис. 3.8. Просмотр связей класса *BaseEntity*.

3.2.5 Экранные формы модуля извлечения знаний из исходного кода

Большая часть системы ИП реализована на *Java* в среде разработки *IntelliJ Idea 2018.1*. Система предполагает наличие CRUD интерфейса для проектов и шаблонов проектирования. Список проектов, добавленных в систему, представлен на рис. 3.9.

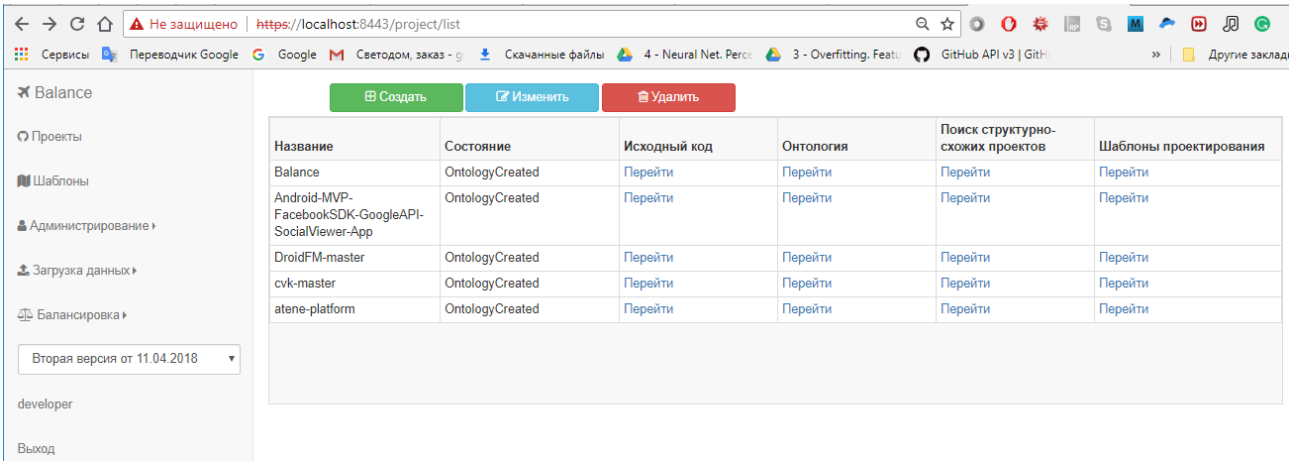

Рис. 3.9 Список проектов, добавленных в систему ИП.

Специалист может добавить все проекты организации в систему с помощью окна, показанного на рисунке 3.10, вызываемого по кнопке «Создать»:

Рис. 3.10 Добавление нового проекта в систему ИП.

После того, как проект создан, необходимо добавить исходный код проекта для формирования его онтологического представления (рисунок 3.11). Если исходного кода проекта на момент взаимодействия с системой еще нет, но есть UML-диаграмма классов, то можно воспользоваться модулем, описанным в пункте 3.2.3, и сразу добавить онтологическое представление.

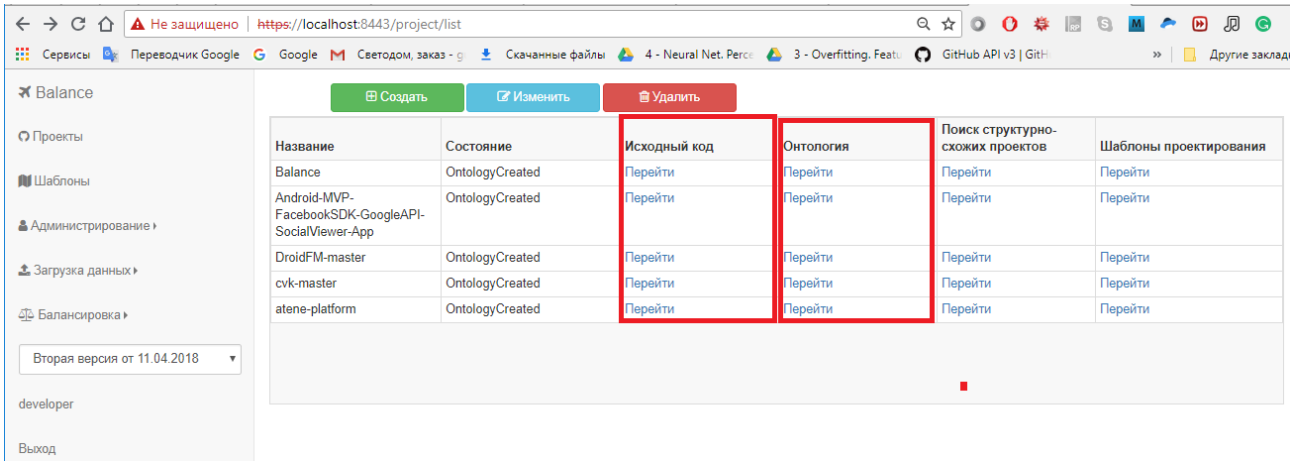

Рис. 3.11 Формирование онтологического представления с помощью исходного кода проекта или его прямая загрузка.

После окончания формирования онтологического представления (рисунок 3.12) проекта, его можно загрузить в формате owl и откорректировать вручную, если это необходимо.

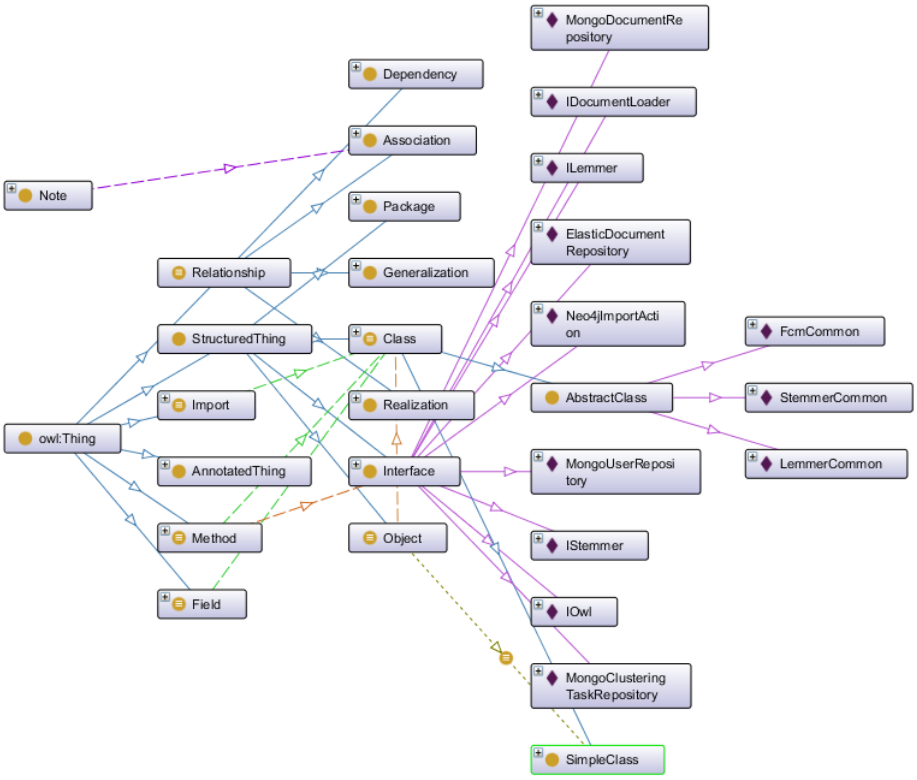

Рис. 3.12 Онтограф полученный в результате загрузки исходного кода проекта *Android-MVP-FacebookSDK*.

3.2.6 Экранные формы модуля вычисления меры архитектурного подобия проектов

После того как знания о ПАК, находящемся на стадии проектирования преобразованы в онтологическое представление, полученное ПО можно проиндексировать с помощью шаблонов проектирования. Каждый модуль ПО может индексироваться на своем уникальном множестве шаблонов проектирования. Для процесса индексации в системе ИП предусмотрена следующая форма, представленная на рисунке 3.13

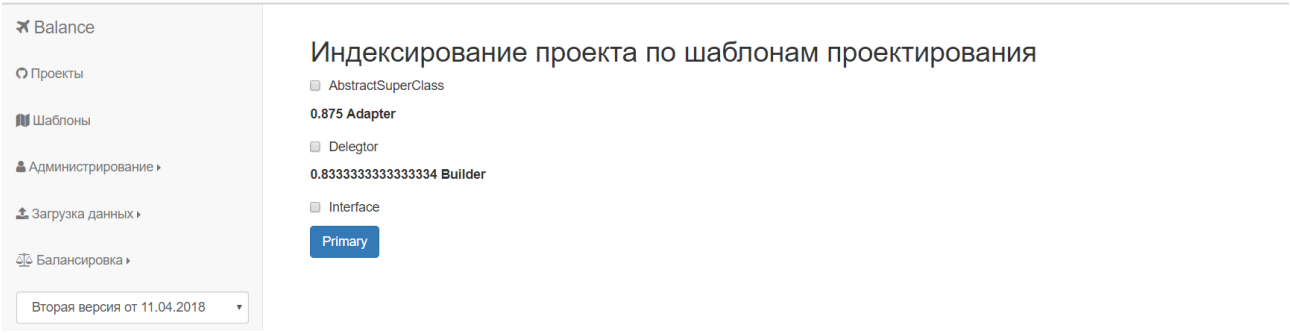

Рис. 3.13 Форма индексирования ПО по шаблонам проектирования.

В форме необходимо отметить шаблоны проектирования, характерные для ПО данной предметной области. Если ПО уже было проиндексировано на соответствие шаблону проектирования, то напротив его названия будет указана мера выраженности.

Для определения меры архитектурного подобия предусмотрена отдельная форма (рисунок 3.14).

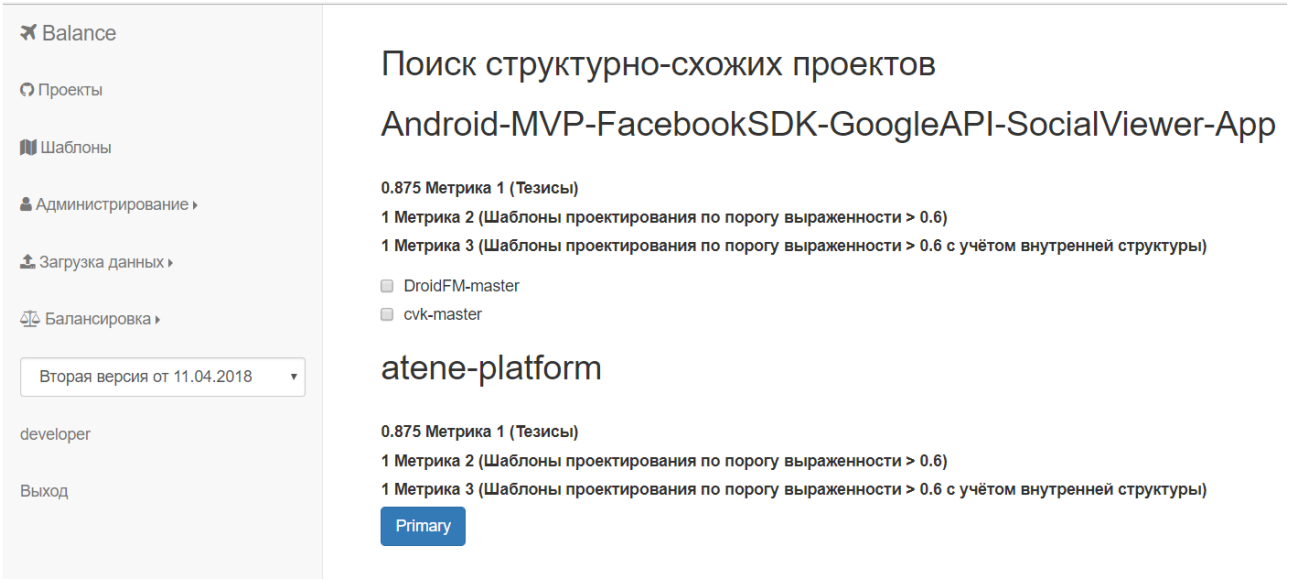

Рис. 3.14 Форма расчета меры архитектурного подобия проектов.

На рис. 3.14 представлены все загруженные в систему проекты. Если мера архитектурного подобия не была определена, то проект доступен для расчета. Если же расчет уже производился, то будут выведены значения по трем мерам архитектурного подобия проектов.

3.3. Подсистема анализа данных

Применения подсистемы анализа данных

Подсистема анализа временных рядов первоначально разрабатывалась как независимый компонент и представляла собой набор реализованных моделей прогнозирования временных рядов (ВР). Система представлена в виде сервиса по электронному адресу - http://tsas.ulstu.ru/.

Подсистема анализа данных была модифицирована и расширена в рамках выполнения хозяйственного договора с ООО «Эверест Ресерч». В результате модификации в подсистему были внесены следующие изменения:

- переработана архитектура подсистемы, выделена расширенная иерархия классов;
- проведена доработка и оптимизация существующего исходного кода;
- добавлены методы нормализации для разных типов временных рядов;
- добавлены классические модели прогнозирования ВР;
- расширен список нечетких моделей прогнозирования ВР;
- разработан алгоритм оптимизации параметров моделей прогнозирования ВР;
- добавлены алгоритмы агрегирования результатов прогноза моделей прогнозирования ВР.

Анализ временных рядов, описывающих процесс разработки ПО для технологической подготовки производства, представлен в пункте 4.5.2.

3.3.1 Требования к подсистеме анализа данных

Исходные данные

В соответствии с техническим заданием хозяйственного договора (Приложения 1, 4) временные ряды хранятся в файлах *.csv, где каждая строка формата «ДД.ММ.ГГГГ; значение» или «Номер; значение» в случае одинаковых временных интервалов между точками ВР. Количество точек ВР определяется количеством строк в файле. Для тестирования программной системы использовались временные ряды соревнования NN3.

3.3.2 Алгоритм функционирования подсистемы анализа данных

Алгоритм работы системы заключается в создании массива моделей экспоненциального сглаживания, каждая из которых формирует уникальный прогноз. Все модели являются параметризованными и могут быть оптимизированы. Результат прогноза по модели существенно зависит от подобранных значений параметров. Для корректного подбора параметров

можно использовать известную часть временного ряда или временной ряд со схожим поведением. Блок-схема алгоритма представлена на рисунке 3.15.

Для каждого временного ряда из множества, переданного для прогнозирования, будут выполняться следующие действия. Первым шагом алгоритма является нормализация значений исходного загруженного временного ряда к интервалу [0..1]. Подсистема содержит реализации нескольких подходов к нормализации данных. Специфические методы нормализации необходимы для корректной обработки временных рядов разных типов: разреженных временных рядов или временных рядов, содержащих дискретные значения 0 и 1.

На втором шаге подсистема определяет набор моделей по типу временного ряда, подходящих для прогнозирования. Для каждой модели производится инициализация. Если эта модель уже использовалась ранее и имеет набор оптимально подобранных параметров, то она будет загружена из хранилища. В случае, если модель применяется для временного ряда такого типа впервые, будет произведена оптимизация параметров для данной модели. Пользователь может вручную указать принадлежность временных рядов к типу, что приведет к использованию ранее рассчитанных оптимизированных для данной модели параметров ко всем временным рядам заданного типа.

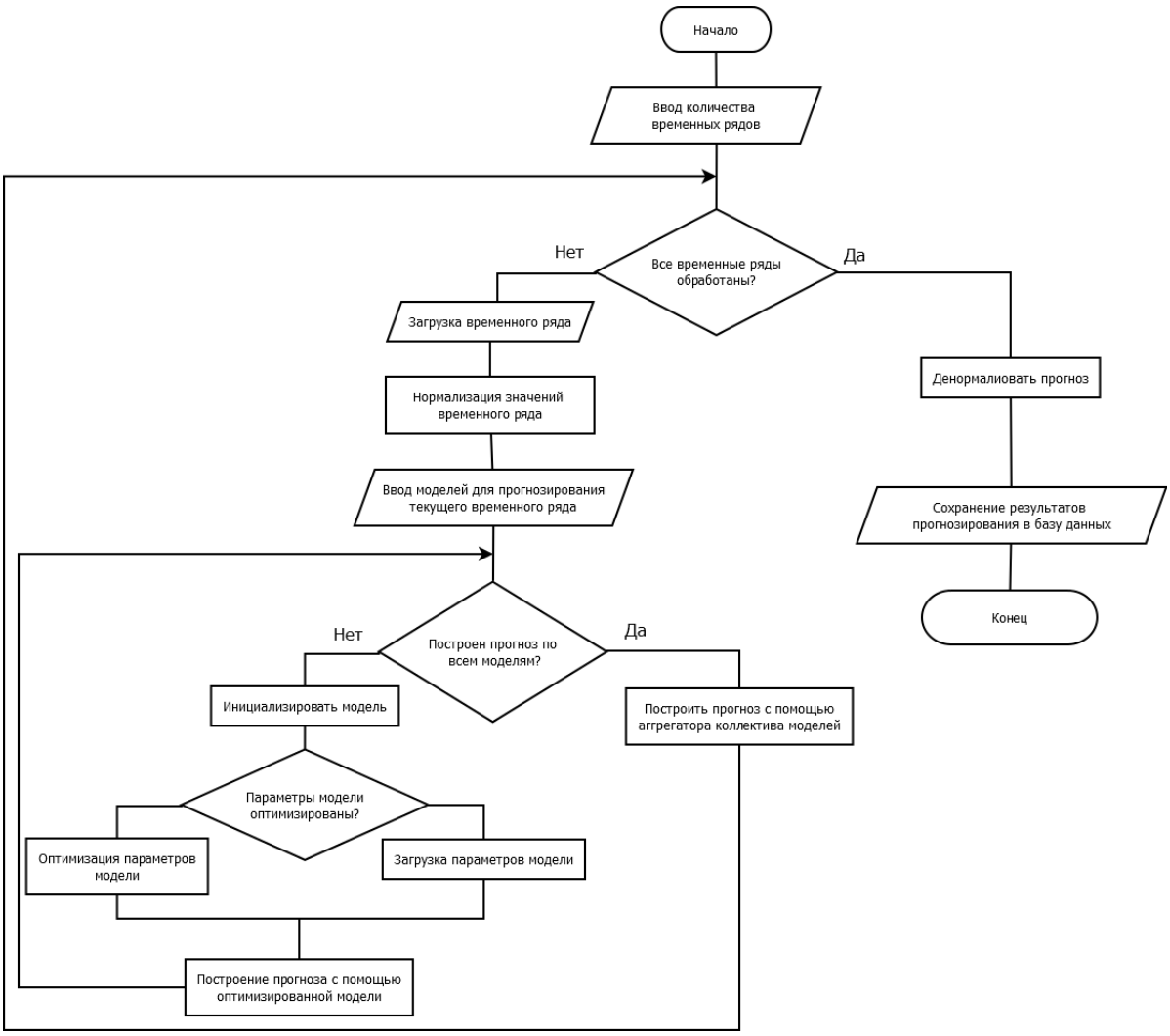

Рис. 3.15 Общий алгоритм функционирования подсистемы анализа данных.

Оптимизация параметров производится путем последовательного перебора по сетке значений. Шаг сетки может настраиваться пользователем и оказывает существенное влияние на скорость работы системы. При уменьшении интервала изменения параметра при оптимизации увеличивается точность прогноза, но при этом увеличивается и время, требуемое на расчеты. Например, в модели экспоненциального сглаживания, не учитывающей тренд и сезонную компоненту, параметр α может быть определен с шагом в 0.01 от 0 до 1, а в модели, построенной на основе нечетких множеств, шаг может быть выбран равным 0.3, поскольку во второй модели больше общее количество параметров.

Так же в системе реализован алгоритм более быстрого поиска оптимального значения параметра относительно простого перебора по сетке с

интервалом. Данный алгоритм основан на алгоритме бинарного поиска и не лишен основного недостатка подобного рода алгоритмов, которые останавливаются на локальном оптимуме.

После отработки основного массива методов становится возможным создание коллектива методов, на вход которого поступает массив моделей экспоненциального сглаживания с оптимальными параметрами. Коллектив формирует прогноз, анализируя одношаговые прогнозы всех моделей и назначая вес каждой из них.

После построения прогноза с помощью каждой модели, программа создает коллектив моделей, агрегирующий прогнозы каждой модели. Коллектив моделей представляет из себя объект, поддерживающий интерфейс модели. На вход коллектива поступает массив ссылок на объекты моделей с заранее оптимизированными параметрами. Коллектив формирует свой уникальный прогноз, анализируя одношаговые прогнозы всех моделей и назначая вес для каждой из них на каждом шаге.

После построения прогноза всеми моделями и коллективами, выбранными пользователем, осуществляется денормализация прогнозов. Результаты прогнозирования по данному временному ряду сохраняются в хранилище.

3.3.3 Архитектура подсистемы анализа данных

Иерархия классов подсистемы анализа данных представлена на рисунке 3.16. Класс *Method* является абстрактным классом, описывающим модель. Все модели прогнозирования, реализованные в подсистеме анализа данных, являются наследниками класса *Method*. В классе *Method* определены сигнатуры всех методов, определяющих взаимодействие моделей прогнозирования с другими объектами подсистемы:

- деление исходного ряда на части обучения и тестирования,
- установка параметра модели,
- вызова алгоритма оптимизации,

- расчет прогноза.

Класс *Vovk* реализует работу коллектива моделей по методу Вовка. Статический класс *FuzzyMembership* – класс, содержащий реализации функции принадлежности для нечетких моделей. Функции принадлежности целесообразно вынести в отдельный класс, так как они являются общими для нечетких моделей и редко изменяются.

Класс *AicWeights* реализует взвешенный критерий Акаике. Данный класс описывает критерий оценки, но не включен в иерархию классов критериев оценки, так как он применим лишь для коллектива моделей и как следствие не соответствует базовому классу критериев оценки ни по внутренней структуре, ни по поведению.

Класс *Param*: осуществляет хранение и просмотр набора параметров для любой модели. Набор параметров для каждой модели уникален, но управление каждым отдельным параметром одинаковое. Так же класс *Param* позволяет установить взаимоотношения между параметрами каждой отдельной модели, например, если α больше 0.5, то β не может быть больше 3.

Класс *Normalization* является абстрактным классом и определяет базовые функции алгоритмов нормализации. В классе *Normalization* определены сигнатуры следующих методов:

- Сдвиг значений точек ВР в область положительных чисел;
- Приведение значений точек ВР к интервалу, как правило интервалу [0,1].

Класс *Preparator* реализует методы предобработки и оптимизации данных прежде, чем с ними начнут работать модели. Например, класс *Preparator* производит предварительные алгебраические преобразования.

Класс *File* реализует операции по работе с файлами для загрузки временных рядов и сохранению результатов прогнозирования.

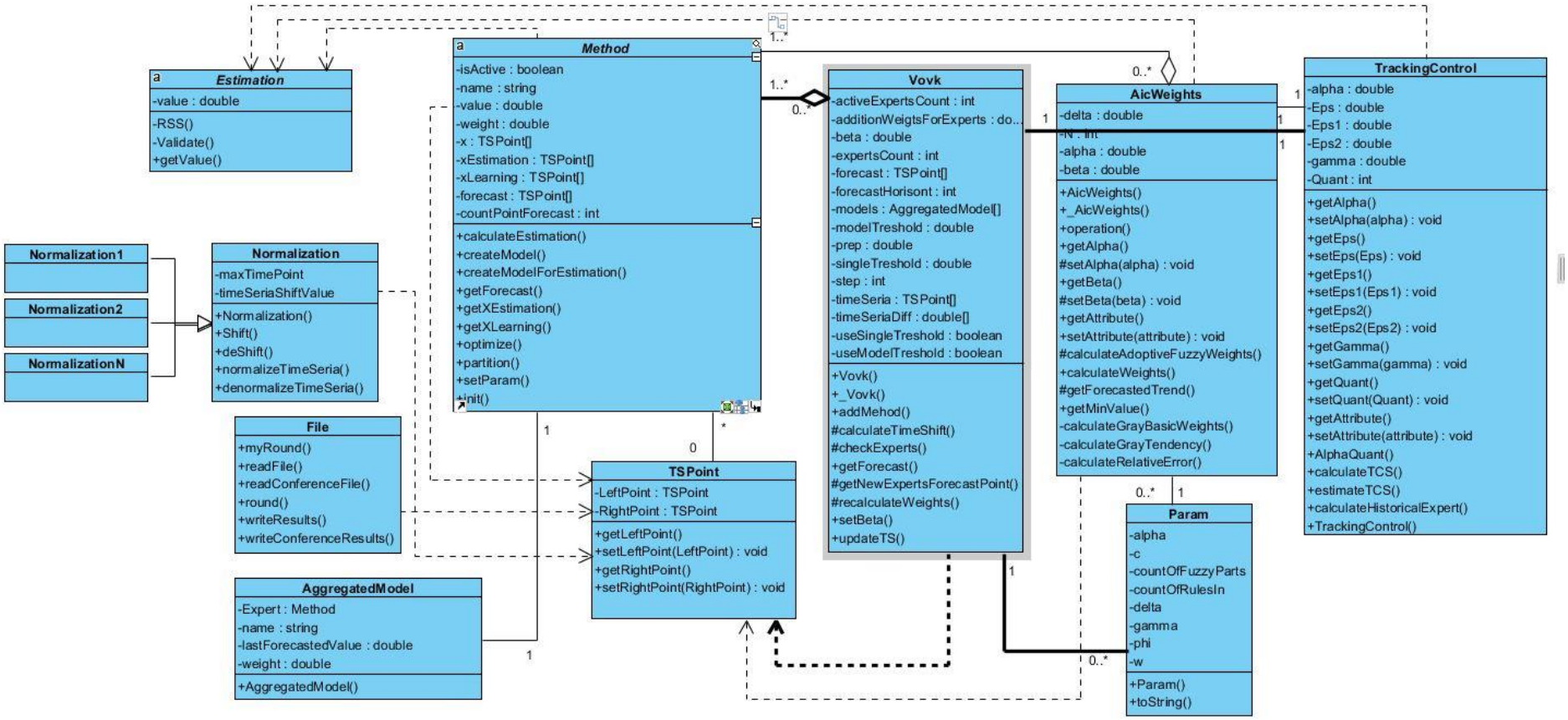

Рис. 3.16 ПАД. UML-диаграмма классов всей подсистемы анализа данных.

Так как составленная диаграмма классов достаточно велика, она будет рассмотрена в виде отдельных диаграмм классов для каждого модуля.

Первый модуль в подсистеме анализа данных аккумулирует иерархию классов, отвечающих за критерии оценки прогнозов временного ряда. UML-диаграмма классов модуля оценки прогноза представлена на рисунке 3.17.

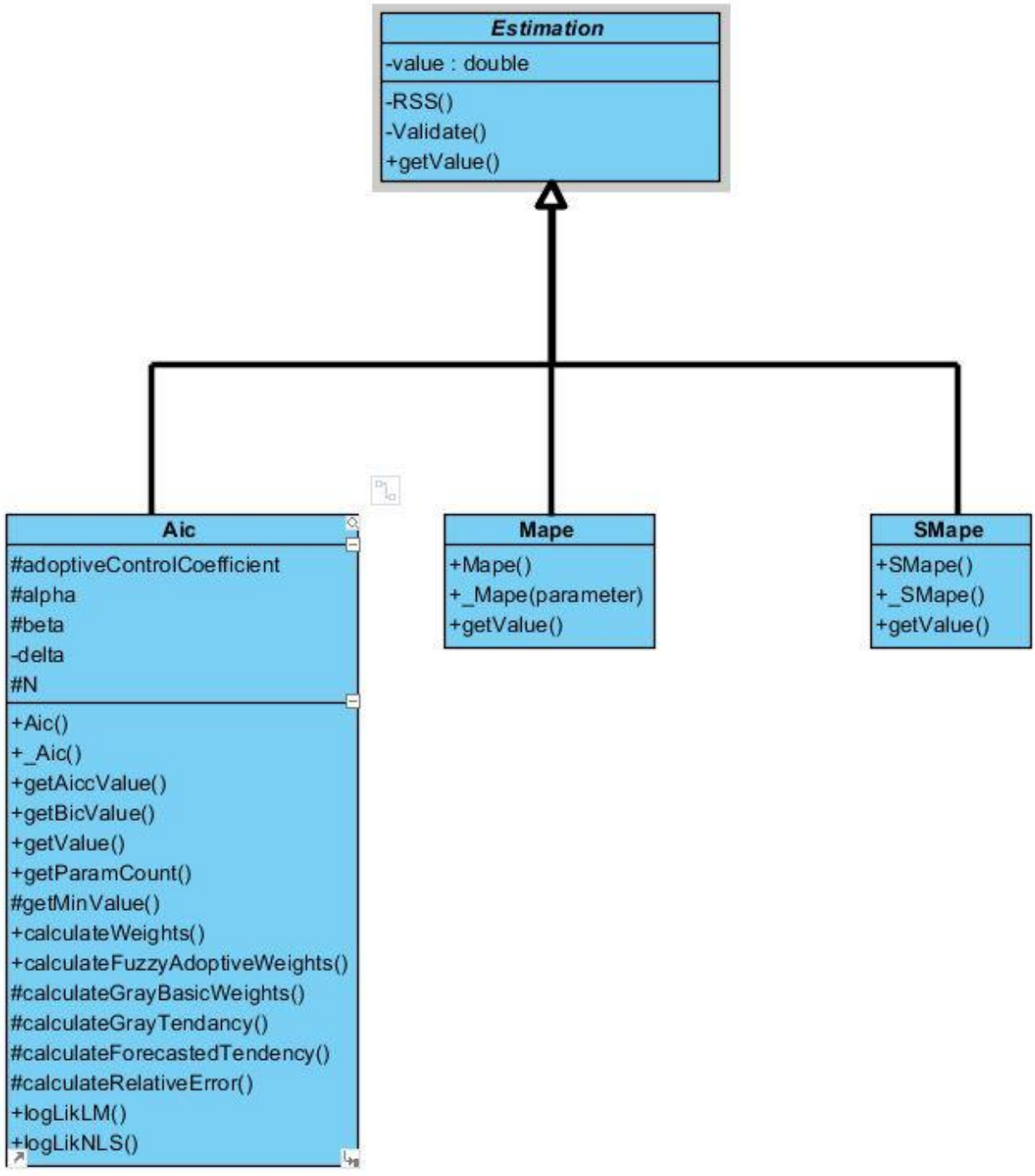

Рис. 3.17 Иерархия классов модуля оценки прогноза.

Estimation – абстрактный класс описывающий базовое поведение и внутреннее устройство объектов, отвечающих за критерии оценки.

Aic – класс, отвечающий за оценку прогнозов по критерию Акаике. Критерий Акаике был использован как основной критерий для отладки моделей при разработке, так же он используется моделями для расчета прогноза.

SMape - класс описывающий критерий оценки *SMape* (*Symmetric mean absolute percentage error*). Критерий *SMape* отражает среднюю абсолютную симметричную ошибку в процентах. Данный класс был добавлен в подсистему для того, чтобы принять участие в конкурсе CIF 2015 (*Computational intelligence in forecasting*). В конкурсе CIF 2015 критерий SMape был использован в качестве основной оценки качества прогноза.

Mape – класс описывающий критерий оценки Mape (*mean absolute percentage error*). Критерий Mape отражает среднюю абсолютную ошибку в процентах.

Большая часть реализованных в системе моделей прогнозирования использует один из представленных критериев для построения прогноза. Для создания прогноза с помощью агрегации оценки по коллективу моделей необходимо, чтобы все входящие в коллектив модели были построены на каком-либо одном критерии.

Классы, отвечающие за реализацию критериев оценки качества прогнозирования (SMape, Mape, Aic), имеют иерархическую структуру, поскольку родительский класс *Estimation* используется как тип в методах оптимизации параметров. Было принято такое решение, поскольку в качестве критерия для оптимизации параметров может выступать любой из реализованных.

Второй модуль в подсистеме анализа включает в себя иерархию моделей прогнозирования временных рядов. Каждая модель представлена в подсистеме отдельным классом и наследуются от абстрактного класса *Method*.

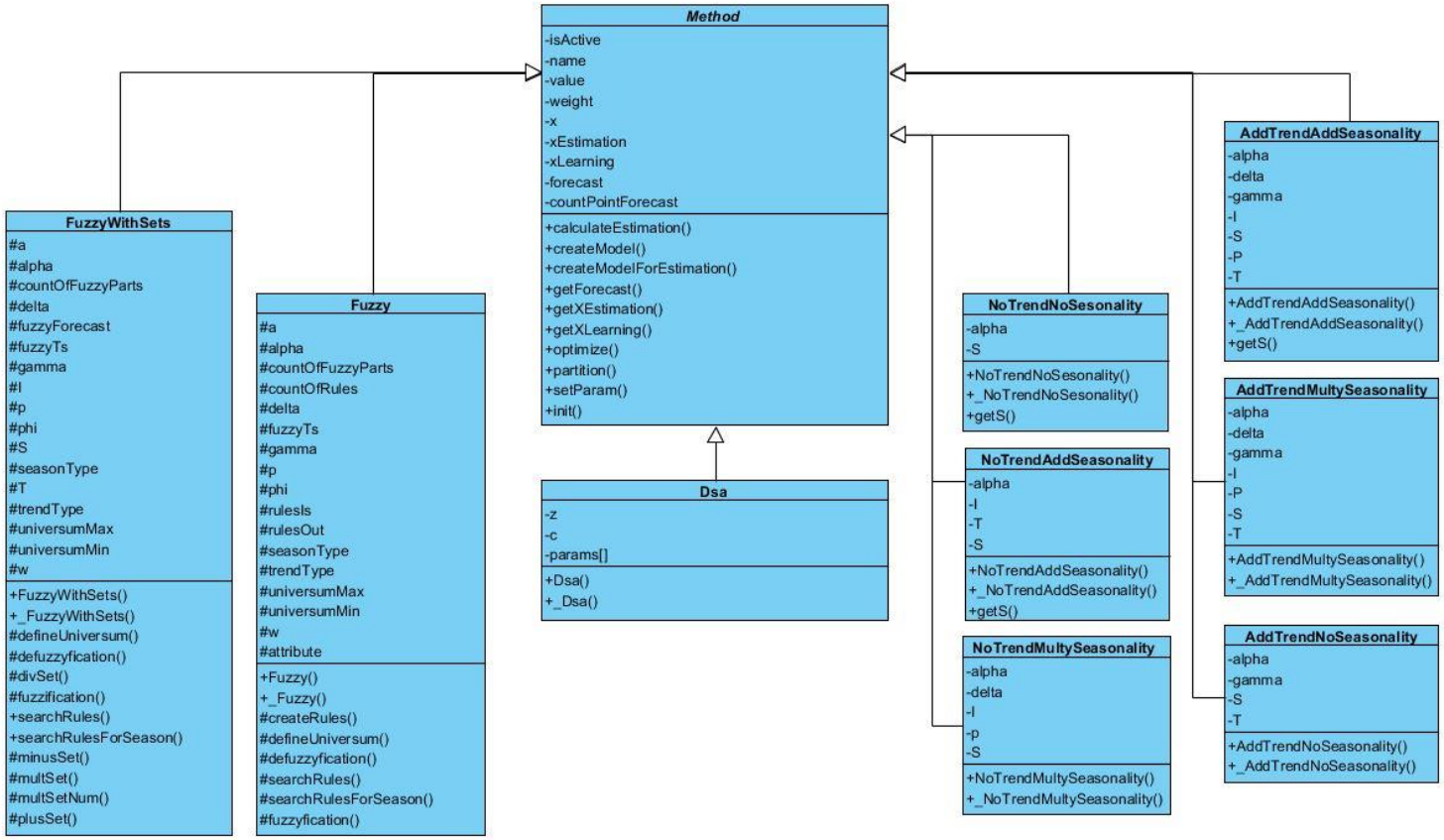

Рис 3.18 Иерархия моделей прогнозирования часть 1.

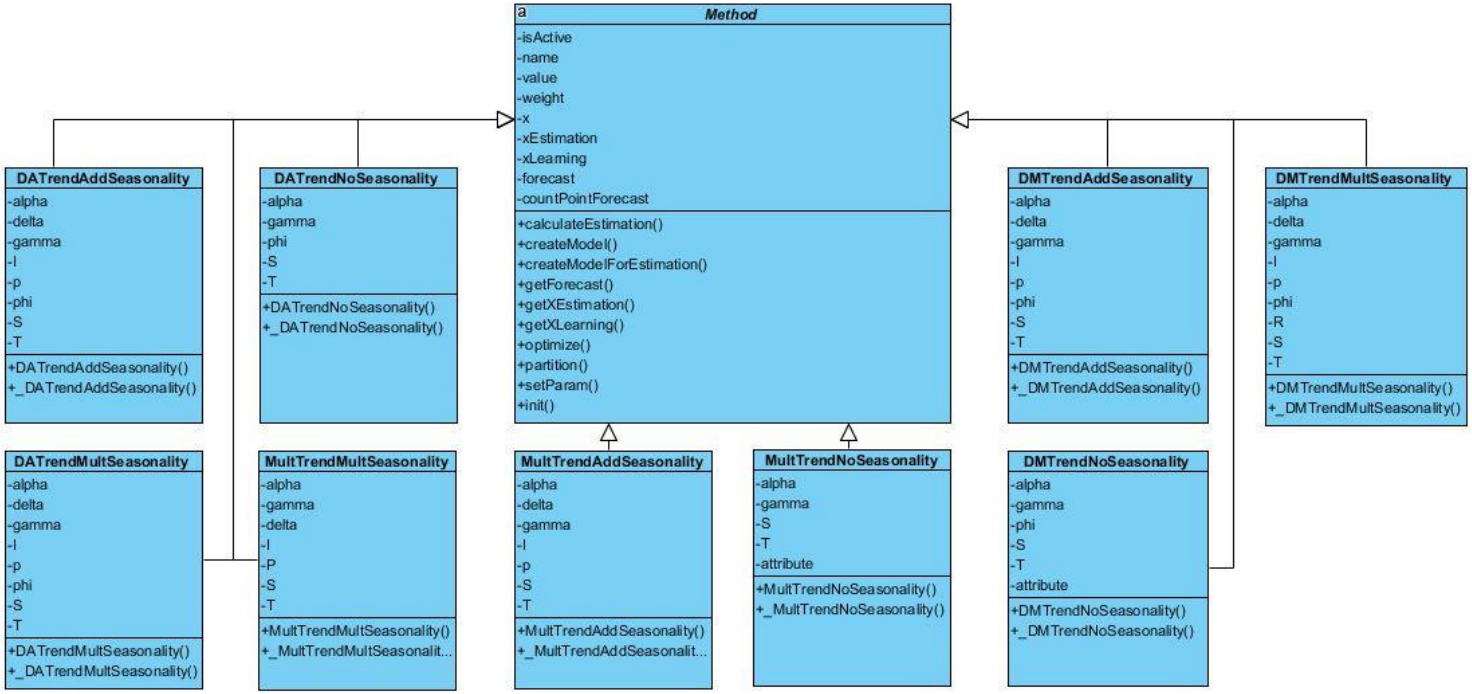

Рис. 3.19 Иерархия моделей прогнозирования часть 2.

3.4 Выводы

При решении задач, поставленных в рамках диссертационного исследования, была разработана программная система, состоящая из двух подсистем:

1. Подсистема поддержки интеллектуального проектирования (ПИП);
2. Подсистема анализа данных (ПАД);

Подсистема ПИП состоит из следующих обособленных модулей:

- Модуль извлечения знаний из UML-диаграмм;
- Модуль извлечения знаний из исходного кода;
- Модуль вычисления меры архитектурного подобия проектов;

Подсистема ПАД состоит из следующих обособленных модулей:

- Модуль прогнозирования временных рядов с помощью классических моделей экспоненциального сглаживания;
- Модуль прогнозирования временных рядов с помощью нечетких моделей;
- Модуль агрегации прогнозов комбинации или коллектива моделей;
- Модуль оценки результатов прогнозирования на основе информационных критериев;
- Модуль формирования рекомендаций по управлению создания ПАК.

Данная система была использована в ходе выполнения НИР «Система интеллектуального поиска и анализа в Интернет-СМИ и социальных сетях» для ФНПЦ АО «НПО «Марс». На этапе проектирования ИС реализующей основные задачи НИР была использована подсистема ПИП для отбора проектов из открытого репозитория *github.com*, наиболее подходящих по предметной области и архитектурному подобию.

Подсистема ПАД была использована в ходе реализации ПО НИР «Система интеллектуального поиска и анализа в Интернет-СМИ и социальных сетях», с ее помощью осуществлялся контроль над наиболее трудозатратными задачами проекта. Подсистема ПАД применялась для анализа временных рядов, извлеченных из истории разработки текущего проекта и схожих проектов. Результатом анализа были рекомендации по управлению разработкой, например, изменение времени итерации или проведение дополнительного совещания.

Для управления проектом «Оптимизация ресурсов на основе производственно-технологического моделирования в условиях мультипродуктовой производственной программы» разрабатываемый в рамках выполнения проекта №2.4760.2017/8.9 Министерства образования и науки России в партнерстве с АО «Авиастар-СП» была необходима система, позволяющая выделять наиболее сложные и нетривиальные задачи, требующие повышенного внимания.

Поиск архитектурно подобных проектов во внутреннем репозитории организации с помощью подсистемы ПИП оказался малоэффективным, так как все внутренние проекты были выполнены на языках программирования отличных от Java и C-подобных языков в целом, кроме того многие из них включали модули, содержащие коммерческую тайну. Так как предметная область данного проекта уникальна для предприятия заказчика, поиск среди проектов в открытых репозиториях оказался не эффективным.

Подсистема ПАД использовалась для анализа знаний, извлеченных из системы контроля версий в виде временных рядов. В качестве обучающей выборки для моделей прогнозирования использовались временные ряды из системы контроля версий той же команды разработчиков, сформированные при работе над другими проектами.

Глава 4 Вычислительные эксперименты и проверка релевантности системы СИПР

4.1 Анализ элементов диаграмм классов и их использования

В ходе разработки подсистемы ПИП был проведен тест модуля извлечения знаний из проектных диаграмм. Эксперимент проводился на 30 проектах, загруженных из открытого репозитория исходного кода github.com. Задачей данного эксперимента стало выявление наиболее часто используемых разработчиками элементов проектных диаграмм. Отталкиваясь от результатов данного эксперимента, можно построить оставшиеся модули подсистемы. Гистограмма частотности использования представлена на рисунке 4.1

Для оценки частотности использования элементов были отобраны проекты, разработанные с помощью языка программирования Java. В рамках диссертационного исследования был выбран язык Java, так как он предполагает объектно-ориентированный стиль разработки и построение сложных архитектурных абстракций, таких как шаблоны проектирования.

Проекты были отобраны по следующим принципам:

- Разнообразие проектов в выборке по размеру (количеству ветвлений и коммитов);
- Разнообразие проектов в выборке по популярности (рейтинг проектов в рамках github.com);
- Разнообразие проектов по предметной области (базы данных, web-приложения, библиотеки и т.д.).

Для проведения эксперимента использовались самые последние версии рассматриваемых проектов. Перечень проектов, которые были исследованы в рамках эксперимента: javaparser/javaparser, Nightonke/GithubWidget, Tencent/RapidView, Nightonke/GithubWidget, Tencent/RapidView, androidstarters/android-starter, NikoYuwono/ToolbarPanel,

dbachelder/CreditCardEntry, libgdx/packr, good-life/PushTalk, DesignativeDave/androrat, wasabeef/Takt, boxme/SquareCamera, anupcowkur/Android-Wheel-Menu, ditclear/TimeLine, freezy/android-xbmcremote, nekocode/CameraFilter, krasa/EclipseCodeFormatter, palantir/docker-compose-rule, yannecer/NCalendar, hexene/LocalVPN, inaka/TinyTask, allegro/grunt-maven-plugin, rafaelsteil/jforum3, pulse00/Symfony-2-Eclipse-Plugin, jhusain/learnrxjava, cenwenchu/beatles, apache/tomcat80, dnmilne/wikipediaminer, alexandrius/accordion-swipe-layout.

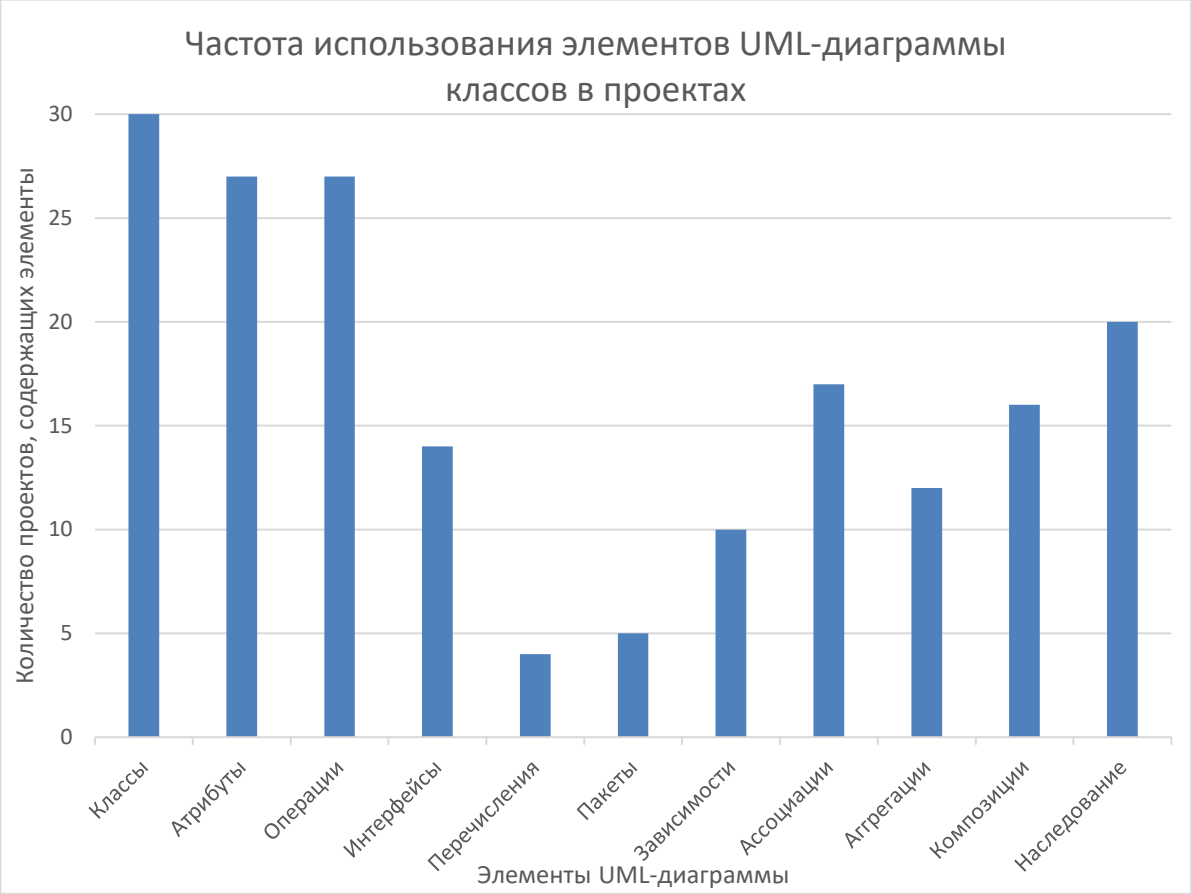

Рис. 4.1. Гистограмма частотности использования элементов в проетах.

Набор наиболее часто используемых элементов нотации UML повлиял на формирование T-Box онтологии языка UML. Косвенным результатом данного эксперимента стала идея о недостаточной семантической силе отдельных элементов UML и переход к рассмотрению шаблонов проектирования.

В результате трансляции имели место некоторые исключительные ситуации, описанные в главе 3, но все элементы диаграмм, были успешно транслированы в онтологию. Стоит отметить, что разработчики редко пользовались широким спектром элементов. Гистограмма частотности, свидетельствует, что выбор элементов в представленном исследовании был сделан верно [18].

4.2 Поиск шаблонов проектирования в проектах публичного репозитория Github

Для проверки предлагаемого подхода к выделению шаблонов проектирования в проектах был проведен отдельный эксперимент, целью которого был поиск шаблонов проектирования в проектах, загруженных из публичного репозитария github.com. Для проведения эксперимента в онтологию была добавлена информация по следующим 10 шаблонам проектирования: Delegation, Interface, Abstract superclass, Builder, Adaptor, Singleton, Bridge, Façade, Decorator, Observer [16].

Выбранные шаблоны проектирования отличаются по количеству элементов и связей между ними. Количество элементов варьируется от 3 до 20.

Для поиска были отобраны только проекты, разработанные с помощью языка программирования Java, так как система адаптирована к данному языку программирования и добавленные UML диаграммы шаблонов проектирования были разработаны с ориентацией на язык Java. В идеальной ситуации шаблон проектирования должен быть абсолютно независим от языка программирования. По крайней мере он должен быть реализуем на любом другом объектно-ориентированном языке. Но на практике различные реализации шаблона проектирования отличаются даже для объектно-ориентированных языков.

Так как общее число проектов в репозитории *Github*, разработанных на языке Java, очень велико, то необходимо ограничить выборку проектов для

эксперимента. В качестве ограничения будем использовать текстовый поиск, реализованный в интерфейсе репозитария. Получим две выборки проектов при помощи текстового поиска по словосочетанию "*design patterns*".

В результате запроса "vk api" было получено 108 проектов, относящихся к данной социальной сети. Результатом по запросу "design patterns" была выборка из 6516 проектов. Для проведения тестирования выборка была ограничена первыми 100 проектами по обоим запросам.

Вторая выборка, по ключевому слову "design patterns" необходима, чтобы проверить работу системы в условиях повышенного содержания шаблонов проектирования в проектах. Результаты поиска шаблонов проектирования представлены в виде гистограмм на рисунках 4.2 и 4.3.

Поиск шаблонов проектирования осуществлялся по совпадению структуры без учета названий отношений элементов и элементов вне структуры. Такие ограничения были введены, потому что совсем не обязательно структурные элементы шаблона проектирования, примененного в реальном проекте, будут соответствовать названиям из рассматриваемых при наполнении онтологии источников.

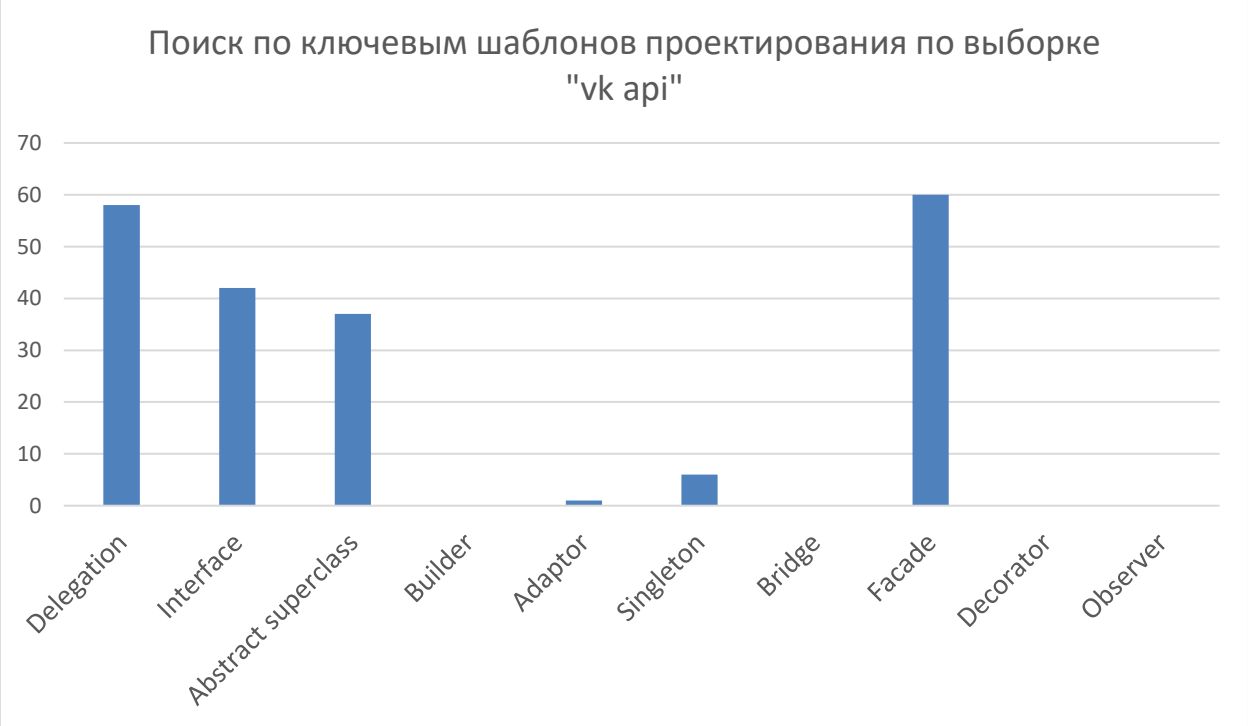

Рис. 4.2. Результаты поиска шаблонов среди проектов, полученных по запросу "vk api".

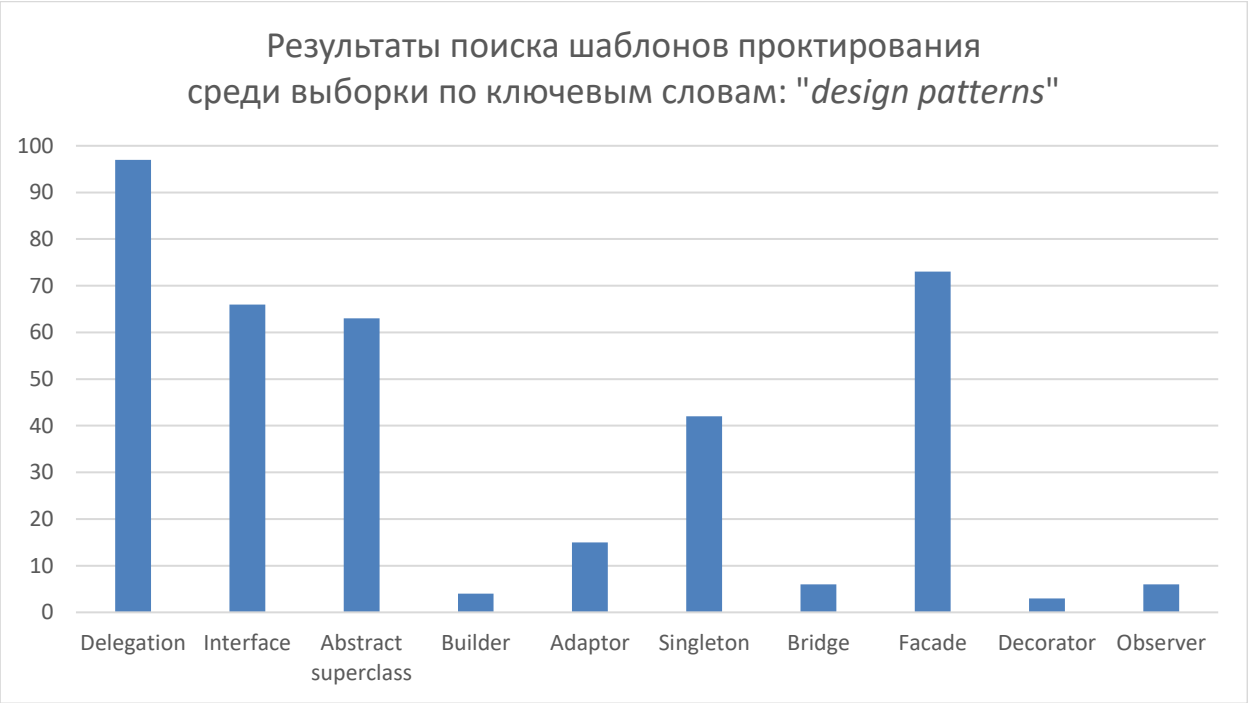

Рис. 4.3 Результаты поиска шаблонов среди проектов, полученных по запросу "*design patterns*".

Результаты экспериментов представляются вполне реалистичными для текущего уровня развития подхода. Высокие результаты для следующих шаблонов: *Delegation*, *Interface*, *Abstract superclass* и *Facade* объясняются за счет их простой структуры. Под простотой структуры понимается сравнительно небольшое число структурных элементов и, как следствие, связей. Данные шаблоны проектирования могли применяться разработчиками неосознанно, или же они могут совпадать по структуре с частью более сложного шаблона.

Шаблонов проектирования *Builder*, *Adapter*, *Bridge*, *Decorator* и *Observer* в контрольной группе проектов нашлось относительно немного. Редкость этих шаблонов объясняется их сложной структурой, содержащей большое количество элементов.

По результатам проведенного эксперимента можно сделать вывод о целесообразности применения подхода и построения модуля анализа над результатами поиска шаблонов проектирования в проектах.

Для проекта, выполняемого совместно с АО «Авиастар-СП» (г. Ульяновск) проводился поиск структурно-семантически схожих проектов, основанный на выборке по ключевым словам: "SCADA", "CNC" и "FOCAS Driver". В данном эксперименте в набор шаблонов проектирования входило 6 шаблонов, характерных для предметной области проекта «Оптимизация ресурсов на основе производственно-технологического моделирования в условиях мультипродуктовой производственной программы», составленных при участии экспертов. ПАК на этапе проектирования сравнивался с проектами, находящимися на более поздних этапах жизненного цикла, что позволило скорректировать его развитие и избежать провала проекта. Динамика развития проекта и проектов схожих с ним представлена далее в пункте 4.5.

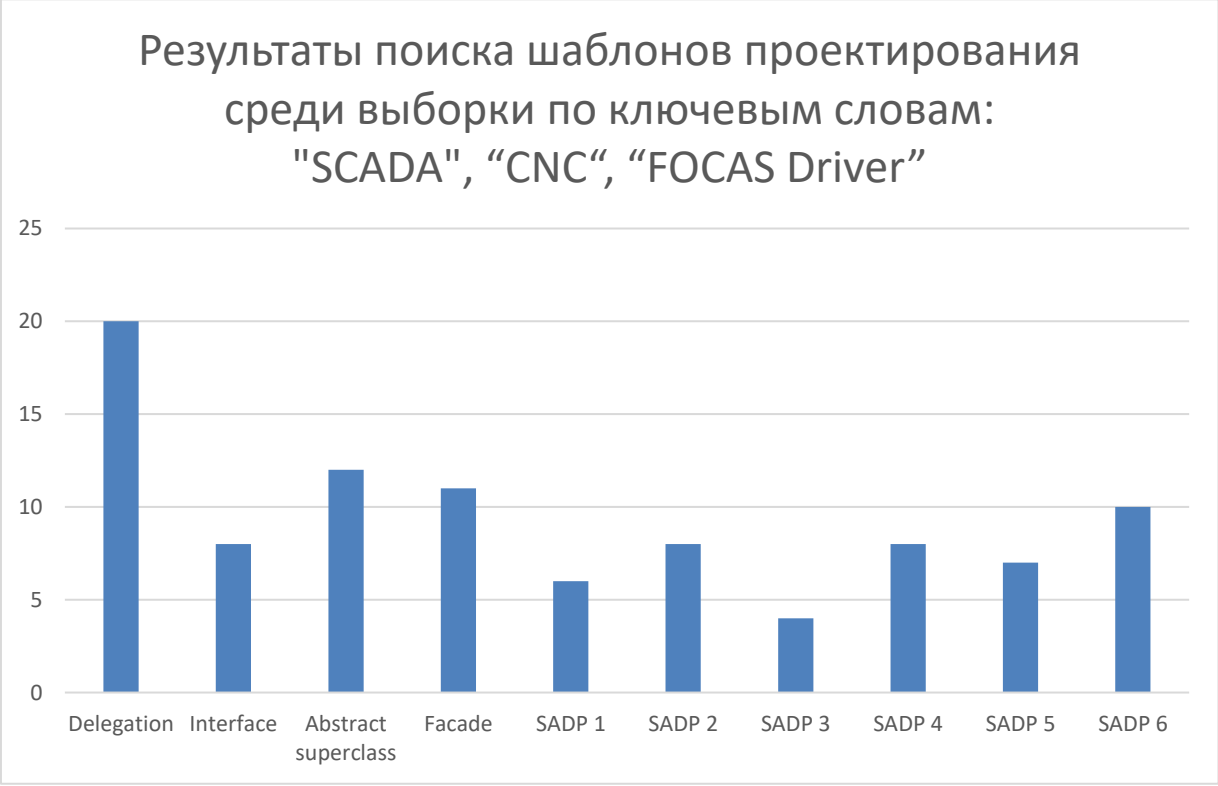

Рис. 4.4 Результаты поиска шаблонов проектирования среди проектов, полученных по запросу "*SCADA*", "CNC", "FOCAS Driver".

4.3 Результаты экспериментов поиска структурно схожих программных проектов

Для определения меры структурного сходства двух проектов, необходимо вычислить меру выраженности каждого шаблона проектирования в обоих проектах. [62].

Мера выраженности шаблона проектирования в проекте вычисляется с помощью наложения A-Box онтологии проекта на A-Box онтологии шаблона проектирования. В таблице 4.1 представлена степень выраженности каждого рассматриваемого шаблона проектирования во всех проектах экспериментальной выборки.

Проекты, рассматриваемые в рамках эксперимента, связаны между собой технологией, позволяющей извлекать публичные данные из широко известной социальной сети vk.com с помощью публичного API. Данный эксперимент был выполнен в рамках проекта «Разработка ПС интеллектуальной системы анализа данных». Основной целью эксперимента была формализация метрик для поиска структурно-схожих проектов с помощью решения реальной задачи по поддержке проектирования ПАК. Одним из компонентов ПАК должен был быть модуль взаимодействия с публичным API.

Таблица 4.1

Результаты экспериментов по вычислению меры выраженности шаблонов проектирования в проектах

Project name / Design pattern name	*Delegator (3)*	*Adapter(8)*	*Builder (12)*	*Abstract superclass (3)*	*Interface (5)*
Android-MVP	1.0	0.875	0.83	1.0	1.0
cordova-social-vk	1.0	0.875	0.83	1.0	0.8

cvk	1.0	0.875	0.92	1.0	1.0
DroidFM	1.0	0.875	0.92	1.0	1.0
VK-Small-API	1.0	0.625	0.42	0.33	0.6
VKontakteAPI	1.0	0.875	0.83	1.0	0.8
VK_TEST	1.0	0.75	0.58	0.66	0.6

Оценки меры выраженности шаблонов проектирования и структурного подобия нормированы от 0 до 1. Оценки меры сходства по первой метрике всегда равны 1, потому что данная метрика отбирает шаблон проектирования с максимальной мерой выраженности для каждого из двух сравниваемых проектов. А так как, например, шаблон проектирования *Abstract superclass*, *interface* и *delegator* состоят из относительно малого количества элементов, это приводит к высокой мере выраженности таких шаблонов в большом количестве проектов.

Таблица 4.2

Результаты экспериментов по вычислению меры структурного подобия программных продуктов по второй (2) метрике

Project 1 / Project 2	*android-mvp*	*cordova-vk*	*Cvk*	*droidfm*	*vk-small-api*	*vkontakteapi*	*vk_test*
android-mvp	-	0.96	0.96	0.98	0.78	0.96	0.78
cordova-vk	0.96	-	1	0.94	0.85	1	0.83
cvk	0.96	1	-	0.94	0.85	1	0.83
droidfm	0.98	0.94	0.94	-	0.78	0.94	0.78
vk-small-api	0.78	0.85	0.85	0.78	-	0.85	0.96
vkontakteapi	0.96	1	1	0.94	0.85	-	0.83
vk_test	0.79	0.83	0.83	0.78	0.95	0.83	-

Таблица 4.3

Результаты экспериментов по вычислению меры структурного подобия программных продуктов по третьей (3) метрике

Project 1 / Project 2	*android-mvp*	*cordova-vk*	*Cvk*	*droidfm*	*vk-small-api*	*vkontakteapi*	*vk_test*
android-mvp	-	0.96	0.96	0.96	064	0.96	0.77
cordova-vk	0.96	-	0.99	0.93	0.67	0.99	0.80
cvk	0.97	0.99	-	0.93	0.67	0.99	0.80
droidfm	0.97	0.93	0.93	-	0.61	0.93	0.74
vk-small-api	0.64	0.67	0.68	0.61	-	0.67	0.87
vkontakteapi	0.97	0.99	0.99	0.93	0.67	-	0.80
vk_test	0.77	0.80	0.80	0.74	0.87	0.80	-

Оценки мера схожести для проектов по второй и третьей метрикам представлены в Таблицах 4.2, 4.3. Результаты по второй и третьей метрикам весьма высоки, что можно объяснить двумя особенностями данного эксперимента.

Шаблоны проектирования с мерой выраженности менее 0.3 хотя бы в одном из сравниваемых проектов исключались. В рамках данного эксперимента предполагается, что шаблон проектирования, выраженный с мерой выраженности менее 0.3, не обнаружен в проекте. Если учитывать шаблоны проектирования с малой мерой выраженности, то при увеличении числа шаблонов будет существенно уменьшаться значение меры схожести любых проектов.

Проекты изначально выбирались, по ключевым словам, из одной предметной области, чтобы увеличить вероятность нахождения проектов со схожей структурой. Предполагается, что разработанная система будет использоваться в рамках крупного предприятия, специализирующегося на конкретной предметной области.

В данном эксперименте все проекты были загружены из открытого репозитория *Github*. Все проекты были отобраны по следующим ключевым словам: “*public API*”, “*social network*”, “*vkontakte*” позволяющим определить принадлежность проектов к предметной области социальной сети *vkontakte*.

Для осуществления анализа временных затрат на процесс проектирования и разработки ПАК проведен ряд экспериментов. Учитывалось общее время разработки проекта, а также временные затраты на процессы анализа существующих решений, проектирования, разработки, тестирования и отладки.

4.4. Вычислительные эксперименты по использованию подсистемы ПИП

В результате проведения первого эксперимента необходимо было проверить гипотезу о том, что разработанная система СИПР, позволит существенно сократить время, затрачиваемое на анализ уже реализованных проектов.

Эксперимент заключался в сравнении времени, затрачиваемого на анализ проектов, с помощью только ключевых слов и с помощью применения системы СИПР к результатам поиска по ключевым словам.

Рассматриваемые проекты были загружены из открытого репозитория *github.com*. Соответствие проектов, отобранных для анализа с помощью традиционного и предлагаемого подходов устанавливалось с помощью экспертов. Задача отбора и анализа проектов на стадии проектирования нового ПАК является нетривиальной и творческой, как правило оценки качества решения подобных задач даются экспертами. Экспертами выступали специалисты: разработчики, проектировщики и менеджеры проектов, входящие в состав научной группы кафедры «Информационные системы» УлГТУ и специалисты предприятий: ФНПЦ АО «НПО «Марс» и АО «Авиастар-СП». Результаты первого эксперимента приведены на рисунке 4.5.

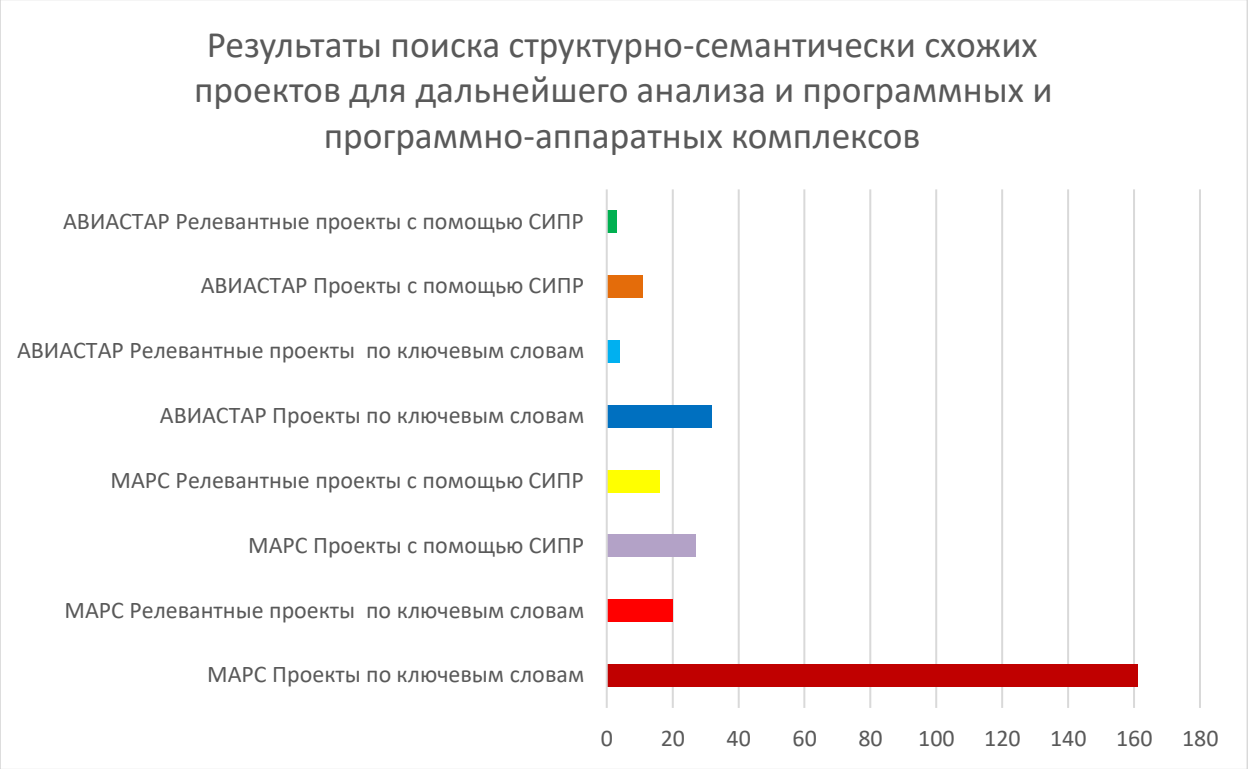

Рис. 4.5. Результаты эксперимента по использованию подсистемы ПИП.

Исходя из результатов, продемонстрированных в ходе проведения первого эксперимента, можно сделать вывод о существенном сокращении объема работы и, как следствие, временных затрат на этапах анализа существующих решений и проектирования нового программного или программно-аппаратного комплекса.

Для проекта разрабатываемого совместного с ФНПЦ АО «НПО «Марс» количество проектов сократилось 5,96 раза, а для проекта разрабатываемого совместного с АО «Авиастар-СП» 2,9 раза.

Качество работы системы определяется количеством релевантных проектов, входящих в выборку.

Для проекта разрабатываемого совместного с ФНПЦ АО «НПО «Марс» качество выборки, по ключевым словам, составило 12,4%, а с помощью системы СИПР - 59,2%, что привело к повышению точности в 4,77 раза.

Для проекта разрабатываемого совместного с АО «Авиастар-СП» качество выборки, по ключевым словам, составило 12,5%, а с помощью системы СИПР - 27%, что привело к повышению точности в 2,18 раза.

Результаты первого эксперимента демонстрируют преимущество во временных затратах при использовании разработанной системы. Потери релевантных проектов составили 20% в первом случае и 25% во втором случае, при сокращении объема выборки в среднем 4,5 раза.

Данный результат характеризует использование системы положительно. Подсистему поддержки интеллектуального проектирования (ПИП) системы СИПР можно рекомендовать применять в случае большого размера выборки, с относительно малым количеством релевантных проектов.

4.5. Вычислительные эксперименты по прогнозированию временных рядов

4.5.1 Прогнозирование развития проекта по НИР разработки Автоматизированной системы балансировки мощностей системы управления системой ПАД

Результатом применения подсистемы ПАД стало управление проектом в рамках НИР разработки Автоматизированной системы балансировки мощностей системы управления АО «АВИАСТАР-СП».

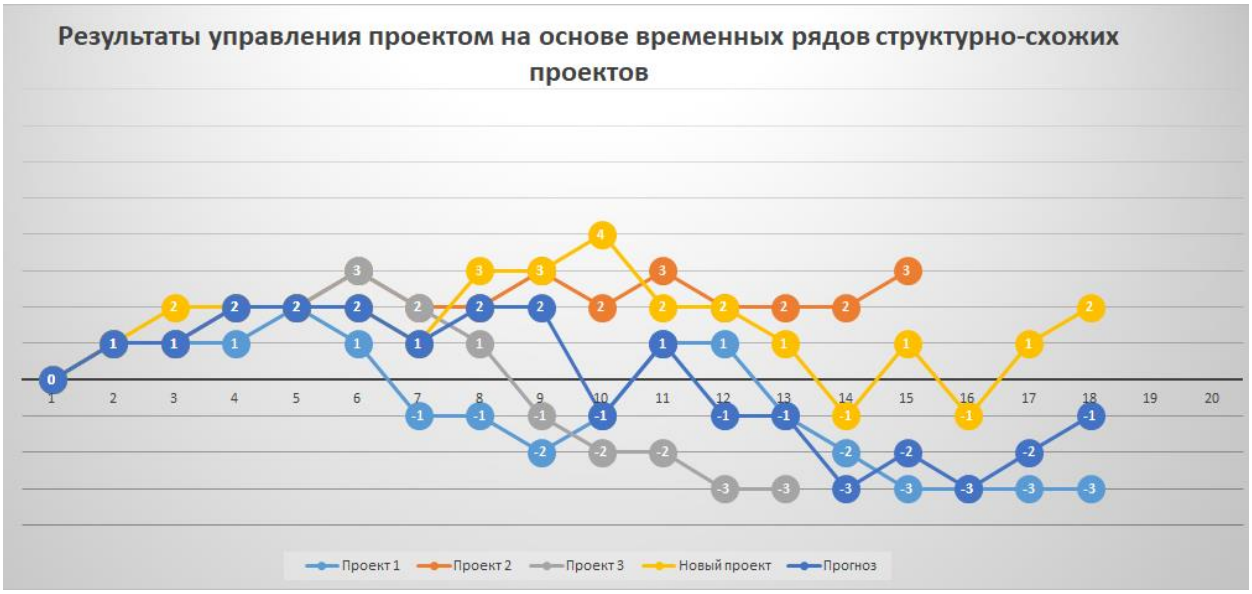

Рис. 4.6. Результаты управления проектом на основе временных рядов структурно-схожих проектов.

На рисунке 4.6 изображены временные ряды состояний структурно-семантически схожих программных проектов, проекта НИР и прогноз, полученный от подсистемы ПАД.

Из результатов эксперимента видно, что несмотря на ограниченность временных рядов структурно-семантически схожих проектов удалось скорректировать развитие НИР относительно прогнозных значений.

Для использования четких моделей прогнозирования использовалось приравнивание состояний проекта к целым числам от -3 до 4, где Fail – это -3, Start – 0, a Unexpected fast – 4.

В качестве алгоритма агрегирования для построения коллектива моделей использовался агрегирующий алгоритм Вовка.

4.5.2 Проверка результативности разработанных моделей прогнозирования временных рядов на наборе NN3

В соответствии с техническим заданием хозяйственного договора (Приложения 1, 3) временные ряды хранятся в файлах *.csv, где каждая строка формата «ДД.ММ.ГГГГ; значение» или «Номер; значение» в случае одинаковых временных интервалов между точками ВР. Количество точек ВР определяется количеством строк в файле. Для тестирования программной системы использовались временные ряды соревнования NN3. Пример ряда №89 приведен в таблице 4.4.

Временные ряды, извлеченные из систем контроля версий, для проекта, описанного в пункте 4.5.2, представляются в обезличенном виде. Семантическая интерпретация результатов прогнозирования происходит лишь в модуле формирования рекомендаций. Любой временной ряд представляется набором точек входных данных для моделей прогнозирования. Подобный подход позволяет сохранить гибкость на уровне моделей прогнозирования и отделить задачу расчета прогноза от задачи его интерпретации.

Таблица 4.4.

Пример временного ряда во входном файле

Номер точки	Значение	Номер точки	Значение	Номер точки	Значение
1	2992	27	5222	53	4448
2	3256	28	4888	54	4526
3	4290	29	4878	55	4248
4	4190	30	4830	56	4142
5	4080	31	4614	57	4502
6	4162	32	4800	58	4460
7	3688	33	4780	59	3930
8	3846	34	4690	60	3670
9	3838	35	4360	61	3192
10	4008	36	4710	62	3606
11	4254	37	4576	63	3926
12	3944	38	4634	64	3760
13	3750	39	3832	65	3950
14	3596	40	4360	66	3830
15	4238	41	3616	67	3920
16	4308	42	3224	68	4002
17	4350	43	3934	69	4178
18	4386	44	3610	70	3938
19	4086	45	4122	71	3820
20	3878	46	4498	72	3194
21	4294	47	4212	73	3682
22	4334	48	4520	74	3920
23	3944	49	4078	75	4442
24	4368	50	4068	76	4168
25	4578	51	5298	77	4202
26	4242	52	4544	78	4258

Значения прогнозов по каждой модели сохраняются в отдельные файлы, количество строк в которых соответствует количеству точек прогноза. Результаты прогноза сохраняются в отведенном для них каталоге, что позволяет использовать их повторно при агрегации прогнозов. Пример прогноза на 30 точек приведен в таблице 4.5.

Таблица 4.5.

Пример прогнозных значений для метода экспоненциального сглаживания с мультипликативным трендом и аддитивной сезонностью

Номер точки	Значение	Номер точки	Значение	Номер точки	Значение
1	3934	7	4300	13	4308
2	3922	8	4522	14	4418
3	4212	9	4810	15	4720
4	4470	10	4742	16	4548
5	4498	11	4780	17	4058
6	4042	12	4618	18	4162

В ходе реализации алгоритма агрегации коллектива методов было реализовано несколько особенностей:

1 Для корректной работы данного алгоритма необходимо нормализовать данные временного ряда. Экспериментально был подобран отрезок нормализации от 0 до 1. В случае большой разницы между точками (порядка нескольких тысяч) происходило вырождение пересчета весов;

2 С учетом представленных моделей для агрегации производится сдвиг временного ряда из области отрицательных значений;

3 Кроме того, были реализованы методы подготовки данных перед прогнозированием, а именно логарифмирование ряда и Box-Cox [47].

Таблица 4.6.

Результаты прогнозирования

Лучший результат по моделям	Веса по модиф. ИК Акаике	Веса по Баиесовскому ИК	Веса по ИК Акаике	Нечеткие веса	Вовк
27.3847	**27.2887**	**27.2887**	**27.2887**	28.3996	29.8256
23.3531	27.3576	27.3576	27.3576	41.801	24.0437
16.6025	16.6128	16.6128	16.6128	39.0262	19.977
23.4329	27.4952	27.4952	27.4952	24.1636	24.1084
6.48463	6.49801	6.49801	6.49801	8.66947	9.99404
8.39089	**7.64793**	**7.64793**	**7.64793**	8.52974	10.6689
5.58562	5.57776	5.57776	5.57776	8.55234	7.4812
5.494	**5.3245**	**5.3245**	**5.3245**	11.9307	6.54996

В таблице 4.6. приведены результаты тестирования реализованных агрегационных алгоритмов и лучший результат по отдельной модели. Для тестирования использовались ряды [49].

4.6 Эффективность внедрения разработанной программной системы СИПР в проектных организациях и на промышленных предприятиях

4.6.1. Автоматизированное проектирование ФНПЦ АО «НПО «Марс»

Ульяновским государственным техническим университетом по заказу ФНПЦ АО «НПО «Марс» выполнена НИР «Система интеллектуального поиска и анализа в Интернет-СМИ и социальных сетях» [100]. В результате НИР разработан интеллектуальный программный инструмент семантического анализа социальных сетей, реализующий новые алгоритмы гибридизации онтологического анализа с методами обработки естественного языка для извлечения семантической составляющей слабоструктурированных и неструктурированных текстовых ресурсов.

Интеллектуальный инструментарий для *Opinion Mining* социальных медиа предполагает создание новых подходов к гибридизации онтологического анализа и методов инженерии знаний с методами анализа текстов на естественном языке в задачах извлечения семантической и эмоциональной составляющей слабоструктурированных и неструктурированных ресурсов. Данные подходы позволят повысить эффективность анализа содержимого социальных медиа с учетом специфики представления данных и нечеткости конструкций естественного языка [34].

Методы автоматизированного интеллектуального анализа текстовой информации, позволяющих за короткое время обработать большие объемы данных и понять смысл (*Text Mining*) и эмоциональную окраску (*Opinion Mining*) пользовательских сообщений и публикаций в настоящее время часто используются, а библиотеки методов широко представлены в репозиториях открытого кода. Поиск аналогичных проектов открытого кода и их повторное использование позволит сократить сроки разработки автоматизированной системы анализа.

Для поиска на начальном этапе отбора проектов были использованы основные термины, извлекаемые экспертом из технического задания, описывающие функциональные подсистемы автоматизированной системы:

1. Термины для подсистемы импорта данных из социальных медиа и социальных сетей: *vk, vkontakte, Facebook, API, Public API,* Одноклассники, *Twitter, Youtube*. Данные термины были отобраны, так как взаимодействие с социальными сетями и интернет-СМИ необходимо было реализовать через публичные программные интерфейсы (*Public API*). На основе результатов выборки проектов было принято решение реализовать загрузчик данных из HTML-страниц Интернет-СМИ на основе правил. Проекты, отобранные по принципу архитектурного подобия, в основном осуществляли обработку интернет-СМИ с помощью собственных правил, состоящих из набора *CSS*-селекторов;

2. Подсистема хранения данных обеспечивает представление информации, извлеченной из социальных медиа, в унифицированной структуре. Данные хранятся в разрезе пользователей, коллекций, источников данных, версий т. д. В качестве систем управления базами данных (СУБД) используются:

- *Elasticsearch* для индексации и поиска данных [54];
- *MongoDB* для хранения данных в формате JSON [81];
- Neo4j для хранения графов социального взаимодействия и онтологии [68];

Конвертер данных обеспечивает преобразование импортированных из социальных медиа данных во внутреннее преставление ПС. Построитель социального графа формирует граф социального взаимодействия на основе данных об отношениях пользователей и сообществ социальных медиа. Транслятор OWL/RDF-онтологии в граф позволяет транслировать онтологию в графовую базу знаний [100];

3. Подсистема семантического анализа данных обеспечивает

выполнение предобработки текстовых ресурсов, проведение статистического и лингвистического анализа текстовых ресурсов;

4. Подсистема поиска данных обеспечивает возможность поиска объектов, имеющих отношение к возникшей ситуативной задаче, формируемой в виде множества ключевых слов.

Для поиска аналогичных проектов необходимо выделить основные сущности модели данных, загружаемых из различных социальных медиа:

- Сущность *MassMedia* – коллекция, элементы которой соответствуют определенным социальным медиа, загрузку которых поддерживает подсистема импорта данных: ВКонтакте, *Facebook, Twitter* и т.д;
- Сущность *Person* – коллекция, элементы которой содержат список пользователей, извлеченных из социальных медиа. Сущность *Person* имеет набор атрибутов, соответствующий атрибутам, часто используемым в социальных сетях: фамилия, имя, отчество, дата рождения, увлечения, сведения об образовании и т.д.;
- Сущность *Group* – коллекция, элементы которой содержат информацию о сообществах, извлеченных из социальных медиа. Сущность *Group* имеет набор атрибутов, соответствующий атрибутам, часто используемым в социальных сетях: название группы, описание группы, возрастные ограничения, дата создания и т.д.;
- Сущность *Post* – коллекция, элементы которой содержат информацию о записях в социальных медиа. Сущность *Post* имеет следующий набор атрибутов: автор, заголовок, содержимое, дата создания, вложения и т.д.;
- Сущность *Comment* – коллекция, элементы которой содержат информацию о комментариях в социальных медиа. Сущность *Comment* имеет следующий набор атрибутов: автор, заголовок, содержимое, дата создания, вложения и т.д.;
- Сущность *Attachment* – коллекция, элементы которой содержат информацию о вложениях записей и комментариев в социальных медиа. Сущность *Attachment* имеет несколько типов и позволяет хранить

следующие виды вложений: фотографии, фотоальбомы, аудиозаписи, видеозаписи, гиперссылки, документы (файлы), опросы и т.д.

В результате использования в ходе выполнения НИР разработанного средства определения схожести проектов была сокращена трудоемкость разработки интеллектуального программного инструмента семантического анализа социальных сетей. Разработка инструментария выполнена в срок и передана Заказчику (Акт о внедрении – Приложение 4)

4.6.2. Автоматизированное проектирование ПАК «Оптимизация ресурсов на основе производственно-технологического моделирования в условиях мультипродуктовой производственной программы» АО «Авиастар-СП»

ПАК «Оптимизация ресурсов на основе производственно-технологического моделирования в условиях мультипродуктовой производственной программы» разработан в рамках выполнения проекта №2.4760.2017/8.9 Министерства образования и науки России в партнерстве с АО «Авиастар-СП».

Систематизация и унификации расчета и управления мощностями серийных заводов, оптимизация ресурсов на основе производственно-технологического моделирования является составной частью технологической подготовки производства и нуждается в автоматизации для эффективной реализации. Задачи ПАК «Оптимизация ресурсов на основе производственно-технологического моделирования в условиях мультипродуктовой производственной программы» следующие:

- Формирование объективной информации о мощностях серийных заводов корпорации;
- Составление плановых и отчетных балансов производственных мощностей серийных заводов корпорации;
- Составление рекомендованных мероприятий по устранению производственных диспропорций корпорации;

- Разработка требований по развитию кадрового потенциала серийных заводов под заданные производственные задачи;
- Обоснование межзаводской кооперации;
- Выявление потребности в развитии сети внешних поставщиков сборочных единиц и комплектов деталей;

4.6.2.1 Автоматизация технологической подготовки производства. Задача оптимизации ресурсов методом балансировки мощностей.

Под производственной мощностью понимается способность закрепленного за предприятием технологического и сборочно-технологического оборудования, производственных площадей, а также основных производственных рабочих в цехах основного производства обеспечить максимальный выпуск продукции за заданный интервал времени в соответствии с режимом работы.

Баланс производственных мощностей – система показателей, характеризующих наличие, движение и уровень использования производственных мощностей, определяющих выпуск продукции. Балансировка производственных мощностей является комплексной задачей и должна решаться на уровне каждого серийного завода в рамках корпорации. Под серийным заводом понимается изготовитель условных единиц отраслевой техники либо компонентов отраслевой техники (агрегатов, отсеков, сборочных единиц и т.д.), входящий в состав корпорации. Балансировка производственных мощностей, выполняемая в рамках серийного завода, состоит из результатов решения задачи балансировки по цехам с учетом плана производственной программы. Результатом балансировки производственных мощностей является формирование технологического паспорта предприятия. Технологический паспорт – это документ, который формируется по каждому серийному заводу и отражает техническое оснащение предприятия и его производственные характеристики на момент составления.

Производственная программа – планируемый объем производства продукции установленной номенклатуры, измеряется в условных единицах продукции и человеко-часах. Товарной программой называется директивный план поставок продукции по отраслевой технике в натуральном и стоимостном выражении.

4.6.2.2 Автоматизация технологической подготовки производства окончательной сборки и агрегатно-сборочного производства.

Автоматизация технологической подготовки производства была разработана для производства окончательной сборки и агрегатно-сборочного производства.

Производство окончательной сборки самолетов проектируется на основе принципа поточной линии сборки. Автоматизация реализована как АСУ поточной линии сборки (ПТС) в соответствие с рабочей документацией «Реконструкция и техническое перевооружение для создания поточной линии сборки самолетов для производства тяжелого транспортного самолета Ил-76МД- 90А на АО «Авиастар-СП».

Автоматизированная система управления поточной линией сборки (АСУ ПЛС) является составной частью интегрированной автоматизированной системы (ИАС) информационной поддержки жизненного цикла воздушного судна (ВС) гражданской и транспортной авиации на основе электронного определения изделия на основе действующих в АО «Авиастар СП» подсистем конструкторско-технологической подготовки производства (КТПП) ИАС и развития подсистем Управления Комплектацией Сборочного Производства и Системы Технологического Моделирования Процессов «ТЕМП2».

В общем случае, ПАК предназначен для автоматизации деятельности структурных подразделений, реализующих производственные процессы в соответствии с Таблицей 4.7:

1. Служба главного конструктора;
2. Служба главного технолога и главных специалистов;

3. Дирекция по производству и цеха сборочного производства;
4. Цеха производства технологического оснащения;
5. Подразделения коммерческих служб, занимающиеся закупками и размещением заказов на проектирование;
6. Подразделения изготовления средств технологического оснащения (СТО);
7. Дирекция по качеству и сертификации.

Таблица 4.7

Автоматизированные процессы

Автоматизируемые процессы	**Структурные подразделения**					
	1	**2**	**3**	**4**	**5**	**6**
Конструкторская подготовка производства	ДА	ДА	ДА	ДА	ДА	ДА
Централизованная технологическая подготовка производства		ДА	ДА	ДА		ДА
Проектирование, изготовление и эксплуатация СТО		ДА	ДА	ДА	ДА	ДА
Технологическая подготовка в цехах сборочного производства		ДА	ДА	ДА		ДА
Производственно-технологическое моделирование	ДА	ДА	ДА	ДА	ДА	ДА
Управление комплектацией сборочного производства ВС			ДА	ДА	ДА	ДА
Планово-предупредительное обеспечение рабочих мест	ДА	ДА	ДА	ДА	ДА	ДА
Мониторинг и анализ состояния конструкторского, технологического, производственного процессов	ДА	ДА	ДА	ДА	ДА	ДА
Взаимодействие с предприятиями-кооперантами	ДА	ДА	ДА	ДА	ДА	ДА

Объектом автоматизации являются основные производственные процессы, связанные с функционированием рабочих станций поточной линии сборки изд. ИЛ-76-МД.

Отдельные рабочие станции, как объекты в составе линии сборки, могут иметь встроенные средства автоматизации в виде специальных управляющих контроллеров или систем ЧПУ. Описание встроенных средств автоматизации, систем программного управления рабочими станциями, программных средств входных данных, необходимых для разработки управляющих программ и лицензии на необходимое количество ПО входят в состав документации конкретных рабочих станций.

Разрабатываемый ПАК в соответствии с описанием рабочих станций должен обеспечивать подготовку необходимых данных на вход управляющих контроллеров или систем ЧПУ автоматизированных рабочих станций, а также обеспечить прием и хранение соответствующих выходных данных.

Основными функциями АСУ ПЛС являются:

- планирование обеспечения ресурсами рабочих станций;
- контроль за расходованием ресурсов;
- графическая индикация и контроль подачи комплектующих для сборки на рабочую станцию;
- генерация необходимой отчетности.

Процесс сборки и испытаний самолета Ил-76МД-90А на предприятии осуществляется по следующей технологической схеме, в указанном порядке по станциям (таблица 4.8).

Таблица 4.8.

Перечень станций и краткое определение работ на них.

Обозначение станции	Наименование станции	Описание основных операций, выполняемых на станции
ПС.10	Стыковка фюзеляжа	Стыковка секций Ф1, Ф2, ФЗ в фюзеляж, монтаж шасси, проверка герметичности и опрессовка фюзеляжа
ПС.20	Станция монтажа гидросистем	Монтаж трубопроводов гидросистемы
ПС.30	Станция электромонтажей	Монтаж электрожгутов
ПС.40	Станция стыковки крыла	Стыковка крыла, испытание герметичности бака, монтаж десантного, ПНК, бытового оборудования
ПС.50	Станция комплектации шасси	Комплектация и подготовка к установке шасси
ПС.60	Станция стыковки киля со стабилизатором	Разделка киля, стыковка киля со стабилизатором
ПС.70	Станция подготовки двигателя	Подготовка двигателей к навеске на самолет
ОС.10	Станция навески двигателей	Стыковка оперения, навеска двигателей, монтаж топливной системы, системы управления, ПОС, СКВ, ППЗ, НГ, электрооборудование в крыле
ОС.20	Станция монтажа и отработки систем	Отработка системы электроснабжения, промывка и отработка гидросистемы, отработка высотного оборудования, кислородной системы, системы управления самолетом, ППЗ, НГ, окончательный монтаж ТЗИ
ОС.30	Станция финальной отработки	Испытания десантного, транспортного, аварийно-спасательного оборудования, комплектация изделия, отработка на КИС, предъявления, сдача на ЛИС
ЛГ.10	Логистическая станция	Логистика и другое вспомогательное оборудование для прочих станций
АП.10	Мероприятия по агрегатно-сборочному производству	В эту условную станцию включены мероприятия по модернизации оснастки АСП для снижения технических рисков по стыковке КЧК

Каждая рабочая станция, являющаяся звеном в единой цепи сборки агрегатов, на вход получает массив данных, на основании которых специалист принимает решение о возможности запуска процесса сборки на конкретной рабочей станции. Схематично данный процесс представлен на рисунке 4.7.

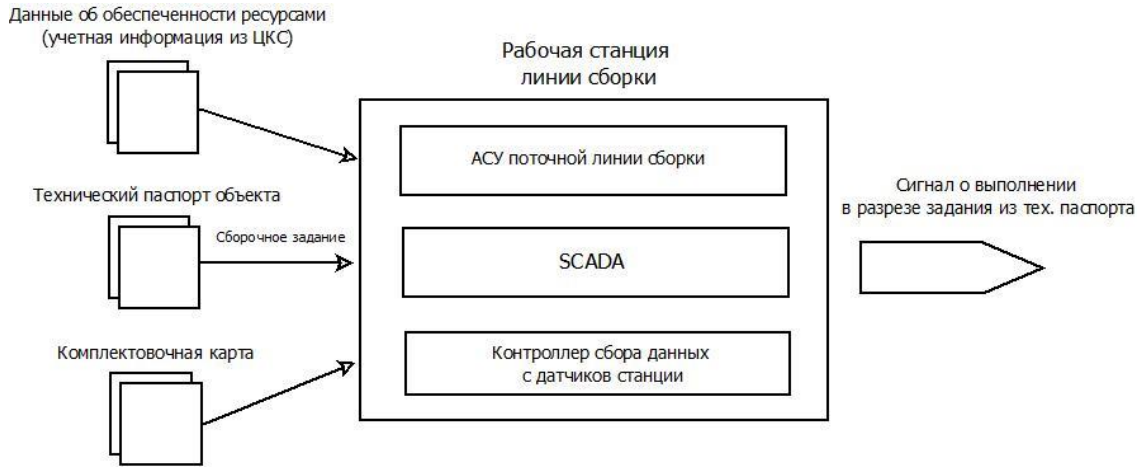

Рис. 4.7 Схема потоков данных на АСУ ПЛС конкретной рабочей станции.

Как видно из данной схемы, на вход АСУ ПЛС рабочей станции помимо комплектующих и агрегата, подается информация об обеспеченности необходимыми ресурсами в соответствии с комплектовочной картой, а также непосредственное сборочное задание.

На самом нижнем уровне системы управления технологическим процессом на каждой конкретной станции находится программируемый контроллер сбора данных с датчиков станции в виде сигналов. Далее полученная контроллером информация обрабатывается системой SCADA (Supervisory Control And Data Acquisition). Основные функции SCADA следующие.

- Логическое управление;
- Обмен данными с промышленными контроллерами и платами ввода-вывода в онлайн-режиме;
- Визуальное представление полученной и обрабатываемой информации;

- Работа с БД, содержащей технологическую информацию;
- Аварийная сигнализация и управление тревожными сообщениями;
- Формирование отчетов относительно хода технологического процесса (ТП);
- Обеспечение взаимосвязи с внешними программными системами (электронные таблицы, текстовые процессоры, СУБД и т. д.).

Верхний уровень системы занимает АСУ ПЛС, целью которой является окончательная обработка и формализация поступивших с нижних уровней данных, управление конкретными операциями и процессами, выполняемыми данной станцией, а также обеспечение человеко-машинного интерфейса (HMI).

Система должна иметь структуру, представленную на рисунке 4.8 или аналогичную ей по функциональным возможностям.

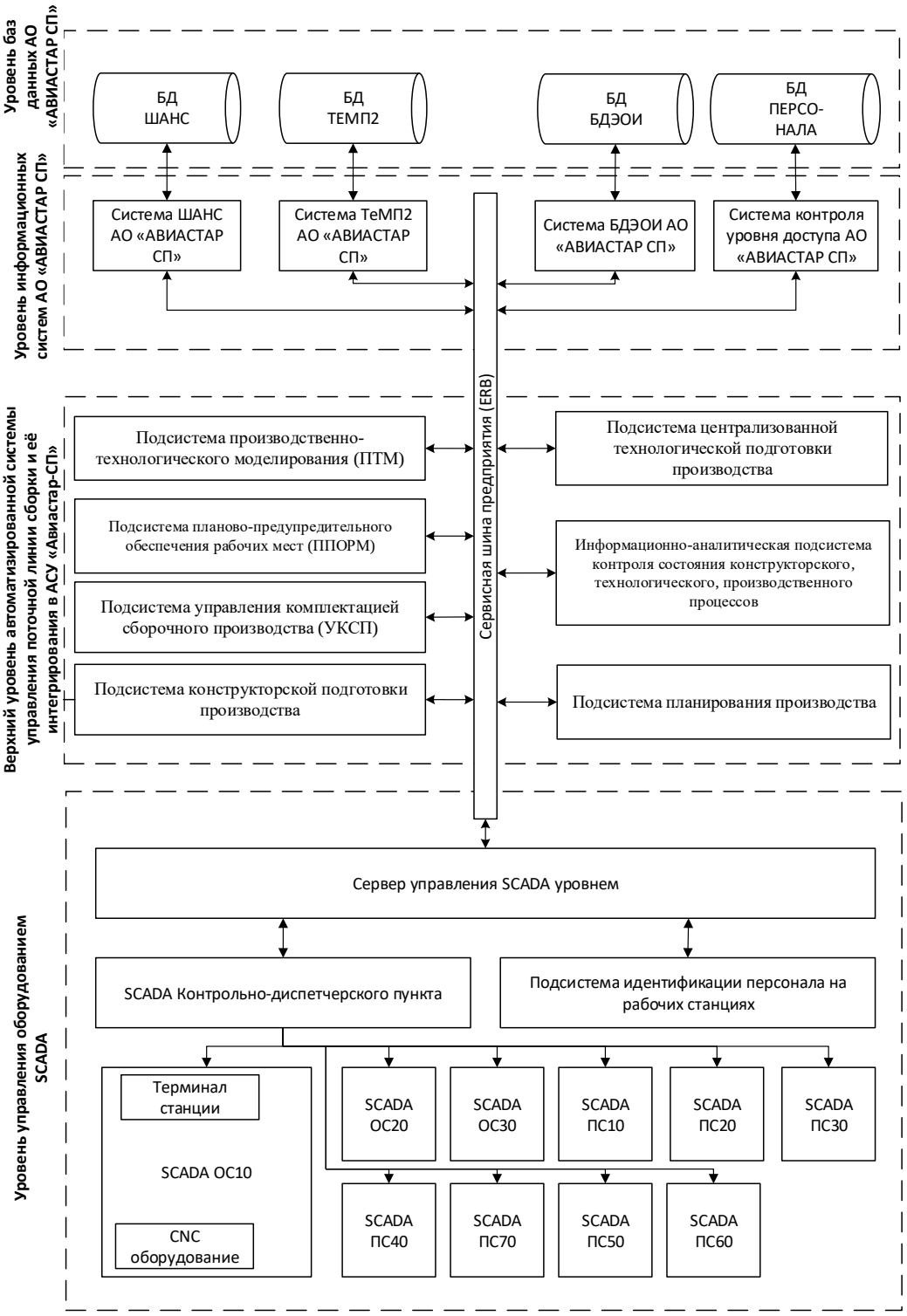

Рис. 4.8 Общая структура АСУ ПЛС.

4.6.2.3 Порядок формирования консолидированного баланса производственных мощностей корпорации

Блок-схема разработки консолидированного баланса мощностей корпорации приведена на рисунке 4.9.

Рис. 4.9 Блок-схема разработки консолидированного баланса мощностей.

Основные требования к решению задачи разработки консолидированного баланса мощностей следующие.

1. Баланс мощностей серийного завода оформляется в виде разделов Технологического паспорта серийного завода;

2. Консолидированный баланс производственных мощностей корпорации формируется на основе балансов производственных мощностей серийных заводов;
3. Для расчета баланса мощностей на серийные заводы поставляются директивные показатели, производственная программа, товарная программа и схема кооперации;
4. На основании директивных показателей и схем кооперации серийные заводы рассчитывают производственные программы и балансы производственных мощностей в соответствии с внутренними организационно-распорядительными и нормативными документами, которые согласовываются с корпорацией для унификации расчетов;

Расчет баланса мощностей серийного завода включает:

1) заполнение технологического паспорта предприятия;
2) расчет мощностей по каждому технологическому переделу, цеху, участку и предприятию в целом;
3) разработку мероприятий по устранению дефицита мощностей;
4) формирование сводной справки о прогнозе выполнения товарной программы;
5) расчет консолидированного баланса мощностей;
6) рассогласование консолидированного баланса мощностей;
7) утверждение консолидированного баланса мощностей.

В технологическом паспорте серийного завода указываются фактические показатели на конец предшествующего года, а также прогнозные по текущему году.

По результатам расчета баланса мощностей по оборудованию предприятия могут быть предложены следующие рекомендации.

Дефицит мощности по площади предполагает следующие действия:

1) переоснащение неиспользуемых площадей под дефицитные виды работ;
2) закупка современного высокопроизводительного оборудования, требующего меньше площади для размещения;

3) покупка дополнительных помещений;
4) аренда дополнительных помещений;
5) строительство дополнительных помещений на территории серийного завода.

Профицит мощности по площади предполагает следующие действия:

1) создание дополнительной продукции или комплектующих;
2) переоснащение части помещений под иные нужды;
3) сдача части помещений в аренду;
4) консервация помещений.

Дефицит мощности по оборудованию предполагает следующие действия:

1) перенос части работ по производственной программе в цех, содержащий подходящее оборудование, но имеющий свободные мощности;
2) переоснащение оборудования (двойного и более назначения) под дефицитные виды работ;
3) закупка современного высокопроизводительного оборудования;
4) переход на двухсменный режим работы.

Профицит мощности по оборудованию предполагает следующие действия:

1) перенос работ с периода, в котором по расчету предполагается дефицит по оборудованию;
2) устранение дефицита мощности по оборудования из других цехов;
3) создание дополнительной продукции или комплектующих.

Дефицит мощности по персоналу предполагает следующие действия:

1) набор дополнительных сотрудников;
2) повышение квалификации сотрудников с целью использования более производительного оборудования;
3) закупка современного высокопроизводительного оборудования.

Профицит мощности по персоналу предполагает следующие действия:

1) создание дополнительной продукции или комплектующих;
2) переход на режим работы в две смены;

3) сокращение штата сотрудников.

4.6.2.4 Архитектура системы оптимизации ресурсов на основе производственно-технологического моделирования в условиях мультипродуктовой производственной программы.

Для решения поставленных задач следует предусмотреть два крупных архитектурных блока: блок согласования интерфейсных частей и блок дополнения полученной информации (рисунок 4.10). Для облегчения сопровождения указанной интерфейсной части следует выделить ее либо в качестве независимого модуля, либо в качестве сервисного ПО, т. к. при изменчивости типов данных нагрузка по их согласованию должна полностью ложиться на этот компонент и не затрагивать работу систем предприятия и системы расчета балансировки.

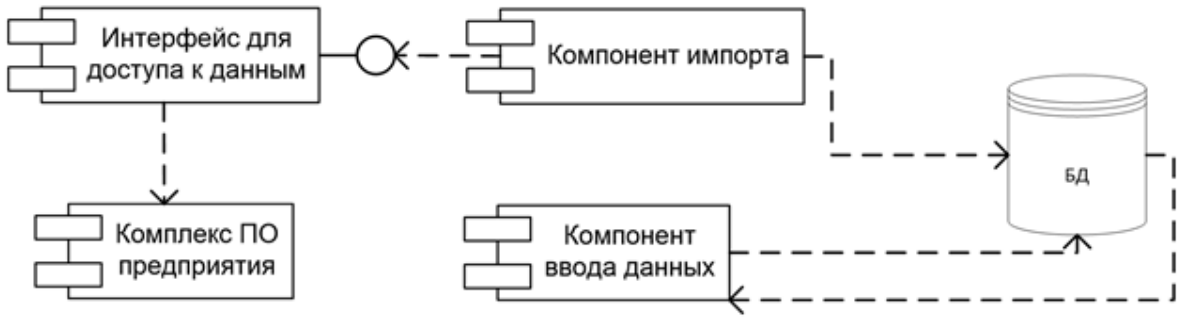

Рис. 4.10 Согласование интерфейсов.

С точки зрения проектируемой системы потребуется наличие собственной БД для хранения дополнительных данных, промежуточных расчетов. Кроме того, отчеты носят периодический характер, а самой трудоемкой операцией выступает ввод и актуализация данных. Для облегчения труда ответственного лица по подготовке отчета предусматривается наличие версионности как в данных, используемых для подготовки отчета, так и самих отчетов.

Задача построения системы балансировки мощностей заключается в возможности проведения предварительного расчета с целью проверки, можно ли выполнить указанный объем работ на текущей конфигурации производства. Эта задача может быть решена, если в систему предварительно

ввести сведения о программе производства, которая в ходе расчета может быть разложена на составляющие операции. Далее следует шаг, на котором рассчитывается дефицит или профицит мощностей оборудования по каждому из видов операций. Т.к. одновременно все производственные мощности не могут быть заняты, возникает возможность частичной параллельной загрузки оборудования для участия в разных производственных процессах или производстве разных частей продукции. Расчет также требует ввода данных о производственной программе, причем его можно вынести в отдельный компонент. Общая структура системы представлена на рисунке 4.11.

Рис. 4.11. Общий вид компонентов системы.

Управление версиями данных и отчетов производится в соответствии со схемой (Рисунок 4.12).

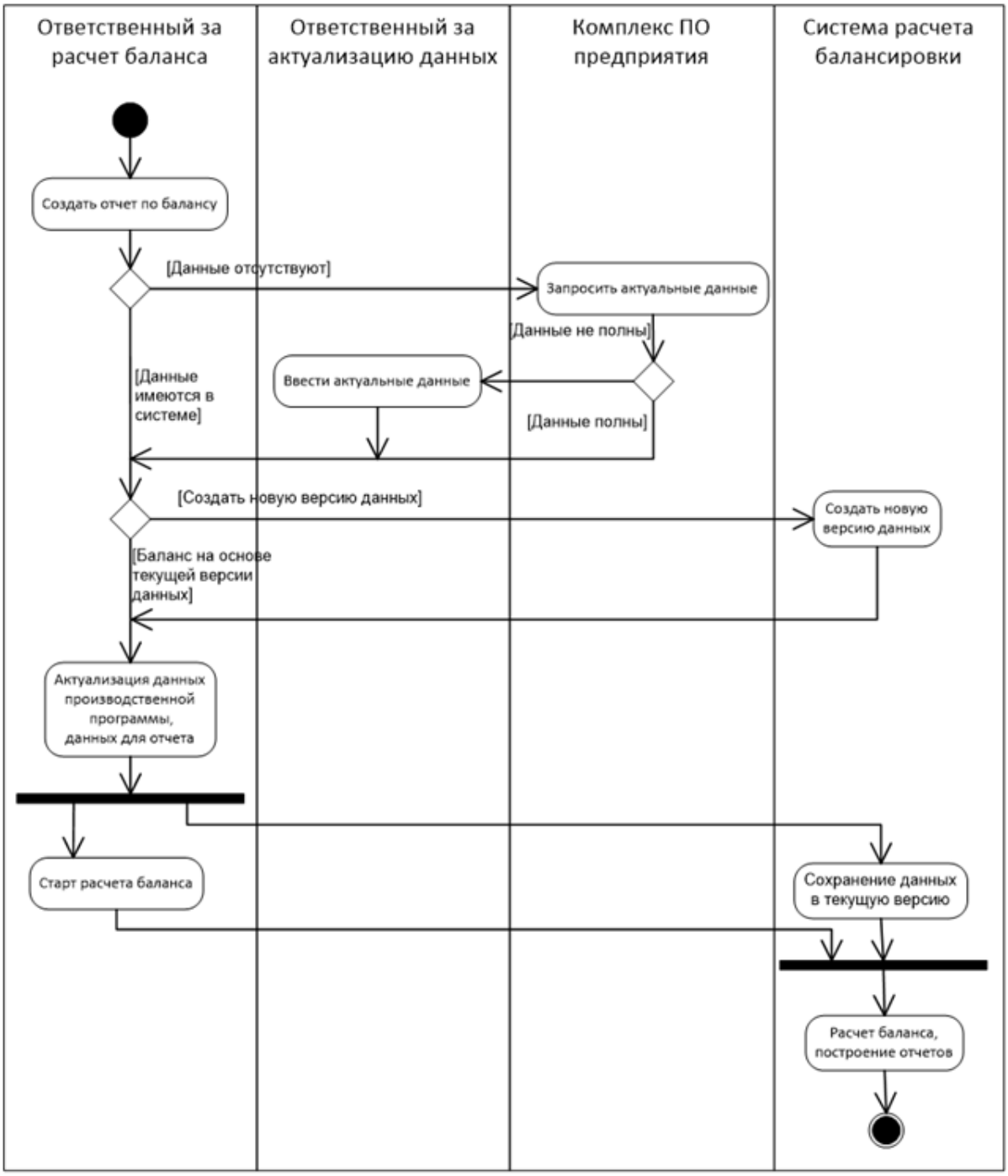

Рис. 4.12. Последовательность выполнения операций при расчете баланса.

Для системы расчета баланса можно выделить следующие основные роли пользователей.

1. Администратор системы. Выполняет импорт данных из комплекса ПО предприятия;

2. Ответственный за составление отчета по балансировке. Создает версии данных и ответов. Актуализирует данные в системе расчета балансировки;

3. Ответственный за актуализацию данных в системе предприятия.

Функции ролей «Администратор» и «Ответственный за составление отчета» могут быть объединены. На рисунке 4.13 представлены возможные сценарии взаимодействия пользователей с системой.

Рис.4.13. Сценарии взаимодействия

4.6.2.1 Модель данных информационной базы системы оптимизации ресурсов

В процессе проектирования информационного обеспечения (ИО) системы оптимизации ресурсов были выделены следующие сущности рассматриваемой проблемной области, которые будут положены в основу модели данных системы оптимизации ресурсов:

1. Сотрудники;

2. Подразделения;
3. Оборудование.

На рисунке 4.14 представлена диаграмма «Сущность-Связь», построенная для модели данных системы оптимизации ресурсов.

Как видно из рисунка 4.14 представленные выше сущности: сотрудники, подразделения, оборудование, являются основными таблицами, связанными различными отношениями с дополнительными таблицами.

Таблица «employee» представляет сущность «Сотрудник» и содержит следующие поля:

- id – первичный ключ таблицы;
- lastname – фамилия;
- firstname – имя;
- patronymic – отчество;
- hiring_date – дата приема на работу;
- data_version_id – версия данных (внешний ключ).

Таблица «position» содержит множество должностей, каждой записи данной таблицы соответствуют следующие поля:

- id – первичный ключ таблицы;
- unit_id – подразделение (внешний ключ);
- name – название;
- data_version_id – версия данных (внешний ключ).

Таблица «employee_position» позволяет реализовать отношение вида «многие-ко-многим» между таблицами «employee» и «position». Фактически данная таблица определяет какую должность занимает сотрудник.

Таблица «unit» представляет сущность «Подразделение» и содержит следующие поля:

- id – первичный ключ таблицы;
- parent_id – идентификатор родительского подразделения в иерархии подразделений;

- unit_type – тип подразделения;
- name – название;
- production_area – производственная площадь;
- additional_area – дополнительная площадь;
- storage_area – складская площадь;
- administrative_area – административная площадь;
- work_time_fund – фонд рабочего времени;
- load_reduction_factor – коэффициент понижения нагрузки;
- data_version_id – версия данных (внешний ключ).

Таблица «tool» представляет сущность «Оборудование» и содержит следующие поля:

- id – первичный ключ таблицы;
- name – название;
- serial_number – инвентарный номер;
- production_date – дата ввода в эксплуатацию;
- unit_id – подразделение (внешний ключ);
- data_version_id – версия данных (внешний ключ).

Таблица «tool_type» содержит множество типов оборудования, каждой записи данной таблицы соответствуют следующие поля:

- id – первичный ключ таблицы;
- name – название;
- data_version_id – версия данных (внешний ключ).

Таблица «tool_tool_type» позволяет реализовать отношение вида «многие-ко-многим» между таблицами «tool» и «tool_type». Фактически данная таблица определяет типы оборудования.

Таблица «work_type» содержит множество типов работ, выполняемых на оборудовании, каждой записи данной таблицы соответствуют следующие поля:

- id – первичный ключ таблицы;
- name – название;
- data_version_id – версия данных (внешний ключ).

Таблица «tool_work_type» позволяет реализовать отношение вида «многие-ко-многим» между таблицами «tool» и «work_type». Фактически данная таблица определяет тип работ, выполняемых на оборудовании.

Таблица «data_version» содержит множество версий данных, каждой записи данной таблицы соответствуют следующие поля:

- id – первичный ключ таблицы;
- data_date – версия.

Таблица «data_version» используется для представления информации, содержащейся в ИО системы оптимизации ресурсов, в различных временных разрезах, что позволяет получить актуальные данные на определенную дату.

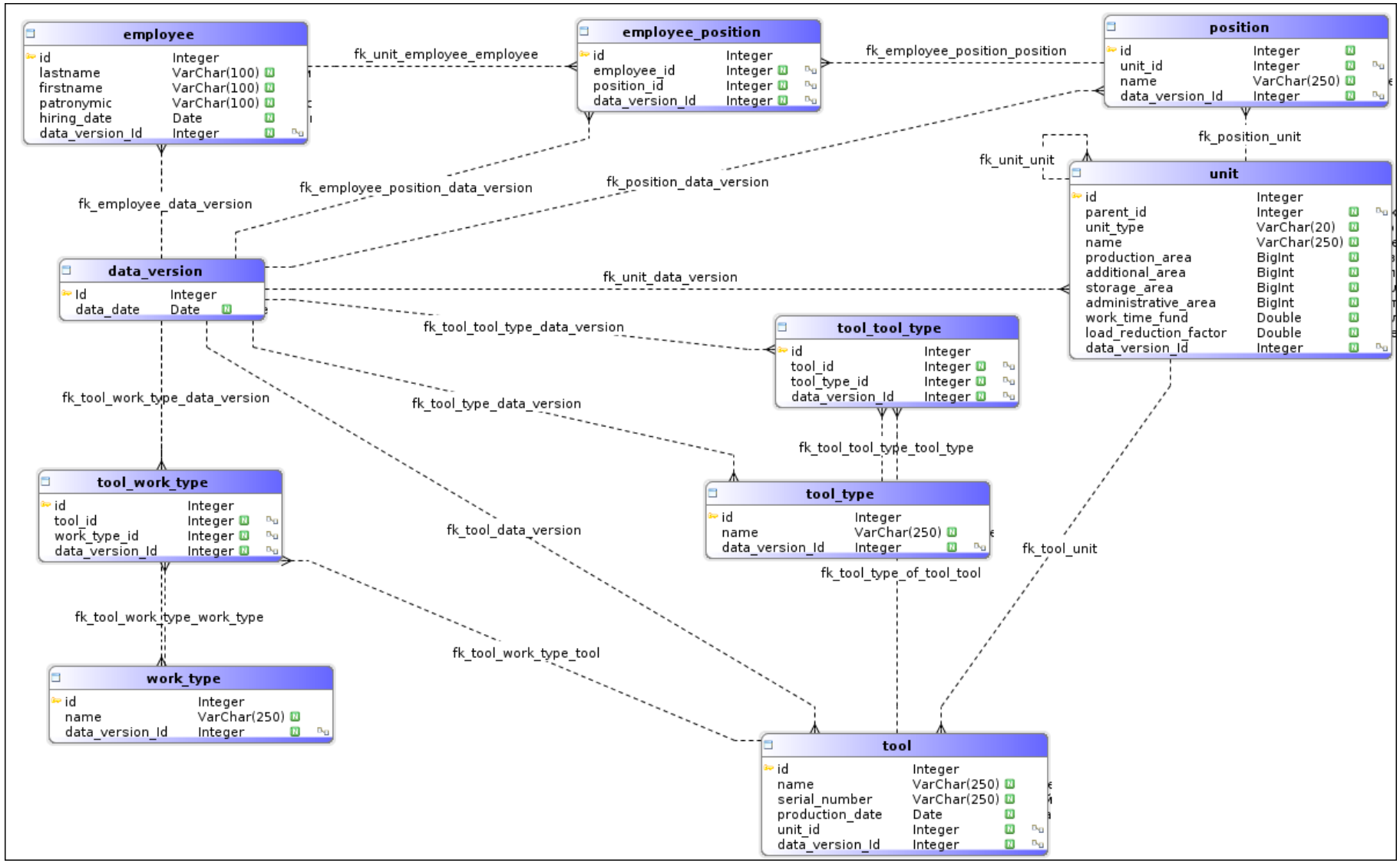

Рис. 4.14. Диаграмма «Сущность-Связь» модели данных системы оптимизации ресурсов

Компоненты данного ПАК имеют высокую связность на уровне предметной области, в следствии чего крайне сложно разделить данный проект на совершенно обособленные модули. Задача разделения предметной области осложняется частыми изменениями нормативных документов, форм отчетности и документов, регламентирующих технический процесс. Помимо проблем, связанных с частыми изменениями технических документов, существует проблема соответствия нормативных документов и реальной ситуации. Не всегда директивы, рекомендуемые в нормативных документах, могут быть полностью исполнены на практике, за счет большого числа уникальных аспектов реального производства, таких как, особенности помещений, логистических маршрутов, распределения квалификаций среди персонала и т.д.

В соответствии с проблемами, описанными выше, приведенный фрагмент описания программно-аппаратного комплекса является достаточно сложным, чтобы для его реализации были использованы только формализованные методы проектирования, в частности, нотация UML.

В ходе проектирования и реализации использовалась система контроля версий. Мониторинг хода развития реализации программного обеспечения выполнялся с помощью, разработанной в данной диссертационной работе подсистемы поддержки интеллектуального проектирования и подсистемы анализа данных.

Динамику реализации проекта руководитель проекта мог эффективно отследить на основе следующих временных рядов показателей:

- количество выявленных ошибок на данном этапе тестирования;
- количество commit за последний этап реализации (спринт);
- количество release за последний этап реализации (спринт);
- количество сущностей онтологии проблемной области, встроенных в диаграммы классов проекта за последний этап реализации (спринт);
- количество сущностей онтологии проблемной области, встроенных в исходный код программы за последний этап реализации (спринт);

- количество шаблонов проектирования, реализованных в версии программы за последний этап реализации (спринт).

Система интеллектуального проектирования и разработки (СИПР) позволила руководителю проекта выполнить прогнозирование развития проекта и своевременно принять проектные и организационные решения, повышающие эффективность разработки средства автоматизации.

Заключение

В ходе диссертационного исследования получены следующие научные результаты.

1. Онтологически-ориентированная модель языка диаграмм UML и онтологическая модель шаблона проектирования.
2. Меры архитектурного подобия программных проектов; меры выраженности шаблона проектирования в рассматриваемых программных продуктах.
3. Методика переноса знаний из концептуальных моделей в онтологию OWL.
4. Алгоритм трансформации UML – диаграммы классов в онтологию формата OWL.
5. Алгоритм агрегации методов прогнозирования временных рядов показателей состояния проекта разработки программно-аппаратного комплекса.

Для решения задач исследования и проверки научных результатов была разработана программная система интеллектуального проектирования и разработки, отличающаяся от известных:

- поддержкой проектов на каждом этапе жизненного цикла;
- унифицированным форматом хранения знаний о проекте;
- блоком анализа, позволяющего оценить состояние проекта автоматически.

Были проведены вычислительные эксперименты, подтверждающие эффективность формирования информационного обеспечения проекта ПАК на основе извлечения знаний о предметной области и проекте из UML-диаграмм в формате OWL и эксперименты по отбору проектов, релевантных техническому заданию ПАК.

Эффективность подсистемы ПАД подтверждают эксперименты, по прогнозированию развития проекта ПАК на основе временных рядов формирования концептов проекта: классов, индивидуалов, отношений,

свойств, аксиом. Включение инструмента прогнозирования развития проекта ПАК в системы автоматизированного проектирования позволяет сократить количество смысловых ошибок проекта, что приводит к экономии средств, затрачиваемых на разработку ПАК.

Внедрение подсистемы ПИП в разработки в крупной проектной организации ФНПЦ АО «НПО «Марс» и на крупном промышленном предприятии АО «Авиастар-СП» позволили успешно и в срок реализовать проект ПАК «Система интеллектуального поиска и анализа в Интернет-СМИ и социальных сетях» в первом случае и «Оптимизация ресурсов на основе производственно-технологического моделирования в условиях мультипродуктовой производственной программы» во втором.

Список литературы

1. Артюхов, М.В. Построение массива методов для прогнозирования временных рядов / М.В. Артюхов, Г.Ю. Гуськов, А.А. Романов, И.А. Тимина // Материалы VIII международной конференции Интегрированные модели и мягкие вычисления в искусственном интеллекте: труды конференции. – Т.3. – М.: Физматлит. – 2015. – С. 332 – 341..

2. Артюхов, М.В. Прогнозирование временных рядов коллективами методов / Г.Ю. Гуськов, А.А. Романов, И.А. Тимина // Радиотехника. – М.: Радиотехника. – № 6. – 2015. – С. 48-54.

3. Афанасьева, Т.В. Интеграция нечетко-гранулярных и онтологических методов в задаче анализа временных рядов / Т.В. Афанасьева, Г.Ю. Гуськов, А.М. Наместников, Н.Г. Ярушкина // Автоматизация процессов управления. – 2015. – №2(40). – С. 72-79.

4. Афанасьева, Т.В. Интеллектуальный веб-сервис экспресс анализа экономического состояния предприятия / Г.Ю. Гуськов, И.Г. Перфильева, А.А. Романов, И.А. Тимина, Н.Г. Ярушкина // Четырнадцатая национальная конференция по искусственному интеллекту с международным участием КИИ-2014 (24-27 сентября 2014 г., г. Казань, Россия): труды конференции. – Казань. – 2014. – Т.2. – С. 213-221.

5. Батыршин, И.З. Нечеткие гибридные системы. Теория и практика / И.З. Батыршин, А.О. Недосекин, А.А. Стецко, В.Б. Тарасов, А.В. Язенин, Н.Г. Ярушкина; под ред Н.Г. Ярушкиной. – М.: ФИЗМАТЛИТ, 2007. – 208 с.

6. Бениаминов, Е.М. Некоторые проблемы широкого внедрения онтологий в IT и направлени их развития / Е.М. Бениаминов // Труды Симпозиума "Онтологическое моделирование". –М.: ИПИ РАН – 2008. – С.71-82.

7. Боргест, Н.М. Границы онтологии проектирования / Н.М. Боргест // Онтология проектирования. – Самара: Предприятие "Новая техника".– №1(7). – 2017 – С. 7-33..

8. Воробьев, А.М. Создание единого информационного пространства предприятия / А.М. Воробьев, Д.К. Щеглов //Материалы семинара

«Развитие информационной инфраструктуры Концерна». – М.: ОАО «Концерн ПВО «Алмаз-Антей», 2007. – С. 93–104.

9. Вязгин, В.А. Математические методы автоматизированного проектирования / В.А. Вязгин, В.В. Федоров. – М.: Высш. школа, 1989.

10. Гаврилова, Т.А. Базы знаний интеллектуальных систем / Т.А. Гаврилова, В.Ф. Хорошевский. – СПб.: Питер, 2000.

11. Голенков, В.В. Онтологическое проектирование интеллектуальных систем / В.В. Голенков // Открытые семантические технологии проектирования интеллектуальных систем (OSTIS-2017): материалы VII Междунар. научн.техн. конф. – Минск: БГУИР. – 2017. – С. 37-56.

12. ГОСТ 34.003-90 Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Термины и определения. – М.: Стандартинформ, 2009. – 15 с.

13. Грищенко, М.А. Разработка экспертных систем на основе трансформации информационных моделей предметной области / М.А. Грищенко, А.Ю. Юрин, А.И. Павлов // Программные продукты и системы. – Тверь: НИИ «Центрпрограммсистем». – №3. – 2013. – С. 143-147.

14. Гуськов, Г.Ю. Разработка многоагентной системы извлечения знаний из гетерогенных источников / Г.Ю. Гуськов, В.С. Мошкин, А.М. Наместников, А.А. Филиппов, Н.Г. Ярушкина // Радиотехника. – М.: Радиотехника. – 2016. – №9. – С. 57-63.

15. Гуськов, Г.Ю. Интеллектуальная система управления проектами разработки программного обеспечения / Г.Ю. Гуськов, А.М. Наместников, И.А. Тимина, Н.Г. Ярушкина// Вестник Ростовского государственного университета путей сообщения, –2016.–№ 3.(63) – С. 31-36.

16. Гуськов, Г.Ю. Подход к поиску похожих по структуре проектов, основанный на онтологии языка uml / Г.Ю.Гуськов, А.М. Наместников, Н.Г. Ярушкина // Радиотехника. – М.:Радиотехника. – 2017. – № 6 – С. 122-127.

17. Гуськов, Г.Ю. Формирование метрик для нечеткой кластеризации

репозитория программных проектов на основе их семантического и структурного сходства / Г.Ю.Гуськов, А.М. Наместников, Н.Г.Ярушкина// Четвертая Всероссийская Поспеловская конференция с международным участием «Гибридные и синергетические интеллектуальные системы» ГИСИС-2018:Труды конференции. – Калининград: Изд-во БФУ им.И.Канта. – 2018. – С. 407-414.

18. Гуськов, Г.Ю. Программная система преобразования UML-диаграмм в онтологии на языке OWL / Г.Ю. Гуськов, А.М. Наместников // Пятнадцатая национальная конференция по искусственному интеллекту с международным участием КИИ-2016 (3-7 октября 2016 г., г. Смоленск, Россия): труды конференции. – 2016. – Т.3. – С. 270-278.

19. Гуськов, Г.Ю. Система управления программными проектами на основе онтологического подхода / Г.Ю. Гуськов, А.М. Наместников // Автоматизация процессов управления. – 2016. – №3(45). – С. 88-94.

20. Гуськов, Г.Ю. Алгоритм и программа преобразования проектных диаграмм UML в предметную онтологию OWL / Г.Ю. Гуськов // Вузовская наука в современных условиях: Труды конференции. – Ульяновск: Изд-во УлГТУ. – 2017. –Т. 2. – С. 129-132.

21. Гуськов, Г.Ю. Инструмент управления программным проектом на основе результатов анализа онтологии, построенной по uml-диаграммам / Г.Ю.Гуськов// Пятый международный молодежный инновационный форум. – Ульяновск: Изд-во УлГТУ. – 2016. – С. 74-77.

22. Добров, Б.В. Лингвистическая онтология по естественным наукам и технологиям: основные принципы разработки и текущее состояние / Б.В. Добров, Н.В. Лукашевич // Десятая национальная конференция по искусственному интеллекту с международным участием (Обнинск, 25–28 сентября 2006 г.). – М.: Физматлит, 2006.

23. Дородных, Н.О. Использование диаграмм классов UML для формирования продукционных баз знаний / Н.О. Дородных, А.Ю. Юрин // Программная инженерия. – М.:Издательство "Новые технологии". – № 4. – 2015. – С. 3-9.

24. Загоруйко, Н.Г. Формирование базы лексических функций и других отношений для онтологии предметной области / Н.Г. Загоруйко, А.М. Налетов, А.А. Соколова, В.А. Чурикова // Труды международной

конференции Диалог-2004. – М.: Наука, 2004. – С. 202–204.

25. Заде, Л.А. Понятие лингвистической переменной и его применение к принятию приближенных решений / Л.А. Заде. – М.: Мир, 1976. – 165 с.

26. Лукашевич Н.В. Тезаурусы в задачах информационого поиска. М.:Изд-во МГУ, 2011. – 512 С.

27. Наместников, А.М. Формирование информационных запросов к электронному архиву на основе концептуального индекса / А.М. Наместников, Р.А. Субхангулов // Радиотехника. – М.: Радиотехника. – №7. – 2014. – С. 126-129.

28. Нариньяни, А.С. Кентавр по имени ТЕОН: Тезаурус+Онтология / А.С. Нариньяни // Труды Международной конференции ДИАЛОГ-2001. – М. – 2001. – Т.1. – С.184-188.

29. Норенков, И.П. Основы автоматизированного проектирования: учеб. для вузов / И.П. Норенков. – 4-е изд., перераб. и доп. – М.: Изд-во МГТУ им. Н. Э. Баумана, 2009.

30. Поспелов, Д.А. Искусственный интеллект: В 3 кн. Кн. 2. Модели и методы: Справочник /Под ред. Д.А. Поспелова. — М.: Радио и связь, 1990.—304 с

31. Филиппов, А.А. Концептуальный индексатор проектных документов / А.А. Филиппов // Тезисы докладов 45-й научно-технической конференции УлГТУ «Вузовская наука в современных условиях» (24-29 января 2011 года). – Ульяновск: УлГТУ, 2011. – С. 181.

32. Филиппов, А.А. Формирование навигационной структуры электронного архива технических документов на основе онтологических моделей / А.А. Филиппов // Автоматизация процессов управления. – 2013. – № 3(33). – С. 61-68.

33. Ярушкина, Н.Г. Мягкие вычисления в интеллектуализации процессов проектирования в САПР / Н.Г. Ярушкина, Т.В. Афанасьева, А.М. Наместников, В.С. Мошкин, Г.Ю. Гуськов // Нечеткие системы и мягкие вычисления. – Тверь: Изд-во ТвГУ –2017. – № 1(12). – С. 19-44.

34. Ярушкина, Н.Г. Разработка программной системы семантического анализа контента социальных медиа / Н.Г. Ярушкина, В.С. Мошкин,

А.А. Филиппов, Г.Ю. Гуськов Г.Ю, А.М. Наместников, А.А. Романов // Радиотехника. – М.Радиотехника–2018. – №. 6. – С. 73-79.

35. Abrahamsson P., Salo O., Ronkainen J., Warsta J. Agile Software Development Methods: Review and Analysis // Agile Software Development, Vol. 1, 2002. pp. 31-59.

36. Agile-манифест разработки программного обеспечения URL: http://agilemanifesto.org/iso/ru/manifesto.html (дата обращения: 06.08.2018).

37. Akaike H. A New Look At The Statistical Model Identification // IEEE Transactions on Automatic Control, Vol. 19, No. 6, 1974. pp. 716 - 723.

38. Almeida Ferreira D., Silva A. UML to OWL Mapping overview An analysis of the translation process and supporting tools // 7th Conference of Portuguese Association of Information Systems. 2007. Vol. 2. pp. 192-207.

39. Astrova I., Kalja A., Korda N. Rule-based transformation of SQL relational databases to OWL ontologies // The 2nd International Conference on Metadata & Semantics Research. 2007. pp. 415-424.

40. Azzem Akbar M., Sang J., Khan A., Fazal-E-Amin, Nasrullah, Hussain S., Khalid Soahil M., Xiang H., Cai B. Statistical Analysis of the Effects of Heavyweight and Lightweight Methodologies on the Six-Pointed Star Model // IEEE Access, Vol. 6, 2018. pp. 8066-8079.

41. Barrasa J., Corcho O., Gomez-Perez A. R2O, an Extensible and Semantically based Database-to-Ontology Mapping Language // Proceedings of the 2nd Workshop on Semantic Web and Databases. 2004. pp. 1069–1070.

42. Basili V.R., Turner A.J. Iterative Enhancement: A Practical Technique for Software Development // IEEE Transactions on Software Engineering, Vol. 1, No. 1, 1975. pp. 390-396.

43. Bergandy J. Work in Progress - Software Engineering Capstone Project with Rational Unified Process // Frontiers in Education Conference. 2008. Vol. FIE 2008. 38th Annual. pp. S4J-1-S4J-2.

44. Bobillo F., Straccia U. Fuzzy Ontology Representation using OWL 2 // International Journal of Approximate Reasoning, Vol. 52, No. 7, October

2010. pp. 1073-1094.

45. Bobillo F., Straccia U. Representing Fuzzy Ontologies in OWL 2 // WCCI IEEE World Congress on Computational Intellegence. Barcelona. 2009. pp. 2696- 2700.

46. Booch G., Rumbaugh J., Jacobson I. Unified Modeling Language User Guide. 2nd ed. Addison-Wesley Object Technology Series, 2005. 496 pp.

47. Box G., Cox D. An Analysis of Transformations (with Discussion) // Journal of the Royal Statistical Society. 1964. pp. 211-252.

48. Cavanaugh J. Unifying the Derivations for the Akaike and Corrected Akaike Information Criteria // Statistics & Probability Letters, Vol. 33, No. 2, 1997. pp. 201-208.

49. CIF 2015 dataset [Электронный ресурс] URL: http://irafm.osu.cz/cif2015/main.php?c=Static&page=download (дата обращения: 07.08.2018).

50. Cockburn A. Agile Software Development. 2001: Addison-Wesley Professional. 304 pp.

51. Dillon T., Chang E., Wongthongtham P. Ontology-based software engineering- software engineering 2.0. // Australian Software Engineering Conference, IEEE Computer Society. 2008. pp. 13-23.

52. Emdad A. Use of ontologies in software engineering // 17th International Conference on Software Engineering and Data Engineering. 2008. pp. 145-150.

53. Everette S., Gardner J. Exponential smoothing: The state of the art // International Journal of Forecasting, Vol. 22, No. 4, 2006. pp. 637-666.

54. García-Moya L., Anaya-Sanchez H., Berlanga-Llavori R. Retrieving product features and opinions from customer reviews //. IEEE Intelligent Systems, Vol. 28, No. 3, 2013. pp. 19-27.

55. Gasevic D., Djuric D., Devedzic V., Damjanovic V. Converting UML to OWL ontologies // 13th international conference on World Wide Web. 204. pp. 488-489.

56. Ge P., Wang J., Ren P., Gao H., Luo Y. A new improved forecasting method integrated fuzzy time series with the exponential smoothing method // International Journal of Environment, Vol. 51, No. 3-4, 2013. pp. 206–221.

57. Goy A., Magro D. Towards an ontology-based software documentation management - a case study, 2012. pp. 125–131.

58. Gruninger M., Fox M.S. Methodology for the Design and Evaluation of Ontologies // Paper presented at the Workshop on Basic Ontological Issues in Knowledge Sharing, 19–20 August, Montreal, Quebec,Canada. 1995.

59. Guskov G., Afanasieva T., Yarushkina H., Zavarzin D., Romanov A. Time series forecasting using combination of exponential models and fuzzy techniques // Proceedings of the First International Scientific Conference Intelligent Information Technologies for Industry (IITI'16). 2016. Vol. 1. pp. 41 - 50.

60. Guskov G., Yarushkina N., Afanasieva T., Zavarzin D. Fuzzy trends data mining in knowledge discovery process // Proceedings of the First Conference Creativity in Intelligent Technologies and Data Science, CIT&DS 2015. 2015. pp. 115-123.

61. Guskov G.Y., Namestnikov A.M., Yarushkina N.G. Approach to the search for similar software projects based on the UML ontology // Intelligent Information Technologies for Industry. Sochi. 2017. Vol. 2. pp. 3–10.

62. Guskov G.Y., Namestnikov A.M. Approach to determining the structural similarity of software projects // Open Semantic Technologies for Intelligent Systems. Minsk. 2018. Vol. 1. pp. 269-272.

63. Guskov G.Y., Namestnikov A.M. Ontological mapping for conceptual models of software system // Open Semantic Technologies for Intelligent Systems. Minsk. 2017. Vol. 1. pp. 111-117.

64. Happel H., Seedorf S. Applications of ontologies in software engineering // Proceedings of International Workshop on Semantic Web Enabled Software Engineering (SWESE06). 2006. pp. 27-30.

65. Hossein S., Sartipi K. Dynamic analysis of software systems using execution pattern mining // ICPC, IEEE Computer Society, 2006. pp. 84–88.

66. Huang Y.L., Horng S.J., He M., Fan P., Kao T.W., Khan M.K., Lai J.L., Kuo L.H. A hybrid forecasting model for enrollments based on aggregated fuzzy time series and particle swarm optimization // Expert Systems with Applications, Vol. 38, No. 7, 2011. pp. 8014-8023.

67. institute P.M. A guide to the Project Management Body of Knowledge (PMBOK guide) (PMBOK Guides). Project Management Institute, 2017. 592 pp.

68. Introducing the Neo4j Graph Platform [Электронный ресурс] URL: Introducing the Neo4j Graph Platform (дата обращения: 07.08.2018).

69. Jeffrey E.J., Jeffrey S.P. The Fuzzy Logic Method for Simpler Forecasting // International Journal of Engineering Business Management, Vol. 3, No. 3, 2011. pp. 25-52.

70. Jiang A., Mei C., E J., Shi Z. Nonlinear combined forecasting model based on fuzzy adaptive variable weight and its application // Journal of Central South University of Technology, Vol. 17, No. 4, 2010. pp. 863–867.

71. Kolassa S. Combining exponential smoothing forecasts using Akaike weights // International Journal of Forecasting, Vol. 27, No. 2, 2011. pp. 238-251.

72. Konstantinou N., Spanos D.E., Chalas M., Solidakis E., Mitrou N. VisAVis: An Approach to an Intermediate Layer between Ontologies and Relational Database Contents // Proceedings of the CAISE'06 Third International Workshop on Web Information Systems Modeling (WISM '06). Luxemburg. 2006.

73. Koukias A., Koukias D. Rule-based mechanism to optimize asset management using a technical documentation ontology // IFAC-PapersOnLine. 2015. Vol. 48. pp. 1001 – 1006.

74. Koukias A., Nadoveza D., Kiritsis D. An Ontology based Approach for Modelling Technical Documentation towards Ensuring Asset Optimization // International Journal of Product Lifecycle Management, Vol. 8, No. 1, 2015. pp. 24-45.

75. Larman C., Basili V.R. Iterative and incremental development: a brief history // Computer, Vol. 36, No. 6, 2003. pp. 47–56.

76. Ma Z., Zhang F., Yan L., Cheng J. Representing and reasoning on fuzzy UML models: A description logic approach // Expert Systems with Applications, Vol. 38, No. 3, 2011. pp. 2536–2549.

77. MacCormack, A D. Product-Development Practices That Work: How Internet Companies Build Software // MIT Sloan Management Review, Vol. 2, 2001. pp. 75-84.

78. Makridakis S. Accuracy measures: Theoretical and practical concerns // International Journal of Forecasting, Vol. 9, No. 4, 1993. pp. 527-529.

79. Marcus A., Rajlich V., Buchta J., Petrenko M., Sergeyev A. Static techniques for concept location in object-oriented code // 13th Int'l Workshop on Program Comprehension. 2005. pp. 33–42.

80. Marvin M. A Framework for Representing Knowledge // In The Psychology of Computer Vision, 1975. pp. 211–277.

81. MongoDB. For Giant ideas [Электронный ресурс] URL: : https://www.mongodb.com (дата обращения: 07.08.2018).

82. Namestnikov A.M., Filippov A.A., Avvakumova V. An ontology based model of technical documentation fuzzy structuring // CEUR Workshop Proceedings. SCAKD 2016. Moscow. 2016. Vol. 1687. pp. 63-74.

83. Novák V., Perfilieva I., Jarushkina N. A general methodology for managerial decision making using intelligent techniques // In: Recent Advances in Decision Making. Berlin, Heidelberg: Springer, Berlin, Heidelberg, 2009. pp. 103-120.

84. Novak V., Perfiljeva I., Dvorak A. Analysis and Forecasting of Time Series // In: Insight into Fuzzy Modeling / Ed. by Novak V., Perfiljeva I., Dvorak A. John Wiley & Sons, 2016. pp. 209-235.

85. OMG® Unified Modeling Language (OMG UML) Version 2.5.1 [Электронный ресурс] URL: https://www.omg.org/spec/UML/About-UML/ (дата обращения: 07.08.2018).

86. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition) URL: https://www.w3.org/TR/owl2-syntax/ (дата обращения: 07.08.2018).

87. Pegels C.C. Exponential forecasting: Some new variations // Management Science, Vol. 15, No. 5, 1969. pp. 311-315.

88. Perfiljeva I. Fuzzy transforms: Theory and applications // Fuzzy Sets and Systems, Vol. 157, No. 8, 2008. pp. 993-1023.

89. Ramos Carvalho N., Joao Almeida J., Rangel Henriques P., Joao Varanda Pereira M. Ontology-driven measurement of semantic relatedness between source code elements and problem domain concepts. Cham: Springer International Publishing, 2014. pp. 116–131.

90. Royce W.W. Managing the development of large software systems // 9th international conference on Software Engineering. Los Angeles. 1970. Vol. 26. pp. 328–388.

91. Schwarz G. Estimating the Dimension of a Model // The Annals of Statistics, Vol. 6, No. 2, 1978. pp. 461-464.

92. Song Q., Chissom B.S. Fuzzy time series and its model. Fuzzy Sets and Systems // Fuzzy Sets and Systems, Vol. 54, No. 3, 1993. pp. 269–277.

93. Taylor J.W. Exponential smoothing with a damped multiplicative trend // International Journal of Forecasting , Vol. 19, No. 4, 2003. pp. 715-725.

94. Turner M.S.V. Microsoft solutions framework essentials: Building successful technology solutions. 1st ed. Microsoft Press, 2006. 342 pp.

95. Uschold M., Gruninger M. Ontologies: Principles, Methods and Applications // In Knowledge Engineering Review, Vol. 11, No. 2, 1996. pp. 93-155.

96. Viertl R. Statistical Methods for Fuzzy Data by Reinhard Viertl. 1st ed. Wiley, 2011. 268 pp.

97. Vovk V. Aggregating strategies // Proceedings of the third annual workshop on Computational learning theory. Rochester, New York, USA. 1990. pp. 371-386.

98. Wongthongtham P., Pakdeetrakulwong U., Marzooq S. Ontology annotation for software engineering project management in multisite distributed software development environments // In: Software Project Management for Distributed Computing. Cham: Springer International Publishing AG , 2017.

pp. 315–343.

99. Xu Z., Zhang S., Dong Y. Mapping between Relational Database Schema and OWL Ontology for Deep Annotation // In Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence. Hong Kong. 2006. pp. 379-384.

100. Yarushkina N., Filippov A., Moshkin V. Development of the unified technological platform for constructing the domain knowledge base through the context analysis // Communications in Computer and Information Science, Vol. 754, 2017. pp. 62-72.

101. Yarushkina N.G., Moshkin V.S., Filippov A.A., Guskov G.Y. Developing a Fuzzy Knowledge Base and Filling It with Knowledge Extracted from Various Documents // Artificial Intelligence and Soft Computing. Zakopane. 2018. Vol. 1. pp. 799-811.

102. Yarushkina N.G., Perfiljeva I., Afanasieva T., Igonin A., Romanov A., Shishkina V. Time Series Processing and Forecasting Using Soft Computing Tools // International Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular-Soft Computing. 2011. pp. 155-162.

103. Zedlitz J., Jorke J., Luttenberger N. From UML to OWL 2 // Proceedings of Knowledge Technology Week. 2012. pp. 154--163.

Приложения

Приложения 1. Свидетельства о регистрации программ для ЭВМ

.

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2015617046

Интегрированная система нечетких методов прогнозирования временных рядов

Правообладатель: ***федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Ульяновский государственный технический университет» (RU)***

Авторы: ***Ярушкина Надежда Глебовна (RU), Романов Антон Алексеевич (RU), Гуськов Глеб Юрьевич (RU), Тимина Ирина Александровна (RU)***

Заявка № **2015613884**

Дата поступления **12 мая 2015 г.**

Дата государственной регистрации в Реестре программ для ЭВМ ***29 июня 2015 г.***

Врио руководителя Федеральной службы по интеллектуальной собственности

Л.Л. Кирий

Приложения 2. Фрагменты исходного кода СИПРИС

UMLtoOWLparser – C# application

UClass.cs

```
namespace UMLtoOWLparser.Universal
{
    public class UClass : UStructuredThing
    {
        public UClass()
        {
            parentOntID = null;
        }
        private bool isAbstract = false;
        public bool IsAbstract { get; set; }
        public string name { get; set; }
        public string ontID { get; set; }
        public List<string> generalizations = new List<string>();
        public string parentOntID { get; set; }
        public List<string> childs = new List<string>();
        public List<UAttribute> attributes = new List<UAttribute>();
        public List<UOperation> operations = new List<UOperation>();

        public override string getName()
        {
            return name;
        }
        public override string getOntID()
        {
            return ontID;
        }
        public override string getParentOntID()
        {
            return parentOntID;
        }
    }
```

UDirectedRelationship.cs

```
namespace UMLtoOWLparser.Universal
{
    public class UDirectedRelationship : URelationship
    {
        List<UElement> toElements;
        List<UElement> fromElements;
    }
}
```

UElement.cs

```
namespace UMLtoOWLparser.Universal
{
    public class UElement
    {
        protected List<UElement> childs;
```

```
        protected List<string> comments;
    }
}
```

UInterface.cs

```
namespace UMLtoOWLparser.Universal
{
    public class UInterface : UStructuredThing
    {
        public UInterface()
        {
            parentOntID = null;
        }
        public string name { get; set; }
        public string ontID { get; set; }
        public string parentOntID { get; set; }
        public List<string> childs = new List<string>();
        public List<UOperation> operations = new List<UOperation>();
        public override string getName()
        {
            return name;
        }
        public override string getOntID()
        {
            return ontID;
        }
        public override string getParentOntID()
        {
            return parentOntID;
        }
    }
}
```

UAttribute.cs

```
namespace UMLtoOWLparser.Universal
{
    public class UAttribute : UStructuredThing
    {
        public string name { get; set; }
        public string type { get; set; }
        public string visability { get; set; }
        public bool defaultType { get; set; }
        public string ontID { get; set; }
        public string parentOntID { get; set; }
        public override string getName()
        {
            return name;
        }
        public override string getOntID()
        {
            return ontID;
        }
        public override string getParentOntID()
        {
            return parentOntID;
        }
    }
}
```

XMIparser.cs

```
namespace UMLtoOWLparser.XMIdir
{
   public class XMIparser
   {
      private string xmiversion = "";
      //private XDocument xmidoc;
      private UDoc unidoc = new UDoc();
      public UDoc parseXMI(string XMIFilePath)
      {
         getXMIversion(XMIFilePath);
#warning MakeversionSelecter
         if (xmiversion == "2.1")
         {
            XmlSerializer serializer = new XmlSerializer(typeof(XMI));
            FileStream fs = new FileStream(XMIFilePath, FileMode.Open);
            XmlReader reader = XmlReader.Create(fs);
            XMI obj = (XMI)serializer.Deserialize(reader);
            fs.Close();
            //TODO облагородить заполнение  enum, class, dt....
ектное решение
            unidoc.Classes = getClassHierarchy(obj);
            unidoc.Interfaces = getClassInterfaces(obj);
            unidoc.datatypes = getDataTypes(obj);
            DataTypeMatch();
            unidoc.Relationships = getRelationshipFromDoc(obj, unidoc);
            //unidoc.Relationships.RemoveAll(x => x == null);
            return unidoc;
         }
         return null;
      }
      private void DataTypeMatch()
      {
         foreach (var _class in unidoc.Classes)
         {
            foreach (var att in _class.attributes)
            {
               foreach (var dt in unidoc.datatypes)
               {
                  if (att.type == dt.Item2)
                  {
                     att.type = dt.Item1;
                  }
               }
               foreach (var cl in unidoc.Classes)
               {
                  if (att.type == cl.ontID)
                  {
                     att.type = cl.name;
                  }
               }
               foreach (var en in unidoc.Enums)
               {
                  if (att.type == en.ontID)
                  {
                     att.type = en.name;
                  }
               }
            }
         }
      }
      private List<Tuple<string, string>> getDataTypes(XMI obj)
      {
         List<Tuple<string, string>> res = new List<Tuple<string, string>>();
         var model = obj.Items.FirstOrDefault(i => i.Name == "uml:Model");
         var childs = model.ChildNodes;
         foreach (XmlNode child in childs)
         {
```

```
        if (child.Name == "ownedMember" && child.Attributes.GetNamedItem("xmi:type").Value == "uml:DataType")
        {
            string name = child.Attributes.GetNamedItem("name").Value;
            string xmiId = child.Attributes.GetNamedItem("xmi:id").Value;
            res.Add(new Tuple<string, string>(name, xmiId));
        }
    }
    return res;
}
private List<UClass> getClassHierarchy(XMI obj)
{
    List<UClass> classes = new List<UClass>();
    var model = obj.Items.FirstOrDefault(i => i.Name == "uml:Model");
    var childs = model.ChildNodes;
    //parsing uml:Model
    foreach (XmlNode child in childs)
    {
        UClass cl = null;
        UEnum en = null;
        //parsing Package если классы внутри пакета
        if (child.Name == "ownedMember" && child.Attributes.GetNamedItem("xmi:type").Value == "uml:Package")
        {
            var pacChilds = child.ChildNodes;
            //внутри пакета может быть много классов
            foreach (XmlNode pacChild in pacChilds)
            {

                cl = getClass(pacChild);
                if (cl != null)
                {
                    classes.Add(cl);
                    continue;
                }
            }
        }
        //если класс не в пакете
        cl = getClass(child);
        if (cl != null)
        {
            classes.Add(cl);
            continue;
        }
        //если элемент диаграммы enum
        en = GetEnum(child);
        if (en != null)
        {
            unidoc.Enums.Add(en);
            continue;
        }
    }
    return classes;
}
private List<UInterface> getClassInterfaces(XMI obj)
{
    List<UInterface> interfaces = new List<UInterface>();
    var model = obj.Items.FirstOrDefault(i => i.Name == "uml:Model");
    var childs = model.ChildNodes;
    //parsing uml:Model
    foreach (XmlNode child in childs)
    {
        UInterface inter = null;
        UEnum en = null;
        //parsing Package если классы внутри пакета
        if (child.Name == "ownedMember" && child.Attributes.GetNamedItem("xmi:type").Value == "uml:Package")
        {
            var pacChilds = child.ChildNodes;
            //внутри пакета может быть много классов
            foreach (XmlNode pacChild in pacChilds)
            {
```

```
                    inter = getInterface(pacChild);
                    if (inter != null)
                    {
                        interfaces.Add(inter);
                        continue;
                    }
                }
            }
            //если класс не в пакете
            inter = getInterface(child);
            if (inter != null)
            {
                interfaces.Add(inter);
                continue;
            }
            //если элемент диаграммы enum
            en = GetEnum(child);
            if (en != null)
            {
                unidoc.Enums.Add(en);
                continue;
            }
        }
        return interfaces;
    }
    private UInterface getInterface(XmlNode inter)
    {
        UInterface curInterface = null;
        //проверка на то, что элемент xml является интерфейсом
        if (inter.Name == "ownedMember" && inter.Attributes.GetNamedItem("xmi:type").Value == "uml:Interface")
        {
            curInterface = new UInterface { name = inter.Attributes.GetNamedItem("name").Value, ontID =
inter.Attributes.GetNamedItem("xmi:id").Value };
        }
        return curInterface;
    }
    private UClass getClass(XmlNode cl)
    {
        UClass curClass = null;
        //проверка на то, что элемент xml является классом
        if (cl.Name == "ownedMember" && cl.Attributes.GetNamedItem("xmi:type").Value == "uml:Class")
        {
            curClass = new UClass { name = cl.Attributes.GetNamedItem("name").Value, ontID =
cl.Attributes.GetNamedItem("xmi:id").Value };
            curClass.IsAbstract = cl.Attributes.GetNamedItem("isAbstract").Value.Equals("true");
            curClass.generalizations = getGeneralizationFromClass(cl);
            curClass.attributes = getAttributes(cl, curClass.ontID);
            curClass.operations = getOperations(cl, curClass.ontID);
            if (curClass.generalizations.Count != 0)
            {
                curClass.parentOntID = curClass.generalizations.FirstOrDefault();
            }
        }
        return curClass;
    }
    private List<UOperation> getOperations(XmlNode elem, string parentOntID)
    {
        List<UOperation> operations = new List<UOperation>();
        var childs = elem.ChildNodes;
        foreach (XmlNode child in childs)
        {
            if (child.Name == "ownedOperation")
            {
                XmlNode nameEl = child.Attributes.GetNamedItem("name");
                string nameVal = nameEl != null ? nameEl.Value : "";
                string ontID = child.Attributes.GetNamedItem("xmi:id").Value;
                string typeVal = "void";
                List<UOperationParam> uoperationParams = new List<UOperationParam>();
                foreach (XmlNode paramsEl in child.ChildNodes)
                {
```

```
                if (paramsEl.Name == "ownedParameter")
                {
                    XmlNode kindEL = paramsEl.Attributes.GetNamedItem("kind");
                    XmlNode typeEl = paramsEl.Attributes.GetNamedItem("type");
                    //если ownedParametr - возвращаемое значение
                    if (kindEL != null && kindEL.Value == "return")
                    {
                        if (typeEl != null)
                            typeVal = paramsEl.Attributes.GetNamedItem("type").Value; // null?
                        continue;
                    }
                    else
                    //если ownedParametr - параметр операции
                    {
                        XmlNode paramNameEl = paramsEl.Attributes.GetNamedItem("name");
                        if (typeEl != null)
                            typeVal = paramsEl.Attributes.GetNamedItem("type").Value;
                        uoperationParams.Add(new UOperationParam { name = paramNameEl.Value, type = typeVal });
                        continue;
                    }

                }
            }
            //Attributes.GetNamedItem("type");

            bool defaultType = Trunslation.Types.Exists(t => t.Item2 == typeVal);
            operations.Add(new UOperation { name = nameVal, returnType = typeVal, operationParams = uoperationParams,
ontID = ontID, parentOntID = parentOntID});
        }
    }
    return operations;
}
private List<UAttribute> getAttributes(XmlNode elem, string parentOntID)
{
    List<UAttribute> res = new List<UAttribute>();
    var childs = elem.ChildNodes;
    foreach (XmlNode child in childs)
    {
        if (child.Name == "ownedAttribute")
        {
            XmlNode nameEl = child.Attributes.GetNamedItem("name");
            XmlNode typeEl = child.Attributes.GetNamedItem("type");
            string ontID = child.Attributes.GetNamedItem("xmi:id").Value;
            string nameVal = nameEl != null ? nameEl.Value : "";
            string typeVal = typeEl != null ? typeEl.Value : "";
            bool defaultType = true;
            if (Trunslation.Types.Exists(t => t.Item2 == typeVal))
            {
                defaultType = true;
            }
            else
            {

                defaultType = false;
            }

            res.Add(new UAttribute
            {
                name = nameVal,
                type = typeVal,
                defaultType = defaultType,
                ontID = ontID,
                parentOntID = parentOntID
            });
        }
    }
    return res;
}
```

```
        private List<URelationship> getRelationshipFromDoc(XMI obj, UDoc doc)
        {
            List<URelationship> reals = new List<URelationship>();
            var model = obj.Items.FirstOrDefault(i => i.Name == "uml:Model");
            var childs = model.ChildNodes;
            //parsing uml:Model
            foreach (XmlNode child in childs)
            {
                URelationship real = null;
                UEnum en = null;

                if (child.Name == "ownedMember" && child.Attributes.GetNamedItem("xmi:type").Value == "uml:Package")
                {
                    var pacChilds = child.ChildNodes;
                    foreach (XmlNode pacChild in pacChilds)
                    {

                        real = getRelationship(pacChild, doc);
                        if (real != null)
                        {
                            reals.Add(real);
                            continue;
                        }
                    }
                }
                real = getRelationship(child, doc);
                if (real != null)
                {
                    reals.Add(real);
                    continue;
                }

            }
            return reals;
        }
        private URelationship getRelationship(XmlNode elem, UDoc doc)
        {
            //проверка на то, что элемент xml является реализацией
            if (elem.Name == "ownedMember" && elem.Attributes.GetNamedItem("xmi:type").Value == "uml:Realization")
            {
                return new URelationship
                {
                    startElem = getRelationshipEnd(doc, doc.getUStructuredThings().FirstOrDefault(x => x.getOntID() ==
elem.Attributes.GetNamedItem("client").Value)),
                    endElem = getRelationshipEnd(doc, doc.getUStructuredThings().FirstOrDefault(x => x.getOntID() ==
elem.Attributes.GetNamedItem("supplier").Value)),
                    type = "Realization"
                };
            }
            if (elem.Name == "ownedMember" && elem.Attributes.GetNamedItem("xmi:type").Value == "uml:Association")
            {
                URelationship urel = new URelationship();
                urel.type = "Association";
                foreach (XmlNode subElem in elem.ChildNodes)
                {
                    if (subElem.Name == "ownedEnd" && urel.startElem == null)
                        urel.startElem = getRelationshipEnd(doc, doc.getUStructuredThings().FirstOrDefault(x => x.getOntID() ==
subElem.Attributes.GetNamedItem("type").Value));
                    else if (subElem.Name == "ownedEnd")
                    {
                        urel.endElem = getRelationshipEnd(doc, doc.getUStructuredThings().FirstOrDefault(x => x.getOntID() ==
subElem.Attributes.GetNamedItem("type").Value));
                        break;
                    }
                }
                return urel;
            }
            if (elem.Name == "ownedMember" && elem.Attributes.GetNamedItem("xmi:type").Value == "uml:Dependency")
            {
```

```
            return new URelationship
            {
                startElem = getRelationshipEnd(doc, doc.getUStructuredThings().FirstOrDefault(x => x.getOntID() ==
elem.Attributes.GetNamedItem("client").Value)),
                endElem = getRelationshipEnd(doc, doc.getUStructuredThings().FirstOrDefault(x => x.getOntID() ==
elem.Attributes.GetNamedItem("supplier").Value)),
                type = "Dependency"
            };

        }
        return null;
    }
    private UStructuredThing getRelationshipEnd(UDoc doc, UStructuredThing st)
    {
        if (st == null)
            return null;
        if (st is UClass || st is UInterface || st is UEnum  )
        {
            return st;
        }
        else
        {
            return getRelationshipEnd(doc, doc.getUStructuredThings().FirstOrDefault(x => x.getOntID() ==
st.getParentOntID()));
        }
    }
    private List<string> getGeneralizationFromClass(XmlNode elem)
    {
        List<string> res = new List<string>();
        var childs = elem.ChildNodes;
        foreach (XmlNode child in childs)
        {
            if (child.Name == "generalization")
            {
                res.Add(child.Attributes.GetNamedItem("general").Value);
            }

        }
        return res;
    }
    private UEnum GetEnum(XmlNode en)
    {
        UEnum curEnum = null;
        if (en.Name == "ownedMember" && en.Attributes.GetNamedItem("xmi:type").Value == "uml:Enumeration")
        {
            curEnum = new UEnum
            {
                name = en.Attributes.GetNamedItem("name").Value,
                ontID = en.Attributes.GetNamedItem("xmi:id").Value
            };
            foreach (XmlNode child in en.ChildNodes)
            {
                if (child.Name == "ownedLiteral" && child.Attributes.GetNamedItem("xmi:type").Value ==
"uml:EnumerationLiteral")
                {
                    curEnum.Values.Add(child.Attributes.GetNamedItem("name").Value);
                }
            }

        }
        return curEnum;
    }
    private void getXMIversion(string XMIFilePath)
    {
        XDocument xmidoc = XDocument.Load(XMIFilePath);
        XNamespace xmins = "http://schema.omg.org/spec/XMI/2.1";
        xmiversion = xmidoc.Root.Attribute(xmins + "version").Value;
    }
}
```

}

OWLWriter.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;
using UMLtoOWLparser.Universal;

namespace UMLtoOWLparser.OWL
{
   public static class OWLWriter
   {
      private static UDoc udoc = null;
      public static void Writing(UDoc udoc)
      {
         //задаем путь к нашему рабочему файлу XML
         string fileName = @"Q:\Users\Gleb\Desktop\" + udoc.ontologyName + ".owl";

         #region формальное начало онтологии
         XDocument doc = new XDocument(new XDeclaration("1.0", "utf-8", "no"), new XDocumentType("Ontology", null,
null, " \n\t<!ENTITY xsd \"http://www.w3.org/2001/XMLSchema#\" >\n\t<!ENTITY xml
\"http://www.w3.org/XML/1998/namespace\" > \n\t<!ENTITY rdfs \"http://www.w3.org/2000/01/rdf-schema#\" >
\n\t<!ENTITY rdf \"http://www.w3.org/1999/02/22-rdf-syntax-ns#\" > "),
            new XElement("Ontology",
               new XAttribute("ontologyIRI", "http://www.semanticweb.org/gleb/ontologies/2016/0/" + udoc.ontologyName),
               new XElement("Prefix",
                  new XAttribute("name", "owl"),
                  new XAttribute("IRI", "http://www.w3.org/2002/07/owl#")),
               new XElement("Prefix",
                  new XAttribute("name", "rdf"),
                  new XAttribute("IRI", "http://www.w3.org/1999/02/22-rdf-syntax-ns#")),
               new XElement("Prefix",
                  new XAttribute("name", "xsd"),
                  new XAttribute("IRI", "http://www.w3.org/2001/XMLSchema#")),
               new XElement("Prefix",
                  new XAttribute("name", "rdfs"),
                  new XAttribute("IRI", "http://www.w3.org/2000/01/rdf-schema#"))
               ));
         #endregion
         #region добавление классов
         foreach (var _class in udoc.Classes)
         {
            doc.Root.Add(new XElement("Declaration",
                  new XElement("Class",
                     new XAttribute("IRI", "#" + _class.name))));
            foreach (var att in _class.attributes)
            {
               if (att.defaultType)
               {
                  doc.Root.Add(new XElement("DataPropertyDomain",
                                 new XElement("DataProperty",
                                       new XAttribute("IRI", "#" + att.name)),
                                 new XElement("Class",
                                       new XAttribute("IRI", "#" + _class.name))));
                  doc.Root.Add(new XElement("DataPropertyRange",
                                 new XElement("DataProperty",
                                       new XAttribute("IRI", "#" + att.name)),
                                 new XElement("Datatype",
                                       new XAttribute("abbreviatedIRI", Trunslation.Types.Find(t => t.Item3 ==
att.type.Trim()).Item1))));
               }
               else
               {
                  if (udoc.Enums.Exists(e => e.name == att.type)) //enumiration
                  {
```

```
                doc.Root.Add(new XElement("DataPropertyDomain",
                                new XElement("DataProperty",
                                        new XAttribute("IRI", "#" + att.name)),
                                new XElement("Class",
                                        new XAttribute("IRI", "#" + _class.name))));
                doc.Root.Add(new XElement("DataPropertyRange",
                                new XElement("DataProperty",
                                        new XAttribute("IRI", "#" + att.name)),
                                new XElement("Datatype",
                                        new XAttribute("IRI", att.type))));
            }
            if (udoc.Classes.Exists(c => c.name == att.type))//classifer
            {
                doc.Root.Add(new XElement("Declaration",
                            new XElement("ObjectProperty",
                                new XAttribute("IRI", "#" + att.name))));
                doc.Root.Add(new XElement("ObjectPropertyDomain",
                                new XElement("ObjectProperty",
                                        new XAttribute("IRI", "#" + att.name)),
                                new XElement("Class",
                                        new XAttribute("IRI", "#" + _class.name))));
                doc.Root.Add(new XElement("ObjectPropertyRange",
                                new XElement("ObjectProperty",
                                        new XAttribute("IRI", "#" + att.name)),
                                new XElement("Class",
                                        new XAttribute("IRI", "#" + udoc.Classes.FirstOrDefault(c => c.name ==
att.type).name))));
            }
        }
    }
}
#endregion


#region добавление наследования
foreach (var cl in udoc.Classes)
{
    if (cl.parentOntID != null)
    {
        doc.Root.Add(new XElement("SubClassOf",
            new XElement("Class",
                new XAttribute("IRI", "#" + cl.name)),
            new XElement("Class",
                new XAttribute("IRI", "#" + udoc.Classes.FirstOrDefault(c => c.ontID == cl.parentOntID).name)
            )));
    }
}
#endregion

#region добавление перечислений(enums) базового типа
foreach (var uenum in udoc.Enums)
{
    doc.Root.Add(new XElement("Declaration",
        new XElement("Datatype",
            new XAttribute("IRI", "#" + uenum.name))));
    var dataOneOftag = new XElement("DataOneOf");
    foreach (var elem in uenum.Values)
    {
        dataOneOftag.Add(new XElement("Literal",
                new XAttribute("datatypeIRI", "&rdf;PlainLiteral"), elem));
    }
    var tagenum = new XElement("DatatypeDefinition",
                    new XElement("Datatype",
                            new XAttribute("IRI", "#" + uenum.name)),
                        dataOneOftag);

    doc.Root.Add(tagenum);
}
#endregion
```

```
            //сохраняем наш документ
            var xml = doc.ToString();
            xml = xml.Replace("&amp;amp;", "&");
            //doc.ReplaceNodes(xml);
            File.WriteAllText(fileName, xml);
            //doc.Save(fileName);

        }

        public static void WiritingToStructuredOntology(UDoc _udoc, string fileName)
        {
            udoc = _udoc;
            XDocument doc = XDocument.Load("TemplateOntology.owl");
            foreach (UClass _class in udoc.Classes)
            {
                if (_class.IsAbstract)
                    AddIndividualToOWLClass(doc, "AbstractClass", udoc.ontologyName.Replace(' ', '_') + '_' + _class.name.Replace(' ', '_'));
                else
                    AddIndividualToOWLClass(doc, "SimpleClass", udoc.ontologyName.Replace(' ', '_') + '_' + _class.name.Replace(' ', '_'));
                foreach (UAttribute att in _class.attributes)
                {
                    AddIndividualToOWLClass(doc, "Field", udoc.ontologyName.Replace(' ', '_') + '_' + _class.name.Replace(' ', '_') + '_' + att.name.Replace(' ', '_'));
                }
                foreach (UOperation op in _class.operations)
                {
                    AddIndividualToOWLClass(doc, "Method", udoc.ontologyName.Replace(' ', '_') + '_' + _class.name.Replace(' ', '_') + '_' + op.name.Replace(' ', '_'));
                }
                foreach (string rel in _class.generalizations)
                {
                    UClass child = udoc.Classes.FirstOrDefault(x => x.ontID == rel);
                    string genName = udoc.ontologyName.Replace(' ', '_') + '_' + child.name.Replace(' ', '_') + '_' + _class.name.Replace(' ', '_');
                    AddIndividualToOWLClass(doc, "Generalization", genName);
                    AddObjectProperty(doc, "endWith", genName, _class.name);
                    AddObjectProperty(doc, "startWith", genName, child.name);
                }
            }
            foreach (UInterface _interface in udoc.Interfaces)
            {
                AddIndividualToOWLClass(doc, "Interface", udoc.ontologyName.Replace(' ', '_') + '_' + _interface.name.Replace(' ', '_'));
                foreach (UOperation op in _interface.operations)
                {
                    AddIndividualToOWLClass(doc, "Method", udoc.ontologyName.Replace(' ', '_') + '_' + _interface.name.Replace(' ', '_') + '_' + op.name.Replace(' ', '_'));
                }

            }
            foreach (URelationship real in udoc.Relationships)
            {
                if (real.startElem == null || real.endElem == null)
                    continue;


                List<UStructuredThing> namedElems = udoc.getUStructuredThings();
                UStructuredThing startElem = namedElems.FirstOrDefault(x => x.getOntID() == real.startElem.getOntID());
                UStructuredThing endElem = namedElems.FirstOrDefault(x => x.getOntID() == real.endElem.getOntID());
                string genName = udoc.ontologyName.Replace(' ', '_') + '_' + startElem.getName().Replace(' ', '_') + '_' + endElem.getName().Replace(' ', '_');
                AddIndividualToOWLClass(doc, real.type, genName);
                AddObjectProperty(doc, "endWith", genName, endElem.getName());
                AddObjectProperty(doc, "startWith", genName, startElem.getName());
            }

            //сохраняем наш документ
            var xml = doc.ToString();
```

```
        xml = xml.Replace("&amp;amp;", "&");
        //doc.ReplaceNodes(xml);
        File.WriteAllText(fileName, xml);
        //doc.Save(fileName);

    }
    private static void AddIndividualToOWLClass(XDocument doc, string owlClassName, string individualName)
    {
        // <Declaration>
        doc.Root.Add(new XElement("Declaration",
                new XElement("NamedIndividual",
                    new XAttribute("IRI", individualName))));
        //<ClassAssertion>
        doc.Root.Add(new XElement("ClassAssertion",
                new XElement("Class",
                    new XAttribute("IRI", owlClassName)),
                new XElement("NamedIndividual",
                     new XAttribute("IRI", individualName))));
    }
    private static void AddObjectProperty(XDocument doc, string owlObjectPropertyName, string individualName, string
propertyValue)
    {

        doc.Root.Add(new XElement("ObjectPropertyAssertion",
                new XElement("ObjectProperty",
                    new XAttribute("IRI", owlObjectPropertyName)),
                new XElement("NamedIndividual",
                        new XAttribute("IRI", individualName)),
                 new XElement("NamedIndividual",
                        new XAttribute("IRI", propertyValue))));

    }
  }
}
```

. Balance – Java

```
JavaSourceExtractor.java
package ontologies;

import com.github.javaparser.JavaParser;
import com.github.javaparser.ast.CompilationUnit;
import com.github.javaparser.ast.Modifier;
import com.github.javaparser.ast.body.ClassOrInterfaceDeclaration;
import com.github.javaparser.ast.body.FieldDeclaration;
import com.github.javaparser.ast.body.MethodDeclaration;
import models.*;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;


/**
 * Created by Gleb on 15.08.2017.
 */
public class JavaSourceExtractor {

    private Map<String, CompilationUnit> units = null;
    private File projectDir = null;

    public JavaSourceExtractor(File projectDir) throws FileNotFoundException
{
        this.projectDir = projectDir;
        units = new HashMap<String, CompilationUnit>();
        units.putAll(extractCompilationUnits(projectDir));
    }
```

```
    private Map<String, CompilationUnit> extractCompilationUnits(File
projectDir) throws FileNotFoundException {
        HashMap<String, CompilationUnit> compilationUnits = new
HashMap<String, CompilationUnit>();
        if (projectDir.isDirectory()) {
            for (File f : projectDir.listFiles()
                    ) {
                if (f.isDirectory()) {
                    compilationUnits.putAll(extractCompilationUnits(f));
                }
                if (f.getPath().endsWith(".java")) {
                    CompilationUnit cu = JavaParser.parse(f);
                    compilationUnits.put(f.getPath(), cu);
                }
            }
        }
        return compilationUnits;
    }

    public CDProject getCDProject(){
        CDProject cdProject = new CDProject();

        String name = projectDir.getName();
        cdProject.setProjectName(name);

        List<CDClass> CDClassList = new ArrayList<CDClass>();
        for (Map.Entry<String, CompilationUnit> entry : units.entrySet()) {
            List<ClassOrInterfaceDeclaration> classesOrInterfaces =
entry.getValue().getChildNodesByType(ClassOrInterfaceDeclaration.class);//get
ChildNodes();
            for (ClassOrInterfaceDeclaration classOrInterface :
classesOrInterfaces
                    ) {

                CDClass cdClass = getClass(classOrInterface);

                //Name
                String className = classOrInterface.getName().asString();
                cdClass.setName(className);




                CDClassList.add(cdClass);
            }

        }
        return cdProject;
    }
    //TODO
    public CDClass getClass(ClassOrInterfaceDeclaration classOrInterface){
        CDClass cdClass = new CDClass();

        //TODO Methods
        List<CDMethod> methodsList = new ArrayList<>();
        for (MethodDeclaration cdMethod: classOrInterface.getMethods()
             ) {
            methodsList.add(getMethod(cdMethod));
        }
        cdClass.setMethodsList(methodsList);

        //TODO Fields
```

```
        List<CDField> fieldList = new ArrayList<>();
        for (FieldDeclaration cdField: classOrInterface.getFields()
                ) {
            fieldList.add(getField(cdField));
        }
        cdClass.setFieldsList(fieldList);

        return cdClass;
    }
    //TODO
    public CDMethod getMethod(MethodDeclaration methodDeclaration){
        CDMethod cdMethod = new CDMethod();

        cdMethod.setName(methodDeclaration.getName().asString());
        methodDeclaration.isPrivate();

        EnumSet<Modifier> modifiers =  methodDeclaration.getModifiers();
        for (Modifier modifer: modifiers
             ) {
            modifer.name();
        }
        return cdMethod;
    }
    //TODO
    public CDField getField(FieldDeclaration fieldDeclaration){
        return new CDField();
    }



    /*private List<UMLClass> getUMLClassList(){
        List<UMLClass> UMLClassList = new ArrayList<UMLClass>();
        for (Map.Entry<String, CompilationUnit> entry : units.entrySet()) {
            List<ClassOrInterfaceDeclaration> classesOrInterfaces =
entry.getValue().getChildNodesByType(ClassOrInterfaceDeclaration.class);//get
ChildNodes();
            for (ClassOrInterfaceDeclaration classOrInterface :
classesOrInterfaces
                    ) {
                UMLClass umlClass = new UMLClass();

                //Name
                String className = classOrInterface.getName().asString();
                umlClass.setName(className);

                //TODO Methods
                List<UMLMethod> methodsList = new ArrayList<>();
                umlClass.setMethodsList(methodsList);

                //TODO Fields
                List<UMLField> fieldList = new ArrayList<>();
                umlClass.setFieldsList(fieldList);

                UMLClassList.add(umlClass);
            }

        }
        return UMLClassList;
    }
    //private UMLClass
    private List<UMLMethod> getUMLMethodsList(ClassOrInterfaceDeclaration
classOrInterface){
        classOrInterface.getMethods();
        return new ArrayList<UMLMethod>();
```

```
    }*/
}

ProjectExtractor.java
package ontologies;

import com.github.javaparser.JavaParser;
import com.github.javaparser.ast.*;
import com.github.javaparser.ast.body.*;
import com.github.javaparser.ast.expr.AnnotationExpr;
import com.github.javaparser.ast.type.ClassOrInterfaceType;
import com.github.javaparser.ast.type.ReferenceType;
import ontologies.algos.SWProject;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;

/**
 * Created by Gleb on 07.08.2017.
 */
public class ProjectExtractor {
    private Map<String, CompilationUnit> units = null;
    private File projectDir = null;

    public ProjectExtractor(){
    }

    public ProjectExtractor(File projectDir) throws FileNotFoundException {
        this.projectDir = projectDir;
        units = new HashMap<String, CompilationUnit>();
        units.putAll(extractCompilationUnits(projectDir));
    }

    public SWProject getProject(File projectDir) throws
FileNotFoundException{
        this.projectDir = projectDir;
        units = new HashMap<String, CompilationUnit>();
        units.putAll(extractCompilationUnits(projectDir));
        SWProject SWProject = new
SWProject(getProjectName(),getCompilationUnits());
        return SWProject;
    }

    private Map<String, CompilationUnit> extractCompilationUnits(File
projectDir) throws FileNotFoundException {
        HashMap<String, CompilationUnit> compilationUnits = new
HashMap<String, CompilationUnit>();
        if (projectDir.isDirectory()) {
            for (File f : projectDir.listFiles()
                    ) {
                if (f.isDirectory()) {
                    compilationUnits.putAll(extractCompilationUnits(f));
                }
                if (f.getPath().endsWith(".java")) {
                    CompilationUnit cu = JavaParser.parse(f);
                    compilationUnits.put(f.getPath(), cu);
                }
            }
        }
        return compilationUnits;
    }

    public Map<String, CompilationUnit> getCompilationUnits(){
```

```
        return units;
    }
    public String getProjectName(){
        return  projectDir.getName();
    }
    public List<ClassOrInterfaceDeclaration> getClassesOrInterfaces(){
        List<ClassOrInterfaceDeclaration> classesOrInterfaces = new
ArrayList<>();
        for (Map.Entry<String, CompilationUnit> entry : units.entrySet()) {
            List<ClassOrInterfaceDeclaration> nodes =
entry.getValue().getChildNodesByType(ClassOrInterfaceDeclaration.class);
            classesOrInterfaces.addAll(nodes);
        }
        return  classesOrInterfaces;
    }

    private ClassOrInterfaceDeclaration getClass(String className) throws
ClassOrInterfaceDeclarationNotFoundException {
        for (Map.Entry<String, CompilationUnit> entry : units.entrySet()) {
            List<Node> nodes = entry.getValue().getChildNodes();
            for (Node node : nodes
                    ) {
                if(node.getClass() == ClassOrInterfaceDeclaration.class) {
                    String _className =((ClassOrInterfaceDeclaration)
node).getName().toString();
                    if (_className.equals(className))
                        return (ClassOrInterfaceDeclaration) node;
                }
            }
        }
        throw new ClassOrInterfaceDeclarationNotFoundException("Class " +
className + " not found");
    }

    public Map<String, List<String>> getInfoAboutClass(String flags) throws
ClassOrInterfaceDeclarationNotFoundException {
        Map<String, List<String>> nodeList = new HashMap<String,
List<String>>();
        String className = flags.split(" ")[1];
        ClassOrInterfaceDeclaration classOrInterfaceDeclaration =
getClass(className);
        List<Node> classNodes = classOrInterfaceDeclaration.getChildNodes();
        ArrayList<String> nodesListForClass = new ArrayList<String>();
        for (Node node : classNodes
                ) {
            if ((flags.contains("-all") || flags.contains("-meth")) &&
node.getClass() == MethodDeclaration.class) {
                MethodDeclaration methodDeclaration = (MethodDeclaration)
node;
                String methodInfo = "Method : ";
                methodInfo +=
getAnnotationSting(methodDeclaration.getAnnotations());
                methodInfo +=
getModifersSting(methodDeclaration.getModifiers());
                methodInfo += methodDeclaration.getName() + " ";
                methodInfo +=
getParametersSting(methodDeclaration.getParameters());
                methodInfo +=
getExceptionString(methodDeclaration.getThrownExceptions());
                nodesListForClass.add(methodInfo);

            }
            if ((flags.contains("-all") || flags.contains("-fld")) &&
node.getClass() == FieldDeclaration.class) {
```

```
                FieldDeclaration fieldDeclaration = (FieldDeclaration) node;
                String fieldInfo = "Field : ";
                fieldInfo +=  fieldDeclaration.toString();
                nodesListForClass.add(fieldInfo);
            }
            if ((flags.contains("-all") || flags.contains("-constr")) &&
node.getClass() == ConstructorDeclaration.class) {
                ConstructorDeclaration constructorDeclaration =
(ConstructorDeclaration) node;
                String constructorInfo = "Constructor : ";
                constructorInfo +=
getAnnotationSting(constructorDeclaration.getAnnotations());
                constructorInfo +=
getModifersSting(constructorDeclaration.getModifiers());
                constructorInfo +=
getParametersSting(constructorDeclaration.getParameters());
                constructorInfo +=
getExceptionString(constructorDeclaration.getThrownExceptions());
                nodesListForClass.add(constructorInfo);
            }
        }
        if (flags.contains("-all") || flags.contains("-impl")) {

nodesListForClass.add(getImplementedNodesString(classOrInterfaceDeclaration.g
etImplementedTypes()));
        }
        if (flags.contains("-all") || flags.contains("-ext") ) {

nodesListForClass.add(getExtendedNodesString(classOrInterfaceDeclaration.getE
xtendedTypes()));
        }
        nodeList.put(classOrInterfaceDeclaration.getName().toString(),
nodesListForClass);
        return nodeList;
    }
    private String getImplementedNodesString(NodeList<ClassOrInterfaceType>
implementList){
        if(implementList.isEmpty() || implementList == null)
            return "";
        String implementedString = "\r\n Implemented : ";
        for (ClassOrInterfaceType implementNode : implementList
                ) {
            implementedString += "\r\n\t" + implementNode.getName();
        }
        return implementedString;
    }
    private String getExtendedNodesString(NodeList<ClassOrInterfaceType>
extendedList){
        if(extendedList.isEmpty() || extendedList == null)
            return "";
        String extendedString = "\r\n Extended : ";
        for (ClassOrInterfaceType extendedNode : extendedList
                ) {
            extendedString += "\r\n\t" + extendedNode.getName();
        }
        return extendedString;
    }
    private String getExceptionString(NodeList<ReferenceType> exceptions){
        String exceptionsString = "";
        if(!exceptions.isEmpty()) {
            exceptionsString += "\r\n\t Exceptions: ";
            for (ReferenceType exception : exceptions
                    ) {
                exceptionsString += "\r\n\t\t" + exception.asString();
```

```
            }
        }
        return exceptionsString;
    }
    private String getParametersSting(NodeList<Parameter> parameters){
        String parametersString = "";

            if (!parameters.isEmpty()) {
                parametersString += "\r\n\t Arguments: ";
                for (Parameter parameter : parameters
                        ) {
                    parametersString += "\r\n\t\t" + " " +
getModifersSting(parameter.getModifiers()) + " " +
parameter.getType().asString() + " " + parameter.getName();
                }

        }
        return parametersString;
    }
    private String getModifersSting(EnumSet<Modifier> modifiers){
        String modiferString = "";
        for (Modifier modifer:modifiers
                ) {
            modiferString += " " + modifer.toString().toLowerCase();
        }
        modiferString += " ";
        return modiferString;
    }

    private String getAnnotationSting(NodeList<AnnotationExpr> annotations){
        String annotationString = "";
        for (AnnotationExpr annotation:annotations
                ) {
            annotationString += "\r\n\t" + annotation.toString();
        }
        annotationString += "\r\n\t";
        return annotationString;
    }


    public Map<String, List<String>> getInfoAboutUnits(String flags) {
        Map<String, List<String>> nodeList = new HashMap<String,
List<String>>();
        for (Map.Entry<String, CompilationUnit> entry : units.entrySet()) {
            List<Node> nodes = entry.getValue().getChildNodes();
            ArrayList<String> nodesListForFile = new ArrayList<String>();
            for (Node node : nodes
                    ) {
                if ((flags.contains("-all") || flags.contains("-cl")) &&
node.getClass() == ClassOrInterfaceDeclaration.class) {
                    nodesListForFile.add("Class or interface" + " : " +
((ClassOrInterfaceDeclaration) node).getName());
                }
                if ((flags.contains("-all") || flags.contains("-imp")) &&
node.getClass() == ImportDeclaration.class) {
                    nodesListForFile.add("Import" + " : " +
((ImportDeclaration) node).getName());
                }
                if ((flags.contains("-all") || flags.contains("-ann")) &&
node.getClass() == AnnotationDeclaration.class) {
                    nodesListForFile.add("Annotation" + " : " +
((AnnotationDeclaration) node).getName());
                }
                if ((flags.contains("-all") || flags.contains("-enum")) &&
```

```
node.getClass() == EnumDeclaration.class) {
                    nodesListForFile.add("Enum" + " : " + ((EnumDeclaration)
node).getName());
                }
            }
            nodeList.put(entry.getKey(), nodesListForFile);
        }
        return nodeList;
    }



    /*
    public Map<String, List<String>> getClassList() throws
FileNotFoundException {

        Map<String, List<String>> classList = new HashMap<String,
List<String>>();
        for (Map.Entry<String, CompilationUnit> entry : units.entrySet()) {
            List<ClassOrInterfaceDeclaration> classOrInterfaceNodes =
entry.getValue().getChildNodesByType(ClassOrInterfaceDeclaration.class);
            ArrayList<String> classListForFIle = new ArrayList<String>();
            for (ClassOrInterfaceDeclaration classOrInterfaceNode :
classOrInterfaceNodes
                    ) {
                classListForFIle.add(
classOrInterfaceNode.getName().toString());
            }
            classList.put( entry.getKey(),classListForFIle);
        }
        return classList;
    }
    public Map<String, List<String>> getImportList() throws
FileNotFoundException {

        Map<String, List<String>> classList = new HashMap<String,
List<String>>();
        for (Map.Entry<String, CompilationUnit> entry : units.entrySet()) {
            List<ImportDeclaration> importDeclarationNodes =
entry.getValue().getChildNodesByType(ImportDeclaration.class);
            ArrayList<String> importListForFIle = new ArrayList<String>();
            for (ImportDeclaration importDeclaration : importDeclarationNodes
                    ) {
                importListForFIle.add(
importDeclaration.getName().toString());
            }
            classList.put( entry.getKey(), importListForFIle);
        }
        return classList;
    }
*/
}
```

OwlWorker.java

```
package ontologies.owl;

import com.github.javaparser.ast.CompilationUnit;
import com.github.javaparser.ast.ImportDeclaration;
import com.github.javaparser.ast.PackageDeclaration;
import com.github.javaparser.ast.body.ClassOrInterfaceDeclaration;
import com.github.javaparser.ast.body.FieldDeclaration;
import com.github.javaparser.ast.body.MethodDeclaration;
import com.github.javaparser.ast.body.Parameter;
import com.github.javaparser.ast.type.ClassOrInterfaceType;
```

```
import ontologies.algos.SWProject;
import org.semanticweb.owlapi.apibinding.OWLManager;
import org.semanticweb.owlapi.model.*;
import org.semanticweb.owlapi.util.DefaultPrefixManager;
import ontologies.algos.CustomOWLIndividual;
import ontologies.algos.CustomObjectProperty;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.*;
import java.util.stream.Collectors;


/**
 * Created by Gleb on 10.08.2017.
 */
public class OwlWorker {
    private OWLOntology ontology = null;
    private PrefixManager pm;
    private String ontologyFileName = "";
    private OWLDataFactory factory = null;
    private OWLOntologyManager manager;
    private Map<String, CompilationUnit> units = null;
    private String projectName = null;

    public OwlWorker(String ontologyFileName){
        this.ontologyFileName = ontologyFileName;
        manager = OWLManager.createOWLOntologyManager();
        try {
            ontology = manager.loadOntologyFromOntologyDocument(new
File(ontologyFileName));
            pm = new DefaultPrefixManager(null, null, "");

            factory = manager.getOWLDataFactory();
        } catch (OWLOntologyCreationException e) {
            e.printStackTrace();
            //throw new Exception("Ontology file is not exist");
        }
    }

    public SWProject getByOWLIndividuals(SWProject p) {

p.structuredElementsAndRelations.addAll(getIndividualsByClass("SimpleClass"))
;

p.structuredElementsAndRelations.addAll(getIndividualsByClass("AbstractClass"
));

p.structuredElementsAndRelations.addAll(getIndividualsByClass("Interface"));

p.structuredElementsAndRelations.addAll(getIndividualsByClass("Association"))
;

p.structuredElementsAndRelations.addAll(getIndividualsByClass("Dependency"));

p.structuredElementsAndRelations.addAll(getIndividualsByClass("Generalization
"));

p.structuredElementsAndRelations.addAll(getIndividualsByClass("Realization"))
;

p.structuredElementsAndRelations.addAll(getIndividualsByClass("Method"));
```

```
p.structuredElementsAndRelations.addAll(getIndividualsByClass("Field"));
        p.structuredElementsAndRelations =
fillObjectProperties(p.getStructuredElements());
        return p;
    }

    public List<CustomOWLIndividual> getIndividualsByClass(String className){
        OWLClass owlClass = ontology.classesInSignature().filter(item ->
(item.getIRI().toString().contains(className))).findFirst().orElse(null);
        List<OWLIndividual>  individuals =
ontology.axioms(AxiomType.CLASS_ASSERTION)
                .filter(owlClassAssertionAxiom ->
owlClassAssertionAxiom.getClassExpression().asOWLClass().getIRI().getRemainde
r().orElseThrow(() -> new
IllegalArgumentException("Remainder")).equals(className))
                .map(owlClassAssertionAxiom ->
owlClassAssertionAxiom.getIndividual().asOWLNamedIndividual())
                .collect(Collectors.toList());
        List<CustomOWLIndividual> res = new ArrayList<CustomOWLIndividual>();
        for (OWLIndividual individual: individuals
             ) {
            CustomOWLIndividual resElem = new CustomOWLIndividual(individual,
owlClass);
            res.add(resElem);
        }
        return res;
    }
    public List<CustomOWLIndividual>
fillObjectProperties(List<CustomOWLIndividual> individulas){
        for (CustomOWLIndividual resElem: individulas
                ) {
           // System.out.println(resElem.getOwlIndividual().toString());
            resElem.objectProperties =
ontology.axioms(AxiomType.OBJECT_PROPERTY_ASSERTION)
                    .filter(owlObjectPropertyAssertionAxiom ->
owlObjectPropertyAssertionAxiom.getSubject().toStringID().equals(resElem.getO
wlIndividual().toStringID()))
                    .map(owlObjectPropertyAssertionAxiom -> new
CustomObjectProperty(owlObjectPropertyAssertionAxiom.getProperty(),
                            individulas.stream().filter(ind ->
(ind.getOwlIndividual().compareTo(owlObjectPropertyAssertionAxiom.getObject()
)) == 0).findFirst().orElse(null)))
                    .collect(Collectors.toList());
            resElem.objectProperties.remove(null);
        }

        return individulas;
    }

    public List<OWLObjectProperty> getObjectProperties(OWLIndividual ind){
        return
ind.objectPropertiesInSignature().collect(Collectors.toList());
    }
   /* public OWLIndividual getEndIndividualByObjectProperty(OWLObjectProperty
owlObjectProperty){
        return owlObjectProperty.
    }
*/
    public void addProject(SWProject SWProject, String fileDirectory) {
        this.units = SWProject.compilationUnitMap;
        this.projectName = SWProject.getProjectName();
        //if(File)
        String projectOntologyPath = fileDirectory + projectName + ".owl";
```

```
        for (Map.Entry<String, CompilationUnit> entry : units.entrySet()) {
            CompilationUnit unit = entry.getValue();

            // Add Package
            PackageDeclaration packageDeclaration =
unit.getPackageDeclaration().orElse(null);
            if (packageDeclaration == null)
                continue;
            OWLClass owlPackageClass = factory.getOWLClass(":Package", pm);
            OWLNamedIndividual packageInd = factory.getOWLNamedIndividual(":"
+ packageDeclaration.getName(), pm);
            OWLClassAssertionAxiom addPackageAx =
factory.getOWLClassAssertionAxiom(owlPackageClass, packageInd);
            manager.addAxiom(ontology, addPackageAx);

            //AddClasses
            List<ClassOrInterfaceDeclaration> classList =
unit.getChildNodesByType(ClassOrInterfaceDeclaration.class);
            for (ClassOrInterfaceDeclaration _class : classList
                    ) {
                OWLClass owlSimpleClassCl =
factory.getOWLClass(":SimpleClass", pm);
                OWLClass owlAbstractClassCl =
factory.getOWLClass(":AbstractClass", pm);
                OWLClass owlInterfaceCl = factory.getOWLClass(":Interface",
pm);
                OWLNamedIndividual simpleClassInd =
factory.getOWLNamedIndividual(":" + _class.getName(), pm);
                System.out.println(_class.getName());
                OWLClassAssertionAxiom addClassAx = null;
                OWLObjectPropertyExpression objProp = null;
                if (_class.isInterface()) {
                    addClassAx =
factory.getOWLClassAssertionAxiom(owlInterfaceCl, simpleClassInd);
                    objProp =
factory.getOWLObjectProperty(":IncludeInterface", pm);
                } else if (_class.isAbstract()) {
                    addClassAx =
factory.getOWLClassAssertionAxiom(owlAbstractClassCl, simpleClassInd);
                    objProp =
factory.getOWLObjectProperty(":IncludeAbstractClass", pm);
                } else {
                    addClassAx =
factory.getOWLClassAssertionAxiom(owlSimpleClassCl, simpleClassInd);
                    objProp =
factory.getOWLObjectProperty(":IncludeSimpleClass", pm);
                }
                manager.addAxiom(ontology, addClassAx);

                //Add methods
                addClassMethodsAndDependensies(_class, simpleClassInd);
                //Add fields
                addClassFieldsAndAssociations(_class, simpleClassInd);
                //Add imports
                addClassImports(unit, _class, simpleClassInd);
                //Add StructuredThing to package
                OWLObjectPropertyAssertionAxiom objPropAx =
factory.getOWLObjectPropertyAssertionAxiom(objProp, packageInd,
simpleClassInd);
                manager.addAxiom(ontology, objPropAx);
                //Add generalization
                addClassExtends(_class);
                //Add implementation
                addImplementedInterfaces(_class);
```

```
            }
            try {
                File file = new File(projectOntologyPath);
                file.createNewFile();
                FileOutputStream fileOutputStream = new
FileOutputStream(projectOntologyPath);
                manager.saveOntology(ontology,fileOutputStream);
            } catch (OWLOntologyStorageException e) {
                e.printStackTrace();
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
            System.out.println(classList.size() + " classes were added!");
        }
        System.out.println("SWProject was added!");
    }

    public Map<String, CompilationUnit> getProjectFromOntology(String
projectName){

        return new HashMap<String, CompilationUnit>();
    }

    private void addClassMethodsAndDependensies(ClassOrInterfaceDeclaration
_class, OWLNamedIndividual simpleClassInd) {
        if (_class.getMethods().isEmpty())
            return;
        for (MethodDeclaration method : _class.getMethods()
                ) {
            OWLClass owlMethodClass = factory.getOWLClass(":Method", pm);
            OWLNamedIndividual methodInd = factory.getOWLNamedIndividual(":"
+ _class.getName() + '_' + method.getName(), pm);
            OWLClassAssertionAxiom addMethodAx =
factory.getOWLClassAssertionAxiom(owlMethodClass, methodInd);
            manager.addAxiom(ontology, addMethodAx);

            //ObjectProperty isPartOfClass
            OWLObjectPropertyExpression objProp =
factory.getOWLObjectProperty(":isPartOfClass", pm);
            OWLObjectPropertyAssertionAxiom objPropAx =
factory.getOWLObjectPropertyAssertionAxiom(objProp, methodInd,
simpleClassInd);
            manager.addAxiom(ontology, objPropAx);

            //DataProperty hasName
            OWLDataPropertyExpression dataPropHasProperty =
factory.getOWLDataProperty(":hasName", pm);
            OWLDataPropertyAssertionAxiom dataPropHasNameAx =
factory.getOWLDataPropertyAssertionAxiom(dataPropHasProperty, methodInd,
method.getName().toString());
            manager.addAxiom(ontology, dataPropHasNameAx);
            addClassDependensies(_class, method);
        }
        /*try {
            manager.saveOntology(ontology);
        } catch (OWLOntologyStorageException e) {
            e.printStackTrace();
        }*/
        System.out.println(_class.getMethods().size() + " methods were added
for class " + _class.getName() + "!");
    }
```

```
    private void addClassFieldsAndAssociations(ClassOrInterfaceDeclaration
_class, OWLNamedIndividual simpleClassInd) {
        if (_class.getFields().isEmpty())
            return;
        for (FieldDeclaration field : _class.getFields()
                ) {
            OWLClass owlMethodClass = factory.getOWLClass(":Field", pm);
            OWLNamedIndividual fieldInd = factory.getOWLNamedIndividual(":" +
_class.getName() + '_' + field.getVariable(0).getName(), pm);
            System.out.println(field.toString());
            System.out.println(field.getCommonType().toString());
            addClassAssociation(_class, field);

            OWLClassAssertionAxiom addMethodAx =
factory.getOWLClassAssertionAxiom(owlMethodClass, fieldInd);
            manager.addAxiom(ontology, addMethodAx);

            //ObjectProperty isPartOfClass
            OWLObjectPropertyExpression objProp =
factory.getOWLObjectProperty(":isPartOfClass", pm);
            OWLObjectPropertyAssertionAxiom objPropAx =
factory.getOWLObjectPropertyAssertionAxiom(objProp, fieldInd,
simpleClassInd);
            manager.addAxiom(ontology, objPropAx);

            //DataProperty hasName
            OWLDataPropertyExpression dataPropHasProperty =
factory.getOWLDataProperty(":hasName", pm);
            OWLDataPropertyAssertionAxiom dataPropHasNameAx =
factory.getOWLDataPropertyAssertionAxiom(dataPropHasProperty, fieldInd,
field.getVariable(0).getName().toString());
            manager.addAxiom(ontology, dataPropHasNameAx);

        }
       /* try {
            manager.saveOntology(ontology);
        } catch (OWLOntologyStorageException e) {
            e.printStackTrace();
        }*/
        System.out.println(_class.getMethods().size() + " feilds were added
for class " + _class.getName() + "!");
    }

    private void addClassImports(CompilationUnit unit,
ClassOrInterfaceDeclaration _class, OWLNamedIndividual simpleClassInd) {

        if (unit.getImports().isEmpty())
            return;
        for (ImportDeclaration _import : unit.getImports()
                ) {
            OWLClass owlImportClass = factory.getOWLClass(":Import", pm);
            OWLNamedIndividual importInd = factory.getOWLNamedIndividual(":"
+ _class.getName() + '_' + _import.getName(), pm);
            OWLClassAssertionAxiom addMethodAx =
factory.getOWLClassAssertionAxiom(owlImportClass, importInd);
            manager.addAxiom(ontology, addMethodAx);

            //ObjectProperty isPartOfClass
            OWLObjectPropertyExpression objProp =
factory.getOWLObjectProperty(":isPartOfClass", pm);
            OWLObjectPropertyAssertionAxiom objPropAx =
factory.getOWLObjectPropertyAssertionAxiom(objProp, importInd,
simpleClassInd);
            manager.addAxiom(ontology, objPropAx);
```

```
            //DataProperty hasName
            OWLDataPropertyExpression dataPropHasProperty =
factory.getOWLDataProperty(":hasName", pm);
            OWLDataPropertyAssertionAxiom dataPropHasNameAx =
factory.getOWLDataPropertyAssertionAxiom(dataPropHasProperty, importInd,
_import.getName().toString());
            manager.addAxiom(ontology, dataPropHasNameAx);

        }
       /* try {
            manager.saveOntology(ontology);
        } catch (OWLOntologyStorageException e) {
            e.printStackTrace();
        }*/
        System.out.println(_class.getMethods().size() + " feilds were added
for class " + _class.getName() + "!");
    }

    private void addClassExtends(ClassOrInterfaceDeclaration _class) {
        if (_class.getExtendedTypes().isEmpty())
            return;
        for (ClassOrInterfaceType parent : _class.getExtendedTypes()
                ) {
            OWLNamedIndividual parentInd = factory.getOWLNamedIndividual(":"
+ parent.getName(), pm);
            OWLNamedIndividual classInd = factory.getOWLNamedIndividual(":"
+ _class.getName(), pm);

            OWLClass genClass = factory.getOWLClass(":Generalization", pm);
            OWLNamedIndividual genInd = factory.getOWLNamedIndividual(":"  +
_class.getName() + '_' + parent.getName(), pm);
            OWLClassAssertionAxiom addGenAx =
factory.getOWLClassAssertionAxiom(genClass, genInd);
            manager.addAxiom(ontology, addGenAx);

            addObjectProperty(":endWith", genInd, parentInd);

            addObjectProperty(":startWith", genInd, classInd);

            addObjectProperty(":startOfRelationship", classInd, genInd);

            addObjectProperty(":endOfRelationship", parentInd, genInd);

        }
       /* try {
            manager.saveOntology(ontology);
        } catch (OWLOntologyStorageException e) {
            e.printStackTrace();
        }*/
        System.out.println(_class.getMethods().size() + " extendds were added
for class " + _class.getName() + "!");
    }

    private void addImplementedInterfaces(ClassOrInterfaceDeclaration _class)
{
        if (_class.getImplementedTypes().isEmpty())
            return;
        for (ClassOrInterfaceType parent : _class.getImplementedTypes()
                ) {
            OWLNamedIndividual parentInd = factory.getOWLNamedIndividual(":"
+ parent.getName(), pm);
            OWLNamedIndividual classInd = factory.getOWLNamedIndividual(":"
+ _class.getName(), pm);
```

```
            OWLClass genClass = factory.getOWLClass(":Realization", pm);
            OWLNamedIndividual implInd = factory.getOWLNamedIndividual(":"  +
_class.getName() + '_' + parent.getName(), pm);
            OWLClassAssertionAxiom addGenAx =
factory.getOWLClassAssertionAxiom(genClass, implInd);
            manager.addAxiom(ontology, addGenAx);


            addObjectProperty(":endWith", implInd, parentInd);

            addObjectProperty(":startWith", implInd, classInd);

            addObjectProperty(":startOfRelationship", classInd, implInd);

            addObjectProperty(":endOfRelationship", parentInd, implInd);


        }
       /* try {
            manager.saveOntology(ontology);
        } catch (OWLOntologyStorageException e) {
            e.printStackTrace();
        }*/
        System.out.println(_class.getMethods().size() + " implementations
were added for class " + _class.getName() + "!");
    }

    private void addClassDependensies(ClassOrInterfaceDeclaration _class,
MethodDeclaration method) {
        for (Map.Entry<String, CompilationUnit> entry : units.entrySet()) {
            CompilationUnit unit = entry.getValue();
            List<ClassOrInterfaceDeclaration> classList =
unit.getChildNodesByType(ClassOrInterfaceDeclaration.class);
            for (ClassOrInterfaceDeclaration _associatedClass : classList
                    ) {
                for (Parameter param :method.getParameters()
                     ) {
                    if
(_associatedClass.getName().asString().contains(param.getType().asString()))
{
                        OWLNamedIndividual dependedClassInd =
factory.getOWLNamedIndividual(":" + _associatedClass.getName(), pm);
                        OWLNamedIndividual classInd =
factory.getOWLNamedIndividual(":" + _class.getName(), pm);

                        OWLClass genClass =
factory.getOWLClass(":Dependency", pm);
                        OWLNamedIndividual depInd =
factory.getOWLNamedIndividual(":"  + _class.getName() + '_' +
_associatedClass.getName(), pm);
                        OWLClassAssertionAxiom addGenAx =
factory.getOWLClassAssertionAxiom(genClass, depInd);
                        manager.addAxiom(ontology, addGenAx);

                        addObjectProperty(":endWith", depInd,
dependedClassInd);

                        addObjectProperty(":startWith", depInd, classInd);

                        addObjectProperty(":startOfRelationship", classInd,
depInd);

                        addObjectProperty(":endOfRelationship",
```

```
dependedClassInd, depInd);

                        return;
                    }
                }
            }
        }
    }

    private void addClassAssociation(ClassOrInterfaceDeclaration _class,
FieldDeclaration field) {
        for (Map.Entry<String, CompilationUnit> entry : units.entrySet()) {
            CompilationUnit unit = entry.getValue();
            List<ClassOrInterfaceDeclaration> classList =
unit.getChildNodesByType(ClassOrInterfaceDeclaration.class);
            for (ClassOrInterfaceDeclaration _associatedClass : classList
                    ) {
                if
(_associatedClass.getName().asString().contains(field.getCommonType().asStrin
g())) {
                    OWLNamedIndividual associatedClassInd =
factory.getOWLNamedIndividual(":"  + _associatedClass.getName(), pm);
                    OWLNamedIndividual classInd =
factory.getOWLNamedIndividual(":"  + _class.getName(), pm);

                    // Объявление индивидуала ассоциации
                    OWLClass genClass = factory.getOWLClass(":Association",
pm);
                    OWLNamedIndividual assInd =
factory.getOWLNamedIndividual(":" + _class.getName() + '_' +
_associatedClass.getName(), pm);
                    OWLClassAssertionAxiom addGenAx =
factory.getOWLClassAssertionAxiom(genClass, assInd);
                    manager.addAxiom(ontology, addGenAx);

                    //  Объявление элемента-окончания ассоциации
                    addObjectProperty(":endWith", assInd,
associatedClassInd);

                    //Объявление элемента-начала ассоциации
                    addObjectProperty(":startWith", assInd, classInd);

                    //Объявление исходящей ассоциации
                    addObjectProperty(":startOfRelationship", classInd,
assInd);

                    //Объявление входящей ассоциации
                    addObjectProperty(":endOfRelationship",
associatedClassInd, assInd);

                    return;
                }
            }
        }
    }

    private void addObjectProperty(String name, OWLNamedIndividual domain,
OWLNamedIndividual range){
        OWLObjectPropertyExpression objProp =
factory.getOWLObjectProperty(name, pm);
        OWLObjectPropertyAssertionAxiom objPropAx =
factory.getOWLObjectPropertyAssertionAxiom(objProp, domain, range );
        manager.addAxiom(ontology, objPropAx);
    }
```

```
    public void testOWL2API() {
        if (ontology == null) {
            System.out.println("No ontology");
            return;
        } else {
            System.out.println(ontology.getOntologyID().toString());

System.out.println(ontology.getOntologyID().getOntologyIRI().get().toString()
);
        }

        //getAllClasses
        Set<OWLClass> classesSet =
ontology.classesInSignature().collect(Collectors.toSet());
        for (OWLClass owlClass : classesSet
                ) {
            System.out.println(owlClass.getIRI());
        }

    }
}
```

Приложения 3. Результаты вычислительных экспериментов

Временные ряды NN3

101	102	103	104	105	106	107	108	109	110	111
4998	5033	5304	685	5007	4950	4318	1740	6318	2605	3140
4480	4510	4264	656	5165	4900	4224	1005	5991	2510	3270
4824	3970	3224	1288	4976	5500	4014	1793	6405	3520	3690
4814	3378	2400	3914	4984	5550	3620	1965	6230	3200	4360
4602	2866	1968	6837	5105	5900	3512	2085	6111	2550	3520
4499	2316	1532	7104	5026	5150	3590	2440	5317	2640	4140
4594	1895	1000	7464	4848	5700	3700	2318	4893	2280	3440
4600	8402	55024	6308	4963	6450	3752	2313	4867	2620	3750
4507	8040	53804	5549	4796	4800	3866	2668	4785	1940	4140
4606	7534	42540	6365	5042	6300	3892	3655	5099	1415	5610
4503	7136	26572	6344	5028	5700	3838	2245	5142	1775	5350
4801	6467	15748	5597	4668	4900	3936	3000	5335	2320	4270
4564	5662	10372	1178	4646	4900	4200	2795	5450	2585	3160
4142	4896	8980	1581	4912	5500	4184	3493	5161	2885	2560
4818	4065	7212	1870	5041	5550	3708	3098	5516	2870	3410
4408	3296	7504	5230	5101	5200	3708	3665	4850	3010	3670
4496	2594	3848	8046	5143	5500	3760	6555	4250	2710	3670
4587	2007	2684	7997	4998	4950	3708	3040	4451	3070	3200
4656	1514	1000	9216	5053	5900	3846	8878	5442	2050	3700
4799	6645	43560	8077	5024	5700	3748	3280	5426	2315	3430
4652	6222	40320	6021	5299	5300	3836	3203	5329	2140	4130
4638	5474	28096	6399	4947	5550	3790	4593	5150	1950	4870
4650	5136	17804	6857	4872	4600	3968	2600	5030	3150	6310
5185	4631	10008	3446	4563	5650	3756	4803	4833	4510	4680
5208	4164	6136	1544	4580	4950	3760	3685	4437	3685	2380
4477	3601	4216	2031	4909	4850	3986	3688	4334	5125	3430
4976	2969	2424	2561	5002	5050	4096	2615	4165	5750	2950
4670	2504	1840	5166	5127	5500	4194	3070	3704	4580	3130

4842	2055	1112	7470	5060	5300	4254	2598	3616	4555	3650
4713	1609	324	6849	4947	5250	4130	2618	3476	3430	2980
4804	1298	100	6968	4810	6100	4044	2500	3118	2700	3720
4996	8485	58676	7253	4589	5200	3766	2685	3056	2010	2960
4574	8164	54768	6413	4408	6400	3930	2500	2922	1965	4300
4841	7814	43624	5854	4558	5700	3910	3693	2755	1185	5550
4688	7454	28680	5710	4509	5000	4024	1178	2691	2180	4380
4766	6889	15008	4926	4331	5400	4158	2923	2586	2705	3620
4994	6284	9028	1300	4770	4250	4018	2788	2810	3345	3350
4514	5712	7240	2200	4926	4950	3836	2090	2740	11760	3300
4766	5030	4884	2782	4679	5750	3764	2735	2565	3665	3800
4642	4488	3812	4708	4775	4600	3684	2855	3204	2490	3420
4806	4059	3696	6792	4707	4450	3668	3225	3149	2290	3860
4645	3585	2912	6193	4609	5400	3770	2218	2732	1600	2900
4784	3200	1200	5647	4644	4600	3888	2060	3267	1320	3240
4979	8181	42468	4317	4641	5400	3808	1965	3250	4950	3150
4530	8220	41900	4628	4436	4850	3794	1755	3194	1305	4460
4942	7866	30180	4658	4574	4100	3646	2465	3304	1310	5040
4651	7517	16836	4552	4503	4650	3712	2433	3332	2390	4660
5150	7116	9732	4349	4292	4700	3848	2465	3458	3685	3450
4987	6616	5728	1642	4479	3900	4064	2385	3787	3265	2660
4532	6217	5728	1069	3930	4200	4156	2458	3588	3770	2400
5046	5700	3584	2244	3860	5050	3724	2038	3839	4365	2200
4783	5179	3068	4551	4065	4250	3520	2538	4177	3380	3020
4958	4728	2992	5242	3741	4900	3796	2165	4097	2420	3450
4815	4225	1092	6225	3651	5250	3956	4390	3561	2490	4110
5055	3781	600	6175	3653	4250	3840	3045	3636	1980	2760
5152	7024	28268	6090	3532	5700	3794	2283	3618	1710	3390
4773	6558	26652	6431	3607	4950	3970	2838	3555	1755	3740
5147	6258	19068	6769	3778	4200	3740	3463	3315	1465	4830
4866	5863	10024	6393	4019	4450	3860	2410	3344	1910	4450
5311	5343	4636	6704	3884	4850	3928	3358	3470	1885	3140
5172	4756	4888	4632	3850	5600	4260	3678	3461	2190	4700
4734	4174	5100	2757	4042	5150	4180	2383	3280	1840	2330
5011	3452	4664	2999	4216	5650	4144	3128	3508	2435	2740
4957	2849	3848	6150	4117	4850	4058	2885	3849	3060	2480
4968	2351	3772	6780	4232	6000	3954	4093	3776	2320	2690
5049	1888	2964	6785	4212	5200	3890	7223	3281	4155	5850
5305	1417	1200	7229	4066	5400	3816	3398	3434	2045	2450
5067	7399	50368	7243	4148	5500	3800	3900	3417	1915	3720
5001	7013	47796	6907	4060	4700	3868	3085	3358	1610	2780
5252	6645	38376	6673	4192	4900	3682	2343	3163	1305	5690
4903	6239	17728	6348	4249	4550	3804	4533	3190	2205	4150
5408	5721	12832	7166	3991	4500	3822	3503	3311	3160	4460
5395	5138	7768	4162	4101	4950	4062	3193	3505	3315	2730
5150	4357	5192	3295	4216	4550	4102	3350	3322	3595	3580
5460	3751	3896	3903	4302	5350	3928	3495	3553	4480	2870
4968	3324	4128	5996	4253	4900	3842	4488	3599	3615	4210
5021	2861	3716	7331	4269	5050	3816	2868	3531	3035	4500
5118	2456	2500	6850	4193	5150	3534	4093	3069	2110	3560
5175	2028	1700	7178	4102	5300	3624	2878	2589	1825	3800
5420	8389	5304	7342	4250	5350	3708	3543	2576	2260	3260

5121	7910	45856	6699	4134	4950	3692	5508	2532	1320	3410
5450	7686	34592	6642	4288	5150	3646	3005	2519	1460	3860
5286	7163	19172	6176	4384	4800	3716	2775	2541	2420	5000
5693	6842	9496	5580	4193	4700	3840	2388	2637	3145	3840
5353	6449	5960	3401	4057	4450	4154	3008	2641	3965	3210
5017	6061	4500	2508	4235	4250	4118	3913	2502	3830	2930
5577	5739	7560	3128	4420	4900	3824	3383	2677	3125	3630
4987	5363	3608	5936	4462	4850	3804	4130	3294	2720	2760
5129	5081	2948	7095	4464	4850	3706	3433	3232	2130	3310
5249	4764	3148	7490	4330	5300	3602	3833	2809	1765	4320
5100	4523	600	8205	4413	5600	3842	4375	3263	1260	3010
5382	9057	4196	8163	4323	6150	3676	2413	3246	965	3440
5039	8352	32476	7080	4215	5500	3750	2760	3190	935	5100
5364	7683	21144	7521	4453	5550	3696	4285	3295	1035	4170
5193	7320	15652	6923	4484	4300	3742	3305	3323	1115	4540
5846	6708	10160	6347	4230	4800	3582	5885	3449	935	3410
5259	6205	7308	3169	4229	4650	3966	3490	3462	1980	2590
4809	5577	5568	1802	4363	4500	4060	2945	3294	900	2640
5297	4777	3724	2574	4442	4650	3748	2595	3592	880	2970
5034	4280	4064	5016	4452	4800	3720	2863	3505	815	3000
5243	3918	5024	6280	4455	5000	3604	2170	3682	380	2900
5150	3289	4388	6520	4355	5200	3548	2098	3350	275	3010
5296	2394	1200	6658	4409	5500	3626	2383	3601	105	3380
5596	8132	49504	6513	4387	5500	3484	5468	3414	1305	3060
4954	8121	43196	6964	4350	5950	3792	3538	3457	1730	3100
5250	7791	30608	6508	4448	5500	3638	3440	3545	1570	3940
5009	7412	19300	5969	4492	4800	3574	3458	3399	2645	5360
5113	6861	10100	6134	4205	5300	3648	2963	3564	15745	3280
5237	6197	5880	2851	4370	5000	3750	2943	3568	2145	3190
4575	5623	5248	2122	4489	4950	3610	2833	3295	2495	4950
5026	4856	3812	2791	4516	5250	3462	2970	3722	2975	2910
4842	4304	3164	5516	4537	4900	3418	3075	3613	2645	2820
5019	3854	3872	6887	4441	5300	3478	2490	3525	2560	3050
5063	3284	3044	7446	4447	5750	3574	3298	3368	2110	2930
5261	2862	1200	7158	4477	5400	3466	2538	3527	2000	2800
5327	9487	55896		4368	5750	3468		3336	1975	3220
5054	9061	49948		4396	5100	3586		3320	1475	3120
5269	8878	35900		4528	4550	3366		3599	1730	4000
5019	8558	19352		4554	4650	3500		3301	2900	4840
5315	8031	7828		4316	5000	3666		3195	3405	2630
5274	7405	4324		4482	4350	3858		3354	3130	2920
4899	6853	4104		4719	4250	3842		3025	3490	2350
5216	6175	3972		4855	5200	3668		3369	3675	2850
5029	5342	2828		4820	4600	3700			2705	2700
5110	4976	4304		4704	4800	3748			2440	2870
5093	4290	4876		4608	5100	3790			2615	2580

Прогнозы различными методами временных рядов NN3

Описание методов:

№ метода	Описание
0	NoTrendNoSeasonality
1	NoTrendAddSeasonality
2	NoTrendMultSeasonality
3	AddTrendNoSeasonality
4	MultTrendNoSeasonality
5	DATrendNoSeasonality
6	DMTrendNoSeasonality
7	AddTrendAddSeasonality
8	MultTrendAddSeasonality
9	DATrendAddSeasonality
10	DMTrendAddSeasonality
11	AddTrendMultSeasonality
12	MultTrendMultSeasonality
13	DATrendMultSeasonality
14	DMTrendMultSeasonality
15	Dsa
16	Fuzzy (NoTrend, NoSeasonality)
17	Fuzzy (AddTrend, NoSeasonality)
18	Fuzzy (NoTrend, AddSeasonality)
19	Fuzzy (AddTrend, AddSeasonality)
20	FuzzyWithSets (NoTrend, NoSeasonality)
21	FuzzyWithSets (NoTrend, AddSeasonality)
22	FuzzyWithSets (NoTrend, MultSeasonality)
23	FuzzyWithSets (AddTrend, NoSeasonality)
24	FuzzyWithSets (AddTrend, AddSeasonality)
25	FuzzyWithSets (AddTrend, MultSeasonality)
26	FuzzyWithSets (MultTrend, NoSeasonality)
27	FuzzyWithSets (MultTrend, AddSeasonality)
28	FuzzyWithSets (MultTrend, MultSeasonality)

№ метода	SMAPE	Alpha	Delta	Gamma	Phi	countRulesIn	countFuzzyParts	c
0	10.4639	0.1	0	0	0	0	0	0
1	4.38708	0.1	0.18	0	0	0	0	0
2	4.57033	0.1	0.35	0	0	0	0	0
3	4.69548	0.1	0	0.21	0	0	0	0
4	6.69968	0.1	0	0.18	0	0	0	0
5	3.81202	0.1	0	0.1	0.1	0	0	0
6	4.00759	0.1	0	0.1	0.1	0	0	0
7	183.351	0.11	0	0	0	0	0	0
8	3.7876	0.12	0.28	0.95	0	0	0	0
9	2.92482	0.1	0.15	0.3	0.1	0	0	0

10	2.74167	0.1	0.2	0.3	0.1	0	0	0
11	3.00145	0.1	0.16	0.13	0	0	0	0
12	3.60987	0.12	0.27	0.97	0	0	0	0
13	5.01282	0.15	0.25	0.6	0.1	0	0	0
14	4.42839	0.1	0.25	0.1	0.1	0	0	0
15	13.2832	0	0	0	0	0	0	1
16	13.4746	0	0	0	0	1	3	0
17	13.2832	0	0	0	0	1	8	0
18	4.35939	0	0	0.3	0	1	2	0
19	7.22012	0	0.9	0	0	1	2	0
20	3.66653	0	0	0	0	0	2	0
21	4.63172	0	0.3	0	0	0	2	0
22	13.2832	0	0	0	0	0	2	0
23	71.0155	0.3	0	0.3	0	0	2	0
24	56.2016	0.3	0.3	0.3	0	0	2	0
25	13.2832	0	0	0	0	0	2	0
26	60.3137	0.6	0	0	0	0	2	0
27	4.03937	0.3	0.6	0	0	0	2	0
28	13.2832	0	0	0	0	0	2	0

Результаты прогнозирования методами ВР 101

№ метода	SMAPE	Alpha	Delta	Gamma	Phi	countRulesIn	countFuzzyParts	C
0	30.6856	0.1	0	0	0	0	0	0
1	13.5111	0.25	0.98	0	0	0	0	0
2	16.2944	0.1	0.61	0	0	0	0	0
3	30.8288	0.56	0	0.16	0	0	0	0
4	31.7044	0.12	0	0.14	0	0	0	0
5	30.4004	0.1	0	0.1	0.1	0	0	0
6	30.3914	0.1	0	0.7	0.9	0	0	0
7	182.293	0.12	0	0	0	0	0	0
8	14.5079	0.37	0.93	0.1	0	0	0	0
9	16.3956	0.4	0.9	0.1	0.1	0	0	0
10	17.341	0.2	0.8	0.1	0.1	0	0	0
11	22.1252	0.19	0.69	0.1	0	0	0	0
12	18.1142	0.29	0.8	0.1	0	0	0	0
13	20.1469	0.2	0.75	0.1	0.1	0	0	0
14	21.1774	0.15	0.7	0.1	0.1	0	0	0
15	31.9753	0	0	0	0	0	0	2
16	31.2597	0	0	0	0	1	2	0
17	45.2025	0	0	0	0	1	7	0
18	36.855	0	0	0.9	0	1	2	0
19	32.2132	0	0.9	0.9	0	1	2	0
20	31.8721	0	0	0	0	0	2	0
21	21.5078	0.6	0.3	0	0	0	2	0
22	22.1922	0	0.6	0	0	0	2	0
23	261.888	0	0	0	0	0	2	0
24	104.743	0.3	0.3	0.3	0	0	2	0

25	339.748	0	0	0	0	0	2	0
26	199.783	0	0	0	0	0	2	0
27	31.3362	0.3	0	0.3	0	0	2	0
28	200.025	0	0	0	0	0	2	0

Результаты прогнозирования методами ВР 102

№ метода	**SMAPE**	**Alpha**	**Delta**	**Gamma**	**Phi**	**countRulesIn**	**countFuzzyParts**	**C**
0	88.305	0.99	0	0	0	0	0	0
1	62.4187	0.28	0.55	0	0	0	0	0
2	63.2895	0.1	0.56	0	0	0	0	0
3	72.1992	0.1	0	0.22	0	0	0	0
4	199.97	0.2	0	0.45	0	0	0	0
5	65.5873	0.6	0	0.6	0.1	0	0	0
6	98.5204	0.1	0	0.3	0.55	0	0	0
7	185.725	0.1	0	0	0	0	0	0
8	55.0673	0.14	0.96	0.11	0	0	0	0
9	78.979	0.1	0.3	0.4	0.1	0	0	0
10	72.9482	0.3	0.4	0.1	0.1	0	0	0
11	62.9658	0.1	0.77	0.16	0	0	0	0
12	116.401	0.19	0.48	0.93	0	0	0	0
13	58.0042	0.1	0.6	0.1	0.1	0	0	0
14	55.1357	0.1	0.6	0.1	0.1	0	0	0
15	67.7978	0	0	0	0	0	0	3
16	206.325	0	0	0	0	1	4	0
17	112.348	0	0	0	0	1	2	0
18	117.967	0	0	0	0	1	3	0
19	86.9144	0	0	0.9	0	1	2	0
20	66.6038	0	0	0	0	0	11	0
21	70.4779	0	0.6	0	0	0	2	0
22	88.3048	0	0	0	0	0	2	0
23	148.004	0.3	0	0.3	0	0	2	0
24	191.77	0	0	0	0	0	5	0
25	88.3048	0	0	0	0	0	2	0
26	88.3048	0.3	0	0.3	0	0	2	0
27	63.7073	0	0.6	0	0	0	2	0
28	88.3048	0	0	0	0	0	2	0
№ метода	SMAPE	Alpha	Delta	Gamma	Phi	countRulesIn	countFuzzyParts	C
0	48.7203	0.1	0	0	0	0	0	0
1	23.2163	0.1	0.78	0	0	0	0	0
2	23.2064	0.1	0.77	0	0	0	0	0
3	29.9321	0.12	0	0.1	0	0	0	0
4	199.999	0.1	0	0.1	0	0	0	0
5	30.8364	0.1	0	0.1	0.95	0	0	0
6	30.4904	0.3	0	0.5	0.5	0	0	0
7	179.47	0.21	0	0	0	0	0	0
8	18.121	0.42	0.34	0.28	0	0	0	0

9	35.6636	0.15	0.7	0.5	0.1	0	0	0
10	23.3641	0.15	0.55	0.55	0.1	0	0	0
11	23.5104	0.1	0.56	0.22	0	0	0	0
12	27.3946	0.22	0.43	0.47	0	0	0	0
13	24.6474	0.1	0.8	0.1	0.1	0	0	0
14	24.6567	0.1	0.8	0.1	0.1	0	0	0
15	55.0156	0	0	0	0	0	0	1
16	211.945	0	0	0	0	1	2	0
17	30.1984	0	0	0	0	1	19	0
18	42.6818	0	0	0.6	0	1	2	0
19	38.6195	0	0	0.9	0	1	2	0
20	53.7893	0.9	0	0	0	0	2	0
21	33.2752	0	0.6	0	0	0	2	0
22	55.0156	0	0	0	0	0	2	0
23	37.653	0.3	0	0	0	0	2	0
24	98.9023	0.6	0.9	0	0	0	2	0
25	55.0156	0	0	0	0	0	2	0
26	55.0156	0.3	0	0	0	0	2	0
27	20.6664	0	0.9	0	0	0	2	0
28	55.0156	0	0	0	0	0	2	0
№ метода	SMAPE	Alpha	Delta	Gamma	Phi	countRulesIn	countFuzzyParts	C
0	5.21855	0.1	0	0	0	0	0	0
1	3.34172	0.15	0.47	0	0	0	0	0
2	3.79245	0.1	0.79	0	0	0	0	0
3	1.81061	0.31	0	0.11	0	0	0	0
4	1.81323	0.27	0	0.11	0	0	0	0
5	1.9882	0.15	0	0.3	0.95	0	0	0
6	1.98944	0.1	0	0.3	0.95	0	0	0
7	184.454	0.1	0	0	0	0	0	0
8	2.0383	0.23	0.39	0.1	0	0	0	0
9	2.07706	0.15	0.5	0.15	0.1	0	0	0
10	2.08808	0.15	0.5	0.1	0.1	0	0	0
11	2.29565	0.11	0.25	0.19	0	0	0	0
12	2.11159	0.25	0.39	0.11	0	0	0	0
13	3.42457	0.1	0.45	0.75	0.1	0	0	0
14	3.46392	0.1	0.35	0.95	0.1	0	0	0
15	5.58796	0	0	0	0	0	0	1
16	13.7156	0	0	0	0	1	2	0
17	5.66277	0	0	0	0	1	2	0
18	2.31829	0	0	0.6	0	1	2	0
19	7.64121	0	0.9	0.9	0	1	2	0
20	5.33463	0.3	0	0	0	0	2	0
21	3.49637	0	0.3	0	0	0	2	0
22	5.58796	0	0	0	0	0	2	0
23	134.384	0	0	0	0	0	2	0
24	136.903	0.6	0.3	0.3	0	0	2	0

25	5.58796	0	0	0	0	0	2	0
26	199.427	0	0	0	0	0	2	0
27	199.427	0	0	0	0	0	4	0
28	5.58796	0	0	0	0	0	2	0

Результаты прогнозирования методами ВР 105

№ метода	SMAPE	Alpha	Delta	Gamma	Phi	countRulesIn	countFuzzyParts	C
0	7.54982	0.1	0	0	0	0	0	0
1	5.97929	0.1	0.3	0	0	0	0	0
2	5.85794	0.12	0.56	0	0	0	0	0
3	6.86546	0.25	0	0.2	0	0	0	0
4	6.85832	0.23	0	0.19	0	0	0	0
5	6.52908	0.65	0	0.75	0.2	0	0	0
6	6.56133	0.55	0	0.95	0.1	0	0	0
7	183.441	0.12	0	0	0	0	0	0
8	5.73389	0.1	0.22	0.64	0	0	0	0
9	5.62878	0.15	0.2	0.1	0.1	0	0	0
10	5.6419	0.15	0.25	0.1	0.1	0	0	0
11	5.05964	0.13	0.18	0.23	0	0	0	0
12	5.57756	0.1	0.23	0.73	0	0	0	0
13	5.95192	0.1	0.25	0.95	0.1	0	0	0
14	5.96073	0.1	0.3	0.15	0.1	0	0	0
15	7.72623	0	0	0	0	0	0	1
16	15.0032	0	0	0	0	1	5	0
17	7.9048	0	0	0	0	1	13	0
18	7.41966	0	0	0.3	0	1	2	0
19	11.9145	0	0.9	0.9	0	1	2	0
20	6.97665	0	0	0	0	0	12	0
21	5.94874	0	0.3	0	0	0	2	0
22	5.99994	0	0.6	0	0	0	2	0
23	104.977	0	0	0	0	0	5	0
24	135.041	0.3	0.9	0.3	0	0	2	0
25	227.547	0	0	0	0	0	2	0
26	199.212	0	0	0	0	0	5	0
27	199.213	0	0	0	0	0	2	0
28	194.588	0	0	0	0	0	2	0

Результаты прогнозирования методами ВР 106

№ метода	SMAPE	Alpha	Delta	Gamma	Phi	countRulesIn	countFuzzyParts	C
0	3.31608	0.12	0	0	0	0	0	0
1	3.17224	0.29	0.38	0	0	0	0	0
2	4.15403	0.2	0.72	0	0	0	0	0
3	3.77134	0.1	0	0.28	0	0	0	0
4	3.96781	0.27	0	0.81	0	0	0	0
5	3.35165	0.1	0	0.6	0.8	0	0	0
6	3.35574	0.1	0	0.5	0.8	0	0	0
7	184.902	0.1	0	0	0	0	0	0

8	3.47882	0.13	0.33	0.1	0	0	0	0
9	3.62783	0.1	0.4	0.15	0.1	0	0	0
10	3.58173	0.1	0.45	0.15	0.1	0	0	0
11	3.21236	0.11	0.18	0.18	0	0	0	0
12	3.49882	0.12	0.33	0.1	0	0	0	0
13	4.21422	0.15	0.3	0.3	0.1	0	0	0
14	3.67659	0.2	0.3	0.1	0.1	0	0	0
15	3.67242	0	0	0	0	0	0	1
16	7.72926	0	0	0	0	1	4	0
17	3.70548	0	0	0	0	1	4	0
18	6.44424	0	0	0.9	0	1	2	0
19	13.6517	0	0.9	0.9	0	1	2	0
20	3.50933	0.3	0	0	0	0	2	0
21	3.3866	0.3	0.6	0	0	0	2	0
22	3.43317	0	0.9	0	0	0	2	0
23	815.256	0	0	0	0	0	2	0
24	80.2509	0.3	0	0.6	0	0	2	0
25	240.552	0	0	0	0	0	2	0
26	199.711	0	0	0	0	0	2	0
27	199.711	0	0	0	0	0	7	0
28	206.824	0	0	0	0	0	2	0

Результаты прогнозирования методами ВР 107

№ метода	SMAPE	Alpha	Delta	Gamma	Phi	countRulesIn	countFuzzyParts	C
0	17.6857	0.1	0	0	0	0	0	0
1	17.7297	0.18	0.14	0	0	0	0	0
2	18.2514	0.14	0.19	0	0	0	0	0
3	16.8343	0.12	0	0.12	0	0	0	0
4	31.6354	0.24	0	0.12	0	0	0	0
5	17.0494	0.1	0	0.15	0.95	0	0	0
6	18.627	0.1	0	0.1	0.75	0	0	0
7	179.468	0.18	0	0	0	0	0	0
8	15.2878	0.1	0.17	0.14	0	0	0	0
9	17.9307	0.1	0.2	0.8	0.1	0	0	0
10	16.6914	0.15	0.15	0.1	0.1	0	0	0
11	17.3378	0.11	0.21	0.11	0	0	0	0
12	16.6574	0.1	0.19	0.13	0	0	0	0
13	17.9624	0.15	0.2	0.1	0.1	0	0	0
14	17.5131	0.1	0.15	0.95	0.1	0	0	0
15	19.0298	0	0	0	0	0	0	1
16	86.3788	0	0	0	0	1	4	0
17	26.8446	0	0	0	0	1	4	0
18	16.5566	0	0	0.6	0	1	2	0
19	20.378	0	0.9	0.6	0	3	2	0
20	18.8419	0.3	0	0	0	0	2	0
21	18.8399	0	0.3	0	0	0	2	0
22	19.0298	0	0	0	0	0	2	0

23	219.988	0	0	0	0	0	2	0
24	134.026	0.6	0	0.3	0	0	2	0
25	19.0298	0	0	0	0	0	2	0
26	19.0298	0.3	0	0	0	0	2	0
27	19.0298	0.3	0	0	0	0	2	0
28	19.0298	0	0	0	0	0	2	0

Результаты прогнозирования методами ВР 108

№ метода	SMAPE	Alpha	Delta	Gamma	Phi	countRulesIn	countFuzzyParts	C
0	7.64361	0.66	0	0	0	0	0	0
1	4.37025	0.37	0.79	0	0	0	0	0
2	4.97289	0.32	0.78	0	0	0	0	0
3	5.97719	0.21	0	0.92	0	0	0	0
4	7.49764	0.15	0	0.69	0	0	0	0
5	3.72994	0.2	0	0.7	0.7	0	0	0
6	3.67308	0.1	0	0.75	0.7	0	0	0
7	183.352	0.14	0	0	0	0	0	0
8	5.34737	0.17	0.14	0.77	0	0	0	0
9	8.14776	0.2	0.45	0.1	0.1	0	0	0
10	8.45684	0.2	0.5	0.1	0.1	0	0	0
11	6.29451	0.2	0.17	0.47	0	0	0	0
12	4.645	0.13	0.41	0.91	0	0	0	0
13	8.96562	0.2	0.55	0.9	0.1	0	0	0
14	7.27382	0.25	0.55	0.95	0.1	0	0	0
15	6.17201	0	0	0	0	0	0	19
16	28.9575	0	0	0	0	1	4	0
17	3.59722	0	0	0	0	1	19	0
18	8.67058	0	0	0	0	1	3	0
19	14.3632	0	0.9	0.9	0	1	2	0
20	7.61565	0.6	0	0	0	0	2	0
21	4.24476	0	0.3	0	0	0	2	0
22	7.64795	0	0	0	0	0	2	0
23	239.551	0	0	0	0	0	2	0
24	82.949	0.3	0	0.3	0	0	2	0
25	7.64795	0	0	0	0	0	2	0
26	200	0	0	0	0	0	2	0
27	7.64795	0.3	0.9	0.6	0	0	2	0
28	7.64795	0	0	0	0	0	2	0

Результаты прогнозирования методами ВР 109

№ метода	SMAPE	Alpha	Delta	Gamma	Phi	countRulesIn	countFuzzyParts	C
0	79.2784	0.1	0	0	0	0	0	0
1	56.5068	0.1	0.42	0	0	0	0	0
2	62.653	0.11	0.78	0	0	0	0	0
3	65.8823	0.58	0	0.45	0	0	0	0
4	149.905	0.1	0	0.64	0	0	0	0
5	39.5453	0.3	0	0.8	0.75	0	0	0
6	73.179	0.1	0	0.1	0.3	0	0	0
7	184.314	0.28	0	0	0	0	0	0
8	48.8793	0.1	0.88	0.63	0	0	0	0
9	48.4003	0.1	0.85	0.5	0.1	0	0	0
10	57.6186	0.1	0.75	0.5	0.1	0	0	0
11	43.2103	0.22	0.91	0.37	0	0	0	0
12	45.3407	0.3	0.88	0.45	0	0	0	0
13	57.0269	0.1	0.15	0.1	0.1	0	0	0
14	56.9194	0.1	0.2	0.1	0.1	0	0	0
15	49.7808	0	0	0	0	0	0	1
16	179.683	0	0	0	0	1	2	0
17	78.1842	0	0	0	0	1	17	0
18	54.1805	0	0	0	0	1	7	0
19	54.6545	0	0.9	0	0	4	2	0
20	48.6214	0	0	0	0	0	19	0
21	58.215	0	0	0	0	0	10	0
22	83.0916	0	0	0	0	0	2	0
23	574.338	0	0	0	0	0	6	0
24	128.174	0.3	0	0.3	0	0	2	0
25	83.0916	0	0	0	0	0	2	0
26	199.972	0	0	0	0	0	6	0
27	199.972	0	0	0	0	0	6	0
28	83.0916	0	0	0	0	0	2	0

Результаты прогнозирования методами ВР 110

№ метода	SMAPE	Alpha	Delta	Gamma	Phi	countRulesIn	countFuzzyParts	C
0	18.1858	0.99	0	0	0	0	0	0
1	17.6922	0.33	0.42	0	0	0	0	0
2	17.4097	0.26	0.15	0	0	0	0	0
3	16.5054	0.44	0	0.1	0	0	0	0
4	17.1563	0.37	0	0.1	0	0	0	0
5	13.6652	0.55	0	0.75	0.1	0	0	0
6	13.9943	0.35	0	0.8	0.1	0	0	0
7	181.719	0.28	0	0	0	0	0	0
8	18.2258	0.17	0.27	0.75	0	0	0	0
9	17.8022	0.1	0.15	0.25	0.1	0	0	0
10	16.5882	0.1	0.15	0.3	0.1	0	0	0
11	15.6391	0.11	0.2	0.26	0	0	0	0

12	18.301	0.17	0.3	0.81	0	0	0	0
13	17.9243	0.3	0.35	0.15	0.1	0	0	0
14	18.2142	0.3	0.3	0.1	0.1	0	0	0
15	18.1857	0	0	0	0	0	0	1
16	35.9981	0	0	0	0	1	4	0
17	13.0285	0	0.3	0	0	2	2	0
18	24.134	0	0	0	0	1	3	0
19	27.9997	0	0.9	0.9	0	1	2	0
20	13.8435	0	0	0	0	0	9	0
21	16.2927	0.9	0.6	0	0	0	2	0
22	159.195	0	0	0	0	0	2	0
23	116.467	0	0	0	0	0	4	0
24	163.571	0.3	0.3	0.6	0	0	2	0
25	168.773	0	0	0	0	0	2	0
26	200	0	0	0	0	0	4	0
27	18.1857	0.3	0.6	0.6	0	0	2	0
28	200	0	0	0	0	0	2	0

Результаты прогнозирования методами ВР 111

Приложения 4. Акты внедрения

АКТ
сдачи-приемки результатов по Заданию 2.1182.2017/ПЧ
на выполнение работы по теме «Разработка методов и средств автоматизации производственно-технологической подготовки агрегатно-сборочного самолетостроительного производства в условиях мультипродуктовой производственной программы»
выполняемой в рамках государственного задания на 2017 год

«26» марта 2018 г. г. Москва

Министерство образования и науки Российской Федерации в лице и.о. директора Департамента науки и технологий Минцаева Магомеда Шаваловича, действующего на основании доверенности от «26» марта 2018 г. № 08-210/н именуемое в дальнейшем «Минобрнауки России», с одной стороны и федеральное государственное бюджетное образовательное учреждение высшего образования «Ульяновский государственный технический университет», именуемое в дальнейшем «Исполнитель», в лице И. о. ректора Пинкова Александра Петровича, действующего на основании _____, с другой стороны, именуемые в дальнейшем «Стороны», составили и подписали настоящий Акт о нижеследующем:

1. Исполнитель сдал, а Минобрнауки России принял результаты по Заданию № 2.1182.2017/ПЧ на выполнение работы по теме «Разработка методов и средств автоматизации производственно-технологической подготовки агрегатно-сборочного самолетостроительного производства в условиях мультипродуктовой производственной программы».

2. Работа выполнена Исполнителем в соответствии с утвержденным Заданием. Отчетные документы оформлены надлежащим образом.

3. Настоящий Акт составлен в двух экземплярах, имеющих одинаковую юридическую силу – по одному для каждой из Сторон.

Исполнитель
Ульяновский государственный технический университет
И. о. ректора
_____(А.П. Пинков)
М.П.

Минобрнауки России
Департамент науки и технологий
и.о. директора
_____(М.Ш. Минцаев)
М.П.

УТВЕРЖДАЮ
Генеральный директор,
председатель НТС ФНПЦ
АО «НПО «Марс», к.т.н.
_____ В.А. Маклаев
«___» _____ 2018 г.

АКТ

об использовании результатов кандидатской диссертации Г.Ю. Гуськова
"Методы и средства формирования предметных онтологий в
автоматизированном проектировании программно-аппаратных комплексов"

Научно-техническая комиссия в составе:

председателя комиссии:	первый заместитель генерального директора по науке, к.т.н. Э.Д. Павлыгин,
членов комиссии:	главный научный сотрудник, д.т.н. Г.П. Токмаков, начальник отдела ИАСУП, к.т.н. А.А. Перцев ведущий инженер-программист, к.т.н. Радионова Ю.А.

настоящим актом подтверждает использование следующих научных и практических результатов диссертационной работы Г.Ю. Гуськова "Методы и средства формирования предметных онтологий в автоматизированном проектировании программно-аппаратных комплексов" для анализа концептуальных моделей программных систем в составе программно-аппаратных комплексов:

- онтологическая модель языка диаграмм UML, обеспечивающая гибкую поддержку различных спецификаций языка представления моделей

программных систем;

- онтологическая модель представления шаблонов проектирования программных модулей программно-аппаратных комплексов;

- алгоритм трансформации диаграмм классов языка UML в онтологию на языке OWL на основе применения специфических наборов логических утверждений ABox;

- комплекс программ, составляющий систему интеллектуального проектирования и разработки, позволяющий выполнять информационную поддержку на протяжении всего жизненного цикла проектируемой системы и оценивать текущее состояние проекта.

Комплекс программ интеллектуального проектирования и разработки использован при проектировании автоматизированных систем специального назначения.

Эффективность использования научно-технических результатов подтверждена экспериментальными исследованиями, целью которых являлось определение количественной оценки качества выполнения поиска прототипов программных модулей в электронном архиве в сравнении с традиционными методами поиска.

Для реализации информационной поддержки проектирования программно-аппаратных комплексов на ФНПЦ АО «НПО «Марс» были сформированы онтологические представления 30 проектов. Точность выполнения поисковых запросов к электронному архиву с использованием онтологических моделей примерно на 40% лучше по сравнению с аналогичными системами при определении схожих прототипов разрабатываемых систем. Построенная гистограмма частот используемых в проектах элементов языка UML на 86% совпадает с экспертным результатом.

Основные результаты диссертационного исследования получены в ходе выполнения и внедрения этапа «Разработка программной системы интеллектуального анализа текстовой информации» НИР «Система интеллектуального поиска и анализа в Интернет-СМИ и социальных сетях»,

выполняемых Ульяновским государственным техническим университетом по заказу ФНПЦ АО «НПО «Марс».

Председатель комиссии:

Первый заместитель генерального директора

по науке, к.т.и  Э.Д. Павлыгин

Члены комиссии:

Главный научный сотрудник, д.т.н.  Г.П. Токмаков

Начальник отдела ИАСУП, к.т.н. А.А. Перцев

Ведущий инженер-программист, к.т.н. Ю.А. Радионова

Приложение №1
к Договору о научно-техническом сотрудничестве
и совместных научных исследованиях
от «12» сентября 2014 г.

План научных исследований

Состав работ			
1. Разработка нечетких моделей экспоненциального сглаживания для прогнозирования временных радов.			
2. Воспроизведение нечеткого метода Direct Set Assignment для прогнозирования временных рядов.			
3. Разработка комбинированных моделей и моделей агрегирования нечетких мнений экспертов, где в роли экспертов выступают модели прогнозирования временных рядов.			
Сроки выполнения работ			
Дата начала	15.09.2014	Дата окончания	30.11.2014
Объем и порядок финансирования			
Объем финансирования	200000 (двести тысяч) рублей		
Источники финансирования	Финансирование работ осуществляется за счет средств Предприятия		
Порядок финансирования	Предприятие перечисляет Университету аванс в размере 30%, в течении 5 дней со дня выставления счета. Оплату в размере 70% Предприятие перечисляет Университету в течении 5 дней со дня подписания акта выполненных работ.		

Университет	Предприятие
Проректор по научной работе	Генеральный директор
_____/Ярушкина Н.Г./	_____/Артюхов М.В./
«09» сентября 2014 г.	«12» сентября 2014 г.
М.П.	М.П.

Приложение №1
к Договору о научно-техническом сотрудничестве
и совместных научных исследованиях
от «12» сентября 2014 г.

Пояснительная записка к плану научных исследований.

1. Разработка нечетких моделей экспоненциального сглаживания для прогнозирования временных радов.

Экспоненциальное сглаживание относится к адаптивным методам прогнозирования временных рядов. За счет наличия адаптивного механизма, простотой реализации и легкой интерпретации результата, экспоненциальное сглаживание играет важную роль в бизнес-прогнозировании. Наряду с классическим экспоненциальным сглаживанием применяются байесовское экспоненциальное сглаживание и подход на основе пространства состояний.

Огромную популярность в бизнесе получил метод автоматического прогнозирования с использованием набора моделей экспоненциального сглаживание. В качестве набора моделей используется расширенная классификация Pegels.

Trend Component		Seasonal Component		
		N (None)	A (Additive)	M (Multiplicative)
N	(None)	N,N	N,A	N,M
A	(Additive)	A,N	A,A	A,M
A_d	(Additive damped)	A_d,N	A_d,A	A_d,M
M	(Multiplicative)	M,N	M,A	M,M
M_d	(Multiplicative damped)	M_d,N	M_d,A	M_d,M

В классификация 5 вариантов тренда и 3 варианта сезонности, на пересечении строки тренда и столбца сезонности получается одна из моделей экспоненциального сглаживания.
Прогнозирование осуществляется с помощью каждой модели, после чего выбирается лучшая модель. В практических задачах бизнес-прогнозирования метод показал очень высокую эффективность.
Автор метода Rob J Hyndman (http://robjhyndman.com/), профессор департамента эконометрики и бизнес прогнозирования университета Monash, Австралия.

Цель работы: аналогично методу Rob J Hyndman, разработать метод прогнозирования временных рядов с использованием набора нечетких моделей экспоненциального сглаживания и выбором лучшей модели. Набор нечетких моделей экспоненциального сглаживания реализовать по аналогии с классификацией Pegels.

2. Воспроизведение нечеткого метода Direct Set Assignment для прогнозирования временных рядов.

Direct Set Assignment - новый нечеткий нелинейный метод прогнозирования временных рядов. Метод был предложен в диссертации Jeffrey Stewart Plouffe, University of Rhode Island. Согласно диссертации, комбинация метода Direct Set Assignment с экспоненциальным сглаживанием показывает очень хороший результат.

Цель работы: воспроизведение нечеткого метода Direct Set Assignment.

3. Разработка комбинированных моделей и моделей агрегирования нечетких мнений экспертов, где в роли экспертов выступают модели прогнозирования временных рядов.

Цель работы: разработать метод комбинирования прогнозов временных рядов, полученных с помощью различных моделей, используя нечеткую регрессию. Разработать метод агрегирования мнений экспертов, где в роли экспертов выступают различные модели прогнозирования временных рядов. Провести сравнение методов.

Наименование исполнителя:
федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Ульяновский государственный технический университет»
432027, г. Ульяновск, ул. Северный Венец, д. 32
ИНН 7325000052/ КПП 732501001
УФК по Ульяновской области
(Ульяновский государственный технический университет л/сч 20686Х85090)
Банк: ГРКЦ ГУ Банка России по Ульяновской обл., г. Ульяновск,
БИК 047308001
р/сч 40503810900001000001
ОКПО 02069378 ОКОНХ 92110
Назначение платежа:
Доходы от выполнения научно-исследовательских и опытно-конструкторских работ
ОФК 6800 (07330201010010000130)

Наименование заказчика:
ООО «Эверест Ресерч»
Адрес: 115432, г. Москва,
ул. Бориса Галушкина, д. 14/1
Р/сч: 40702810802610000143 в
ОАО «Альфа-Банк» г. Москва
К/сч: 30101810200000000593
Банк: ОАО «Альфа-Банк» г. Москва
ИНН: 7728168971
КПП: 771701001

АКТ
сдачи-приемки на выполненные работы
«Разработка нечетких моделей и реализация нечетких методов прогнозирования временных рядов»
по этапу договора № Д346 от 23 сентября 2014г.
Составлен 20 марта 2015г.

Мы, нижеподписавшиеся, представитель Исполнителя - проректор по НР Государственного образовательного учреждения высшего профессионального образования «Ульяновский Государственный Технический Университет» Ярушкина Н.Г., с одной стороны, и представитель Заказчика – генеральный директор ООО «Эверест Ресерч» Артюхов М.В., с другой стороны, составили настоящий акт о том, что разработка нечетких моделей и реализация нечетких методов прогнозирования временных рядов выполнены в полном объеме и удовлетворяют условиям договора.

Право собственности у Заказчика возникает с момента передачи ему результатов выполненной работы в целом и оплаты им выполненных работ.

Объем выполненных работ составляет 200 000 (Двести тысяч) рублей. НДС не предусмотрен.

Работу сдал
от Исполнителя
Н.Г. Ярушкина

Работу принял
от Заказчика
М.В. Артюхов