МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

На правах рукописи

Галочкин Михаил Владимирович

МЕТОДЫ И СРЕДСТВА ОБРАЗНО-СЕМАНТИЧЕСКОГО СОПРОВОЖДЕНИЯ ПРОЦЕССОВ РЕШЕНИЯ ПРОЕКТНЫХ ЗАДАЧ

05.13.12 -Системы автоматизации проектирования (промышленность)

Диссертация на соискание ученой степени
кандидата технических наук

Научный руководитель
д.т.н., профессор
Соснин П.И.

Ульяновск 2016

Принятые сокращения и обозначения 4

Введение 5

Глава 1. Компьютерные системы сопровождения процесса решения задач на основе моделей 13

1.1. Анализ использования образной графики в автоматизированном проектировании 14

1.2. Основы сопровождение процессов решения задач 46

1.3. Обобщенная постановка задачи 52

1.4. Выводы 65

Глава 2. Формализация процесса решения задач 67

2.1. Подход к образно-семантическому сопровождению процессов решения проектных задач 68

2.2. Типизированный набор образно-семантических моделей и система их согласованных преобразований 77

2.3. Обобщенное представление методов понятийно-образной поддержки 83

2.4. Выводы 92

Глава 3. Методологическое обеспечение процесса решения задач на основе моделей 94

3.1. Метод итеративного согласования понятийного и образного содержания текстовых единиц с использованием их преобразования в прологоподобную форму 96

3.2. Метод понятийно-образной поддержки процесса пошаговой детализации в прецедентно-ориентированном решении проектных задач .. 105

3.3. Примеры решения проектной задачи с помощью разработанных методов и средств 124

3.4. Выводы 152

Глава 4. Особенности реализации специализированного графического редактора. 153

4.1. Реализация изобразительного представления 162

4.2. Реализация декларативного представления 165

4.3. Реализация концептуально-алгоритмического представления 179

4.4. Оценка степени повышения эффективности проектировщика при использовании системы поддержки процесса решения задач 185

4.5. Выводы 200

Заключение 202

Список литературы 203

Приложение 1. Акты внедрения 214

Принятые сокращения и обозначения

BPMN – Business Process Model and Notation

DSL – domain specific languages

GOMS - Goals, Operators, Methods and Selection rules

MDA - Model Driven Architecture

MDD - Model Driven Development

MDSE - Model Driven Software Engineering

OMG - Object Management Group

RHS/LHS - Right Hand Side/Left Hand Side

SIS – Software Intensive System

UML – Unified Modeling Language

XML – eXtensible Markup Language

ПО – программное обеспечение

САПР – система автоматизированного проектирования

СКФ – социо-киберфизическая реальность

СУБД – система управления базами данных

Введение

Расширяющаяся компьютеризация всех сфер человеческой активности, включая различные виды персональной и коллективной деятельности, уже вывела на необходимость осуществления профессиональной деятельности в условиях социо-киберфизической реальности (СКФ-реальности), в которой различные неодушевленные предметы и физическая реальность, за счет использования разнообразных вычислительных ресурсов и сенсорных возможностей, наделяются подобием интеллектуальных функций и интеллектуального поведения.

Создавая человека, природа, адаптировав его сущность под свое естество, не рассчитывала на появление таких кибер-посредников во взаимодействии человека с его окружением, что уже сейчас является источником серьезных проблем, среди которых выделяется «проблема чрезвычайно низкой успешности разработок таких составляющих СКФ-реальности как системы, интенсивно использующие программное обеспечение. «Standish Group» на протяжении 12 лет (начиная с 1994 г.) оценивала успешность разработок по всему миру пришла к выводу, что не удается достичь существенного прогресса в этой области и успешность разработок находится на уровне 30% [58]. В отчете 2015 года было проанализировано порядка 50 000 проектов по всему миру в различных областях (таблица 1.1).

Таблица 1.1 - Сравнение успешности разработок ПО [58]

	2011	2012	2013	2014	2015
Завершились успешно	29%	27%	31%	28%	29%
Потребовали дополнительных ресурсов	49%	56%	50%	55%	52%
Завершились не успешно	22%	17%	19%	17%	19%

Следует отметить, что согласно исследование успешность крупных проектов существенно ниже по сравнению с более мелкими проектами (таблица 1.2).

Таблица 1.2 – Сравнение успешности разработок в зависимости от размера проектов [58]

	Завершились успешно	Потребовали дополнительных ресурсов	Завершились не успешно
Очень большие	2%	7%	17%
Большие	6%	17%	24%
Средние	9%	26%	31%
Небольшие	21%	32%	17%
Маленькие	62%	16%	11%
Всего	100%	100%	100%

Отмеченная причина проблем (существенное различие между естественным взаимодействием человека с физическим миром и его взаимодействием с компьютеризованными приложениями) и сами проблемы приводят к новым постановкам вопросов и идей о формах человеко-компьютерного взаимодействия с СКФ-реальностью. Среди таких вопросов и идей особо важны те, которые имеют отношение к процессам разработки составляющих СКФ-реальности, потому что в них нельзя не учитывать необходимость эффективного переплетающегося комплексирования естественных и компьютеризованных составляющих взаимодействия разработчиков (в первую очередь проектировщиков) со средой разработки. Более того, эти специалисты должны учитывать будущее взаимодействие пользователей с разрабатываемыми компьютеризованными продуктами.

Вышесказанное указывает на актуальность научных исследований и разработок, нацеленных на снижение существующих различий между

формами естественного взаимодействия человека с физическим миром и формами взаимодействия, которые «навязываются» ему современной практикой разработки компьютеризованных сред. Для снижения различий в диссертации предлагается ряд новаций, в основу которых положено моделирование естественной экспериментальной активности проектировщика, в процессе которой он согласованно использует мысленное (визуальное) воображение и понятийные механизмы сознания.

Моделирование экспериментальной активности в диссертации рассматривается и осуществляется в рамках решения проектных задач в процессах разработки систем, интенсивно использующих программное обеспечение (Software Intensive Systems, SIS) как самых проблемных составляющих современных компьютеризованных сред. Именно для этого класса компьютеризованных конструктов накоплена богатейшая коллекция причин низкой успешности разработок, а также библиотека методов и средств способствующих повышению эффективности работ. Неоценимый вклад в эту проблемную область внесли S. W. Ambler [53], V. R. Basili, G. Booch, I. Jacobson [17, 23], A. Cockburn [2], W. S. Humphrey, Ph. Kruchten. Отметим, что к классу SIS относятся автоматизированные системы (АС), для которых в отечественной нормативной базе накоплен богатый опыт.

В диссертационной работе роль **области исследования** возложена на управляемый моделями процесс решения проектных задач в условиях оперативного взаимодействия с базой прецедентов.

Направление исследования связано с построением в процессе решения проектных задач их моделей повторного использования (моделей прецедентов), согласованных с естественно-профессиональным опытом.

Функции **объекта исследования** выполняет визуальная поддержка процессов решения, управляемого моделями, в число которых включены модели прецедентов.

Предметом исследования являются модели, методы и средства образно-семантического сопровождения прецедентно-ориентированного решения проектных задач в разработках автоматизированных систем.

Целью диссертационной работы является снижение существующего существенного различие между естественным взаимодействием человека с физическим миром и его взаимодействием с компьютеризованными приложениями и, тем самым, внести вклад в снижение негативных проявлений человеческого фактора в проектировании SIS и AC за счет включения образно-семантического поддержки в процессы прецедентно-ориентированного решения проектных задач.

Задачи диссертационного исследования.

1. Провести анализ современных методов, моделей и средств, которые применяются при решении задач в САПР и смежных областях.
2. С позиций автоматизации, провести исследования таких составляющих интеллектуальной обработки задачных ситуаций как: мысленное экспериментирование на основе данных, поступающих от органов чувств и активности феномена «интеллектуальное воображение» (ограничившись вопросами визуализации); диалоговая структуризация мысленных представлений и их семантический анализ; итеративное согласование названных интеллектуальных активностей, порождающее понимание результатов обработки.
3. Разработать метод понятийно-образной поддержки действий проектировщика в прецедентно-ориентированном решении проектных задач.
4. Разработать модифицированную версию метода пошаговой детализации, адаптировав его реализацию к итеративному построению для проектных задач соответствующих моделей

прецедентов, включающему автоматизированное мысленное экспериментирование и его понятийно-образное сопровождение.

5. Разработать метод согласованных преобразований между представленными моделями, между моделями и их вопросно-ответным представлением, между моделями и их псевдокодовым и прологоподобным описанием.
6. Разработать необходимые компоненты (включая специализированный графический редактор моделей), как расширения инструментальной среды WIQA, обеспечивающие образно-семантическую поддержку прецедентно-ориентированного решения проектных задач.
7. Оценить эффективность предложенных моделей, методов и средств.

Методы исследования основаны на использовании положений и методов теории множеств, теории классификации, теории графов, теории автоматов, теории автоматизированного проектирования, основ системотехники.

Достоверность полученных результатов подтверждается полнотой и корректностью исходных посылок, логичностью рассуждений (использующих вопросно-ответную формализацию и анализ), а также вычислительными экспериментами и результатами практического использования.

На научную новизну претендуют:

1. Типизированный набор образно-семантических моделей и система их согласованных преобразований, отличающихся тем, что для каждого типа моделей используется композиция проекций, ориентированных на программную интерпретацию.

2. Метод понятийно-образной поддержки процесса пошаговой детализации в прецедентно-ориентированном решении проектных задач, использующем конструктивное и управляемое включение автоматизированного концептуального экспериментирования и моделирование.
3. Метод итеративного согласования понятийного и образного содержания текстовых единиц (предложений текста постановки задачи) с использованием их преобразования в прологоподобную форму, специфику которого определяет взаимодополняющее итеративное уточнение графического и текстового представления требований и спецификаций с использованием автоматизированного взаимодействия с онтологией.

Практическая ценность полученных результатов состоит в разработке программного обеспечения, включающего следующие компоненты.

1. Не имеющий аналогов комплекс инструментально-технологических средств обслуживающий авторский подход к прецедентно-ориентированному решению профессиональных задач с использованием образно-семантической поддержки, способствующей конструктивному и управляемому включению в процессы решения мысленного воображения и повышающий эффективность человеко-компьютерного взаимодействия.
2. Средства контроля версий вопросно-ответных проекций графических моделей, учитывающие специфику такого представления и позволяющие фиксировать этапы работы над задачей и переключаться между ними с целью возврата или проверки корректности полученных результатов.

3. Средства перевода декларативного представления в согласованное с ним прологоподобное представление с целью экспериментирования, работы со словарем онтологий и поиска семантических ошибок.
4. Средства перевода концептуально-алгоритмического представления в согласованное с ним псевдокодовое представление и отладка его в псевдокодовом интерпретаторе.

На защиту выносятся следующие результаты:

1. Представление типизированного набора образно-семантических моделей композицией проекций, ориентированных на программную интерпретацию.
2. Взаимодополняющее итеративное уточнение графического и текстового представления требований и спецификаций с использованием автоматизированного взаимодействия с онтологией.
3. Использование пошаговой детализации в формировании постановки задачи для управления процессом построения концептуального решения в виде ее модели прецедента.

Реализация и внедрение результатов работы. Разработанные программные средства внедрены в практику работы ОАО НПО “Марс” (г. Ульяновск), ООО “ФБ-Групп” (г. Москва), Нижегородское отделение компании Intel (г. Нижний Новгород) и учебный процесс Ульяновского государственного технического университета (г. Ульяновск).

Апробация работы. Основные положения и результаты диссертации докладывались и обсуждались на следующих конференциях: Всероссийской научно-технической конференции аспирантов, студентов и молодых ученых “Информатика и вычислительная техника” (ИВТ-2010), г. Ульяновск, 2010; Российской конференции “Информатика и вычислительная техника” (ИВТ-

2012), г. Ульяновск, 2012; Российской школе-семинаре “Информатика, моделирование, автоматизация проектирования” (ИМАП-2010), г. Ульяновск, 2012; Российской школе-семинаре “Информатика, моделирование, автоматизация проектирования” (ИМАП-2013), г. Ульяновск, 2013; Научно-технической конференции профессорско-преподавательского состава (ППС-2013), г. Ульяновск, 2013; XI International conference on interactive systems: problems of human-computer interaction, г. Ульяновск, 2015; Научно-технической конференции семантические модели и технологии (TEL-2016), г. Казань, The First International Scientific Conference “Intelligent Information Technologies for Industry” (IITI’16), г Сочи, 2016; Всероссийская школа-семинар ИМАП-2016, г. Ульяновск, 2016.

Публикации. По проблеме диссертации опубликованы 19 печатных работ, в том числе 3 статьи в российских рецензируемых научных журналах, и 1 статья в издании индексируемым в SCOPUS.

Структура и объем диссертации. Диссертация изложена на 218 страницах машинописного текста, содержит 106 рисунков, 6 таблиц, состоит из введения, четырёх глав, заключения, списка литературы из 105 наименований на 10 страницах и 1 приложения на 5 страницах с актами о внедрении.

Сведения о личном вкладе автора. Научные результаты проведенных исследований, которые представлены в диссертационной работе и, выносимых на защиту, получены автором лично. Научному руководителю, принадлежит выбор направления исследований, постановка задачи и обсуждение. В публикациях с соавторами вклад соискателя определяется рамками представленных в диссертации результатов.

Глава 1. Компьютерные системы сопровождения процесса решения задач на основе моделей

Расширяющаяся компьютеризация всех сфер человеческой активности, включая различные виды персональной и коллективной деятельности, уже вывела на необходимость осуществления профессиональной деятельности в условиях, к которым человек не приспособлен. Природа, адаптировав его сущность под свое естество, не рассчитывала на появление такой кибер-реальности, что уже сейчас является источником серьезных проблем, среди которых выделяется «проблема чрезвычайно низкой успешности разработки систем интенсивно использующих программное обеспечение и САПР. На первое место выходит решение проблем увеличение естественности взаимодействия человека с компьютеризированными средами за счет ориентацию на механизмы интеллектуальной обработкой условных рефлексов, в основе которых лежит механизмы образного (правое полушарие мозга) и символического (левое полушарие мозга) восприятия реальности [52]. Акцент на указанные выше различия в интеллектуальные активности правого и левого полушарий делается для того, что в разработанной системе средств центральное место занимает учет этих когнитивных факторов процесса решения за счёт представление всех видов моделей в образно-семантическом и программном вариантах. Современные тенденции в этой области, такие как MDSE свидетельствуют о важности использование графических моделей для понимания и обмена знаниями о сложном объекте или системе. Принцип MDSE задуман как инструмент для применения графических диаграмм на всех стадиях разработки и преобразования их в артефакты (код, документация и т.д.) будущего программного обеспечения. Очевидно, что применение моделей может улучшить коммуникацию между разработчиками, а также более наглядно изображать аспекты будущей системы.

MDSE может быть описано как методология, которая подразумевает использование различных видов моделей при разработке программного обеспечения. Как правило, MDSE включает следующие аспекты:

- Нотации. Группа языков, используемых в конкретной методологии. Именно с помощью нотаций создаются модели.
- Методы и правила. Представляет собой совокупность активностей, которые приводят к получению решения, посредством следования определенным правилам, которые координируют и управляют процессом.
- Вспомогательные приложения, позволяющие автоматизировать процесс получения решения. Обычно включают в себя специализированный графический редактор, средства прототипирования и отладки решений, средства контроля версий.

Целью данной главы является исследование методологий, методов и средств поддержки процессов решения задач в различных видах человеческой деятельности, особенно при разработке сложных систем, к которым относится САПР. Проведён детальный обзор и классификация моделей, методов и средств, используемых в данной области.

1.1. Анализ использования образной графики в автоматизированном проектировании

Логично начать рассмотрение с общих методов представления информации в графическом виде, которые используются в различных отраслях знаний и представлены в виде периодической таблицы визуальных методов (рисунок 1.1.).

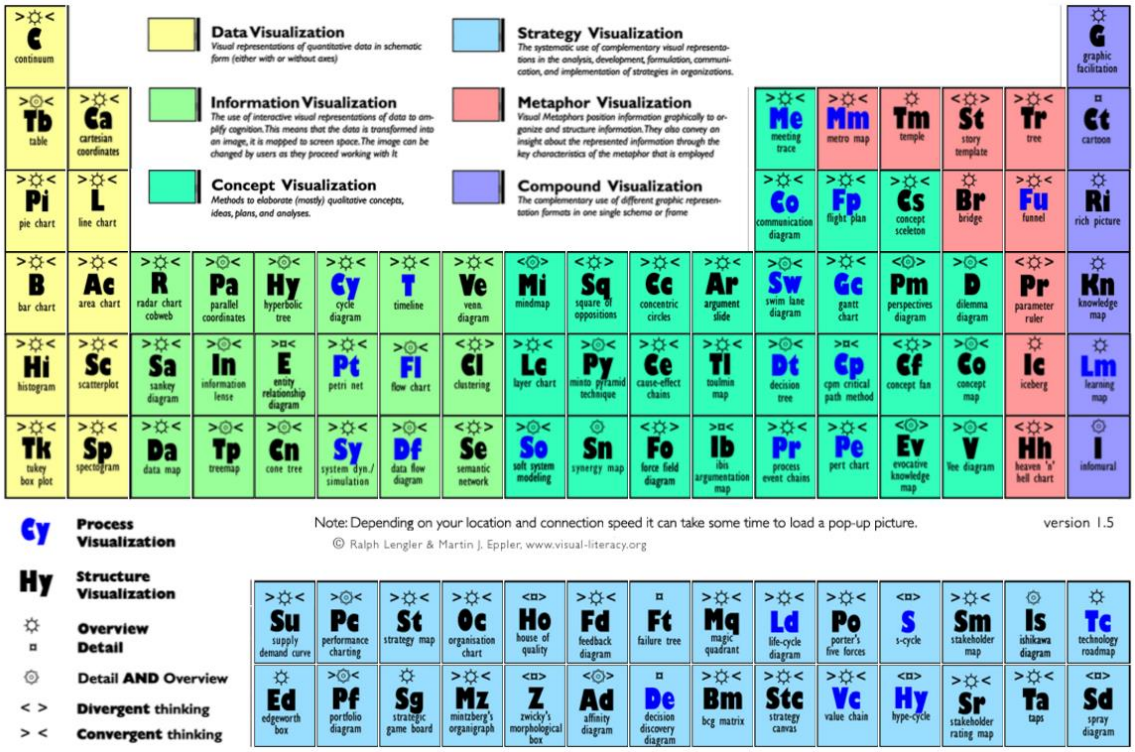

Рисунок 1. 1 - Периодическая таблица визуальных методов [1]

Таблица представляет методы, сгруппированные по группам, выделенных цветом. Строки обозначаются сложность метода; чем ниже строка, тем метод является более сложным. Выделяются следующие группы:

- Методы визуализации данных. Универсальные методы, позволяющие визуально представить количественные данные в схематичной форме: в виде таблиц, круговых диаграмм и т.д.
- Методы визуализации информации. Представление в виде семантических сетей или деревьев. Сюда относятся диаграммы классов (в более общем виде – диаграммы отношений между сущностями), сети Петри, flow chart (прообраз диаграмм активностей) и т.д.
- Методы визуализации концепций. В эти методы входят mind maps, диаграммы связей (упрощенный вид декларативного представления), деревья решений и т.д.

- Метод визуализации метафор. Представления информации через ключевые характеристики в виде метафор (tree, metro map, bridge и т.д.)
- Методы визуализации стратегий.
- Методы объединения визуализаций. Позволяют объединять несколько методов в одном рисунке или фрейме.

Существует большое количество графических нотаций, которые могут быть использованы в процессе решения задач. Их можно разделить на 2 большие группы: специализированные языки, и языки общего назначения. Специализированные ориентированы на описание задачи в определенной предметной области. К ним, можно отнести Business Process Modeling Language (BPML), Business Process Execution Language (BPEL), IDEF0, IDEF3. Языки общего назначения не имеют узкой направленности и могут быть использованы для описания решения в различных областях. К ним можно отнести UML [9], язык ДРАКОН и д. р.

В ряде работ авторов [33, 54, 89] выделяется 2 больших класса, на которые можно разделить языки. Это языки, опирающиеся на математический аппарат сетей Петри, и языки, опирающиеся на концепцию “Pi calculus”, предложенную Робином Милнером в 1991г [33, 32]. Основными преимуществами 1 класса языков (на этой концепции построены, например, диаграммы активностей UML 2.x [105]) является то, что проверенный временем и хорошо проработанный аппарат сетей Петри дает мощный математический фундамент. Но с другой стороны жесткий формализм и невыразительная графическая нотация делают практически невозможным применения этого подхода в чистом виде. Например, диаграммы активностей хоть и основываются на сетях Петри, позволяют отходить от ее принципов с целью увеличения наглядности и простоты восприятия. Также современные подходы к программированию вводят все новые и новые понятий, которые

нельзя выразить на основе сетей Петри (например, понятие «исключения» невозможно выразить на языке сетей Петри). Второй класс языков базируется на концепции “Pi-calculus” с которой детально можно ознакомиться в работах [33, 77, 88]. Например, модели в нотации BPMN [59] могут быть выражены с помощью математического аппарата данной концепции [54, 47].

Unified Modeling Language

Представляет собой универсальный язык моделирования для использования во всех областях человеческой деятельности. Принят в качестве международных стандартов ISO/IEC 19501:2005 (для версии языка 1.4.2) и ISO/IEC 19505-1, 19505-2 (для версии языка 2.4.1) [20]. Универсальность языка объясняется наличием большого количества различных типов моделей, позволяющих описывать как структурные, так и поведенческие аспекты системы (рисунок 1.2.). Поведенческие модели построены на базе концепции сетей Петри.

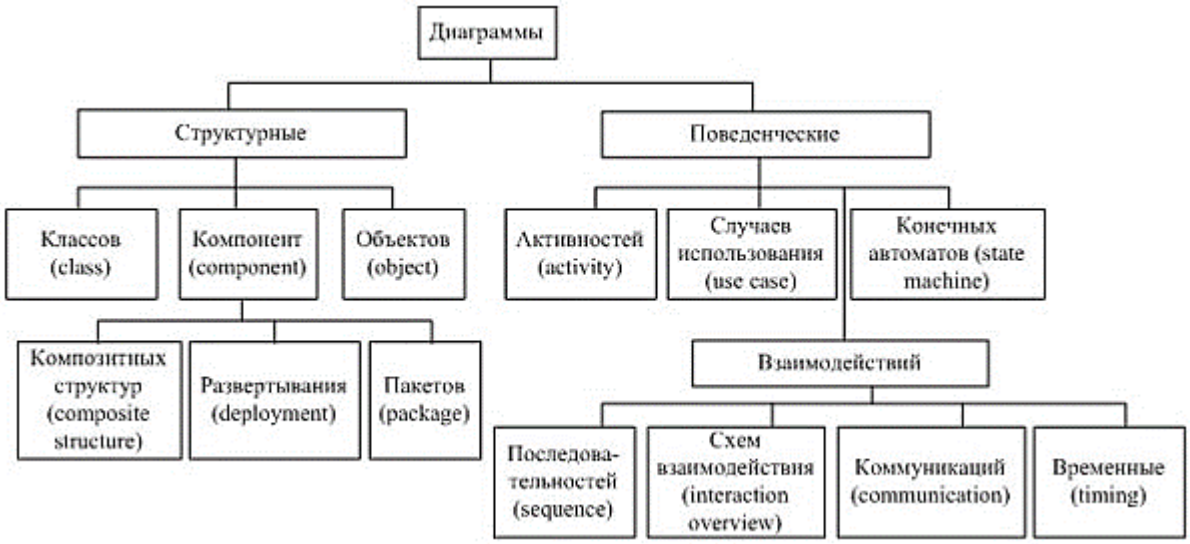

Рисунок 1. 2 - Виды диаграмм UML

Согласно исследованиям [65] наиболее используемыми диаграммами является диаграммы классов, диаграммы активностей, последовательностей и

вариантов использования. Согласно проведенным опросам основными вариантами использования являются:

1. Использование UML в качестве “мыслительного средства”
2. Как средства коммуникации между всеми заинтересованными сторонами
3. Для совместных размышлений (например, в методе “мозговой штурм”)

Uml является наиболее популярной нотации, но не смотря на это только 20% разработчиков использует его для рисования моделей (коммуникации) и только около 6% генерируют артефакты из них (код, документация) [65]. Причинами столь низкого процента использования является:

1. Проблема синхронизации моделей (другими моделями, кодом, документацией). Отсутствие развитых средств создания, редактирования, версионирования моделей
2. Отсутствие привязки к окружению. UML имеет дело с абстрактными моделями и не позволяет спускаться на уровень реализации.
3. Переусложненная нотация. Стандарт описывает порядка 20 типов моделей, каждая из которых имеет свою нотацию. Такое большое количество моделей обусловлено универсальностью языка.

Таким образом, при разработке оригинального подхода надо учитывать выводы, сделанные на основе исследований [65].

Диаграммы описания бизнес правил (VisiRule)

Представляет собой специализированный графический редактор, позволяющий экспертам в предметной области создавать экспертные системы путем описания бизнес правил в виде графических диаграмм (рисунок 1.3.). После описание бизнес правил будет из диаграммы будет

сгенерирован код на языке LPA Prolog , который может быть исполнен в виде веб или desktop приложения.

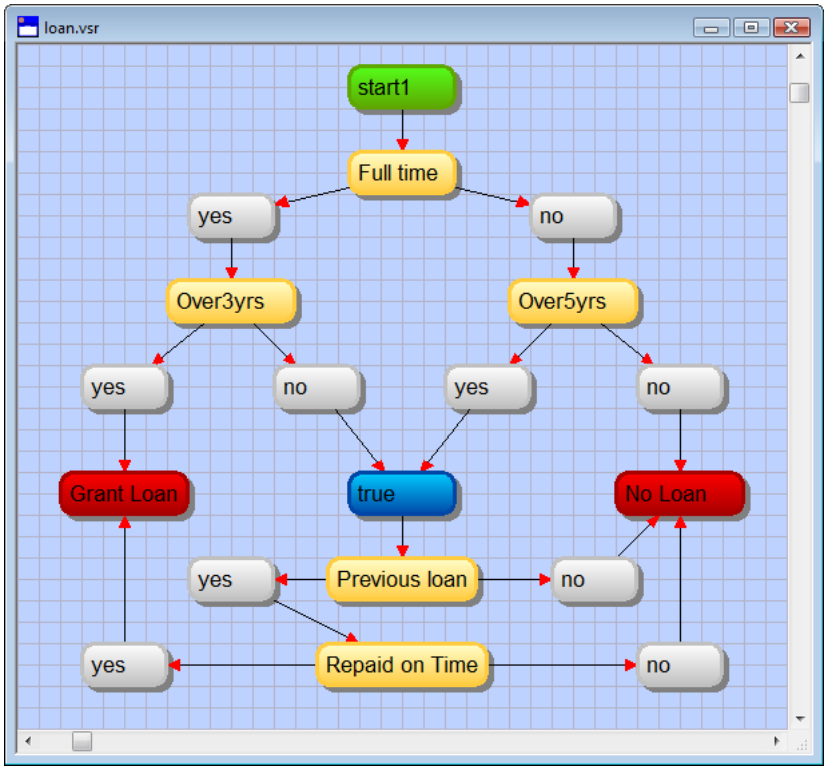

Рисунок 1. 3 - Пример простой диаграммы в редакторе VisiRule[66]

Диаграмма может состоять из следующего набора графических примитивов:

- Простой выбор. Пользователь может выбрать только 1 вариант из меню
- Множественный выбор. Позволяет выбрать несколько или не выбирать варианты из предложенного меню.
- Числовой ввод (отдельно для целых и вещественных чисел). Ожидается ввод числа из определенного диапазона.
- Ввод значений из заранее определенного множества. Используется, например, для ввода цвета (в виде ввода слова из множества: зеленый, красный, желтый)
- Произвольный ввод текста.

При исполнении происходит обход дерева, путем получение ответов на вопросы от пользователя (которые определяются типом используемых

вершин рассмотренных ранее). Обычно завершающей вершиной в графе служит вывод по результатам ответов. К недостаткам данной системы можно отнести закрытость, а также ограниченный набор поддерживаемых графических примитивов и моделей.

Influence Diagram (Analytica)

Представляет собой класс диаграмм, которые используются для описания процесса принятия решения. Были разработаны в 1970 годах и ориентированы на простоту восприятия. Нашли широкое применения в различных областях знаний (рисунок 1.4.).

Рисунок 1. 4 - Пример простой Influence Diagram в среде Analytica [70]

Нотация отличается простотой и состоит из 4 видов вершин и 3 видов связей:

1. Цель. Обычно представляется шести или восьми угольниками. В среде “Analytica” принято обозначать розовым шестиугольником. Целью является ответ на вопрос “Чего мы хотим получить?”;
2. Решение. Принято обозначать прямоугольником. Под решением понимается некая переменная составляющая, которая может быть изменена субъектом или организацией, принимающей решение. Отвечает на вопрос: “Какие факторы можно контролировать?”;
3. Ограничения. Обычно представляются овалом - описывают то, что не поддается непосредственному контролю. Под ограничением также понимают переменные факторы, значение которых точно не

определены или могут варьироваться в определенных пределах вне зависимости от нас;

4. Стрелки. Необходимы для определения зависимостей (или влияний) между элементами диаграммы. Стрелка между фигурами “A” и “B”, означает что изменение переменной “A” влияет на значение переменной “B”, то есть объекты находятся в некоторой зависимости.
5. Дополнительные фигуры. К таким фигурам относятся треугольники (внешние входные данные), прямоугольники со скругленными краями (глобальные переменные значения) и т.д. Среда “Analytica” расширяет стандартную нотацию, добавляя возможность задания иерархии между моделями, определения различных функциональных зависимостей (псевдокод), разрешает использовать циклы.

“Influence Diagram” принято строить справа налево. В верхнем правом углу обычно размещают цели, которые необходимо достичь.

1. Определяют цель через независимые изменяемые значения, которые могут непосредственно контролироваться.
2. Повторяют декомпозицию для каждого изменяемого значения.
3. Определяют входные данные и решения, которые они могут порождать.
4. Определяют хотя бы одно решение, которое может влиять непосредственно на цель.
5. Добавляют возможные значения (диапазоны) для известных переменных.
6. Добавляют математические отношения между элементами модели (например, итоговая цена включает количество товара, умноженного на цену единицы).
7. Вводят ограничения для всех переменных.

Построение Influence Diagram позволяет структурировать и наглядно представить большое количество зависимостей и факторов, которые влияют

на достижение целей. Следует отметить, что при наличии функциональных отношений становиться возможным выполнять моделирование.

Business Process Execution Language (BPEL)

BPEL – xml-подобный язык для формального описания бизнес процессов и протоколов взаимодействия между ними. Обычно среды, поддерживающие данный язык, позволяют создавать и редактировать конструкции в двух представлениях – текстовом и графическом, синхронизируя изменения в обеих перспективах. Первоначально BPEL не имел поддержки HumanTask [82, 19] и представлял собой гибрид языков WSFL и Xlang (IBM и Microsoft).

Любой поток представляет собой сервис, то есть он имеет какие-то входные и выходные параметры. Другими словами, работа с потоками осуществляется на базе архитектуры SOA [37, 38]. В первой спецификации BPEL(2003г) отсутствовало такое понятие как HumanTask, оно появилось только в 2007 г., когда были опубликованы спецификации BPEL4People и WS-HumanTask. Одним из недостатков данного подхода является нотация языка, которая не поддерживает произвольный переход между задачами, что очень часто необходимо в реальных ситуациях. Такие переходы невозможны, так как пользователь не задает связей между задачами. Он только переводит «activity» в нужные позиции. Блоки типа if, while и т.д., представляют собой “монолитные” блоки со всеми связями (любые связи добавляются автоматически). Основная палитра элементов представлена на рисунке 1.5.

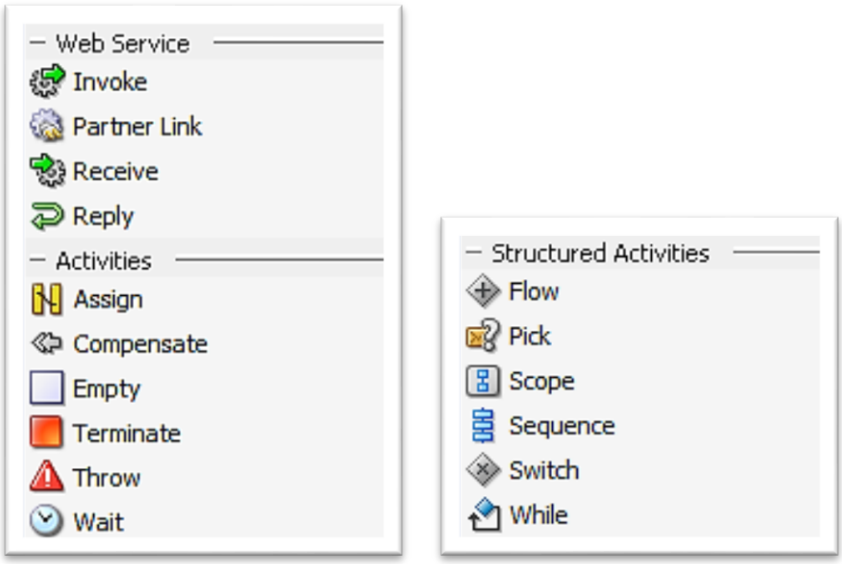

Рисунок 1. 5 - Нотация языка BPEL 1.1

Кострукции flow, while, switch представляют собой цельные блоки, не подразумевающие безусловных переходов (рисунок 1.6):

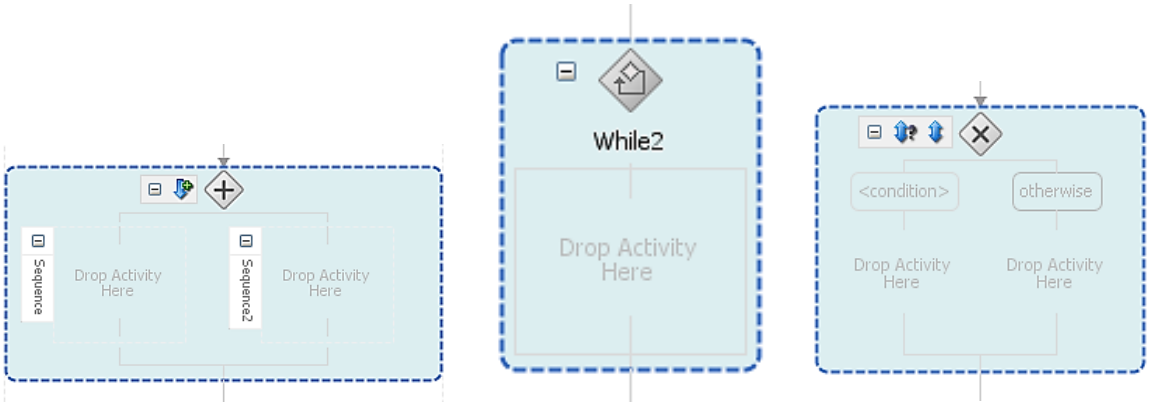

Рисунок 1. 6 - Конструкции языка flow, while, switch

Основной вывод, который можно сделать на основе анализа – нотация первоначально разрабатывалась для разработки программных систем на базе архитектуры SOA. Позже в нее были включены задачи, исполняемые человеком, и язык стал позиционироваться как язык описания бизнес процессов, то есть язык стал универсальным и позволяет описывать последовательности, включающие как человеческие, так и программные активности, но первоначальная направленность ограничивает гибкость системы.

Методы, используемых в процессе решения задач

Существует порядка 40 различных методов, которые могут быть использованы в процессе решения различных видов задач. Такие методы можно разбить на несколько больших групп – методы исследования задачи, визуализации задачи (или решения), методы анализа причин, повлекших появление задачи. Рассмотрим некоторые из них более подробно:

Конструктивная полемика (Constructive Controversy)

Представляет собой метод для исследования задачи и улучшения полученного решения. Человек, решающий задачу подвержен субъективным предрассудкам и не всегда может обозревать область задачи в целом. Для этой проблемы необходимо при решении задачи консультироваться с другими заинтересованными сторонами, что не всегда является возможным. Данная техника позволяет организовать “столкновение идей” с целью изучения решения (или задачи) с разных перспектив и выработке оптимального решения. Техника состоит из 5 шагов, представленных на рисунке 1.7:

Рисунок 1. 7 - Шаги метода “Конструктивная полемика”

Методика заключается в последовательном выполнении шагов и позволяет путем формирования отличного решения решать проблему неполноты и несовершенства оригинального решения. Методика имеет форму цикла и повторяется до достижения определенных показателей.

Анализ видов и последствий отказов (Failure Mode and Effects Analysis)

Первые публикации об этом методе были сделаны в 1949г. в США. В основе метода лежит потребность в анализе решения с точки зрения возможных неблагоприятных ситуаций. Корректность решения проверяется в ситуациях, формулируемых так: “А что, если …?”. Метод заключается в построении матрицы рисков, причем в правом нижнем углу рассматриваются наиболее благоприятные внешние параметры для метода (параметры при которых метод дает наилучший результат) а в левом верхнем – наиболее невыгодные параметры. Цветом выделяют участки, где предложенное решение показывает хороший, удовлетворительный или не удовлетворительный результат (обычно результаты выделяют зеленым, желтым и красным цветами соответственно). Так же данный метод позволяет оценить риски возможных неблагоприятных ситуаций.

Методология мягких систем (Soft Systems Methodology)

В основе подхода лежит принцип, что в случае, когда много факторов влияют на решение и приходится учитывать множество различных условий, существует возможность значительно упростить внешние факторы, не потеряв корректность решения. Метод построен на базе общей теории систем, которая рассматривает все как часть открытой, динамичной и взаимосвязанной системы. Методология была разработана Питером Чекландом в 1960-х годах, в современном формате подход был опубликован в 1981г. На рисунке 1.8. представлены основные шаги данного метода:

Рисунок 1. 8 - Семь шагов методологии мягких систем

5 почему (5 Whys)

Представляет собой простую технику, позволяющую быстро выявить корень проблемы и изучить причинно–следственные связи, лежащие в ее основе. Была разработана в Японии в 1930-х годах. Для этого необходимо последовательно задавать вопросы “Почему?”. Каждый последующий вопрос задается к ответу на предыдущий. Первый вопрос обычно задается к проблеме и имеет вид “почему это произошло?”. Метод не имеет жестких ограничений и может быть применен для широкого круга задач, даже в тех случаях, когда в основе проблемы лежит несколько причин. Обычно ответ на последний вопрос (их может быть больше чем 5) ссылается на процесс, или поддающееся изменению поведение, которое происходит некорректно, что свидетельствует о нахождении первопричины. Часто может быть представлен в виде Fishbone Diagram или табличном формате.

Диаграммы сходства (Affinity Diagrams)

Представляет собой метод группировки знаний по определенным темам. Целью является выделение смысла из большого количества идей и группировки, последних по данному признаку (рисунок 1.9).

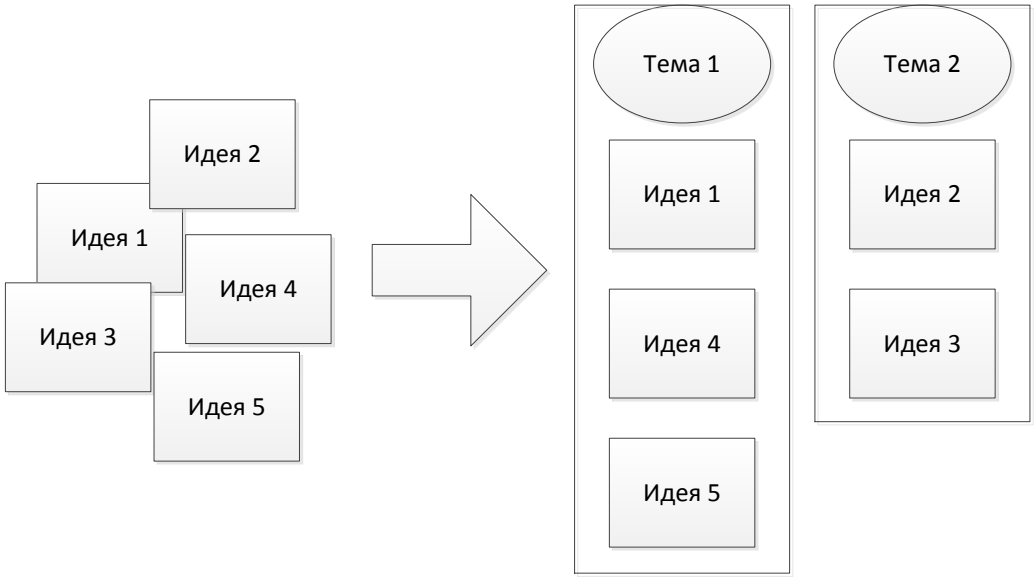

Рисунок 1. 9 - Группировка идей (знаний) по определенным темам (принципам) позволяет упростить и структурировать область задачи

Системная диаграмма

Данный вид диаграмм позволяет визуализировать взаимосвязь между различными частями (факторами) системы. В основе лежит представление, что между различными частями может наблюдаться прямая (simple same way) или обратная (opposite) корреляции. Прямую связь принято обозначать буквой **S,** обратную – буквой **O.** Диаграмма позволяет задавать несколько типов циклов (баланс, усиление), показывать влияние внешних факторов, показывать разрывы и временные задержки. На рисунке 1.10 представлена простая диаграмма баланса.

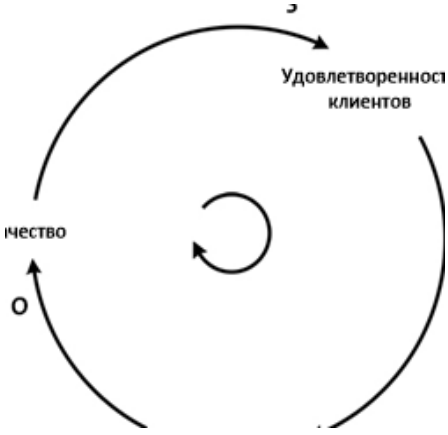

Рисунок 1. 10 - Пример цикла баланса

Данный вид диаграмм представляет зависимости между различными аспектами решения (системы) и позволяет показать:

1. Каким образом связаны факторы, что будет со вторым фактором при изменении значения первого;
2. Какие внешние факторы (и каким образом) могут влиять на остальные;
3. Какие задержки могут быть в системе;
4. Каким образом система функционирует в целом.

Древовидные диаграммы (tree diagram)

Представляют собой визуальное описание отношений, которые начинаются с корневого узла. Этот узел представляет собой проблему, которую необходимо решить, каждое возможное решение представляет отдельную ветку, которая располагается правее корневого узла. Те в свою очередь, могут быть декомпозированы на более мелкие уровни детализации, таким образом, диаграмма приобретает древовидную структуру (рисунке 1.11).

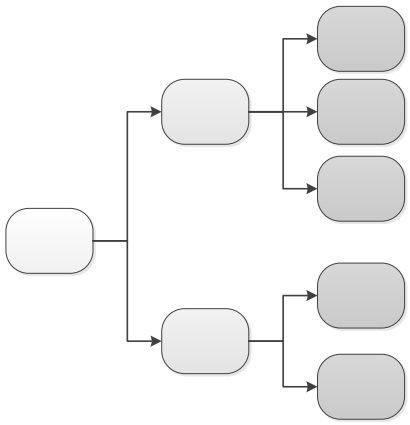

Рисунок 1. 11 - Пример древовидной диаграммы

Такие схемы позволяют упростить сложные проблемы (и их возможные решения) и позволяет легко обозревать задачу целиком. Таким образом, древовидные диаграммы позволяют:

1. Выделить шаги, которые необходимы для решения проблемы (реализовать план)
2. Представить истинный объем проблемы
3. Разбивать шаги на под шаги и т.д.
4. Использовать brainstorm для создания моделей.

Существует несколько разновидностей таких диаграмм:

1. Деревья решений. Широко используются в области машинного обучения, позволяют делать правильный выбор при наличии нескольких изменяемых параметров;
2. Деревья предсказаний. Широко используется в теории игр для предсказания того, как результат соревнования зависит от различных ситуаций;
3. Fishbone diagram. Представляют собой горизонтальную древовидную диаграмму. Позволяют определять узкие места в системе и анализировать, почему процесс работает не так как полагается.
4. Mind Mapping. Используется для представления большого количества знаний в визуальном виде (для удобства восприятия)

Анализ компьютерных систем поддержки решения задач

Основной целью систем визуальной поддержки решения задач является описание целей, предметной области, процесса решения с помощью визуальных моделей, для вовлечения всех заинтересованных сторон.

Рассмотрим некоторые системы визуальной поддержки процесса решения задач.

ДРАКОН схемы. Система “Графит-Флокс”

В настоящий момент наблюдается тенденция к увеличению сложности программных систем. Из этого следует, чтобы нагрузка на разработчика также увеличивается, ему приходится держать больше информации в голове.

Увеличение нагрузки на мозг разработчиков приводит к возрастанию вероятности возникновения ошибок и сбоев, и в конечном итоге, приводит к увеличению стоимости разработки. С другой стороны, узким местом при разработке сложных систем становится не вычислительные, а человеческие ресурсы. Стоимость вычислительных ресурсов с каждым годом все уменьшается, тогда как стоимость человеко-часов постоянно растет. Поэтому задача экономии человеческих ресурсов становится доминирующим в современной индустрии разработки программного обеспечения. На первый план встает проблема производительности систем человек компьютер, где человек является наиболее непредсказуемым и подверженным сбоям подсистемой.

Когнитивные факторы, под которыми часто понимают мыслительные, творческие и другие аспекты разработчиков, становятся основными мерами при выборе того или иного средства или технологии. Существует наука, изучающая такие факторы, она называется эргономика. В рамках этой науки изучаются проблемы понимания человека. Только немногие технологии разработки программного обеспечения учитывают креативные особенности человека. Большинство из них эргономики не уделяет достаточного внимания, например, нет стандартов или рекомендации относительно того насколько большими или детальными должны быть модели. Хотя известен тот факт, что человеческий мозг в состоянии удержать и оценить лишь 7 ± 2 объекта. Аналогичная ситуация наблюдается и для средств, которые используются при разработке программного обеспечения. Существует модель GOMS, которая позволяет оценить, сколько времени необходимо для выполнения определенных действий в приложении. В ней используется эмпирическим путем вычислений параметры, оценивающие время, необходимое обычному человеку для реагирования на определенные события интерфейса, выполнения определенных действий (такие как перемещение мыши и нажатия кнопки) и так далее. Ориентируясь на современные IDE,

можно сделать вывод, что далеко немногие из них спроектированы с использованием вышеописанных принципов. На этапе тестирования проблемы эргономики стоит не менее острым, так как этот этап обычно занимает 40-60% временем. Тестирование является достаточно важным этапом разработки программного обеспечения. Поэтому, необходимо говорить о покрытие тестами наиболее вероятных вариантов использования системы (у большинства пользователей система должна работать корректно).

Все эти факторы побудили разработчиков системы «Графит-Флокс» к разработке языка (графического), учитывающего когнитивные факторы.

Формально состоит из 2 половинок, представленных на рисунке 1.12.

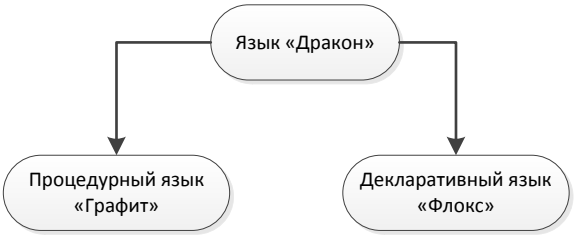

Рисунок 1. 12 - Графит и Флокс представляют собой «половинки» языка ДРАКОН

Основой технологии является два языка «Графит» и «Флокс». «Графит» является процедурным языкам и необходим для описания последовательности шагов в алгоритме. «Флокс» используется для описания словаря или онтологии в рамках решаемой задачи. Пример алгоритма, записанного на языке «Графит» представлен на рисунке 1.13.

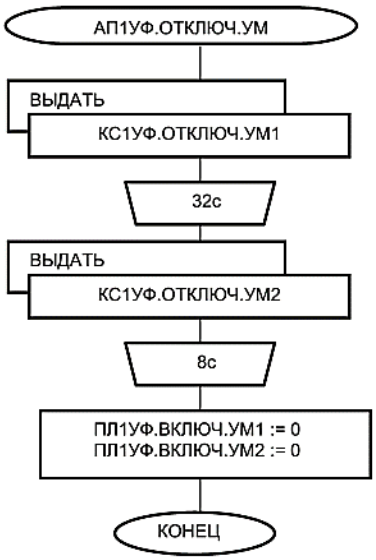

Рисунок 1. 13 - Пример реального алгоритма, записанного на языке «Графит»

«Графит» является визуальным языком программирования, в котором программа рисуется в специализированном графическом редакторе. Такая модель, с одной стороны является легкой для понимания, а с другой может быть основой для коммуникации с другими заинтересованными сторонами не знакомыми с программированием. Второй «половинкой» языка «ДРАКОН» являются «Флокс» таблицы. Они применяются для описания понятий, которые используются в процессе решения задачи. «Флокс» таблица представляет большую «корзину» для мелкого мусора. Основными задачами системы «Графит-Флокс» являются:

1. Повышение производительности труда всех заинтересованных сторон
2. Сокращение сроков разработки системы, уменьшение вероятности возникновения ошибок.
3. Минимизация затрат на поддержку системы
4. Вовлечение всех заинтересованных сторон, путем использования простых визуальных моделей

Язык «ДРАКОН»

Язык “ДРАКОН” или “ДРАКОН схема” являются развитием классических блок-схем. Целью создания языка было увеличение наглядности и степени понимания алгоритмов, заложенных в эти модели. На первое место ставится эргономика, вводятся определенные правила расположения элементов модели (правило «шампура» и правило расположения более редких путей алгоритма дальше от центра). Дракон схема может содержать порядка 25 икон и около 20 макро икон - составных блоков будущее схемы. Типовые иконы языка дракон представлены на рисунке 1.14:

Рисунок 1. 14 - Типовые иконы языка «ДРАКОН» [95]

Основные правила построения «ДРАКОН схем»

Правило: «чем правее, тем позже». Это правило означает, что если ветка расположена правее, то она выполняется позже или реже, чем ветка, расположенная левее. Несмотря на свою простоту, это правило является очень важным при изучении схемы и тестировании, когда перед тестировщиком встает вопрос какие варианты необходимо тестировать в первую очередь, а какие можно протестировать позже. Данное правило

отвечает на этот вопрос - необходимо тестировать сначала ветки, которые ближе к «шампуру», за тем расположенные правее.

Правило «шампура». Икона «начало» и икона «конец» должны находиться на одной линии («шампуре»). Это правило является чисто эргономическим и вытекает из того, что уменьшение количества изломов делает модель более понятной для восприятия. Под «шампуром» понимается путь, который ведет к наибольшему успеху и является основным. Согласно предыдущему правилу, данный маршрут будет протестирован в первую очередь. Следование этому правилу позволяет значительно сократить количество изломов (рисунок 1.15).

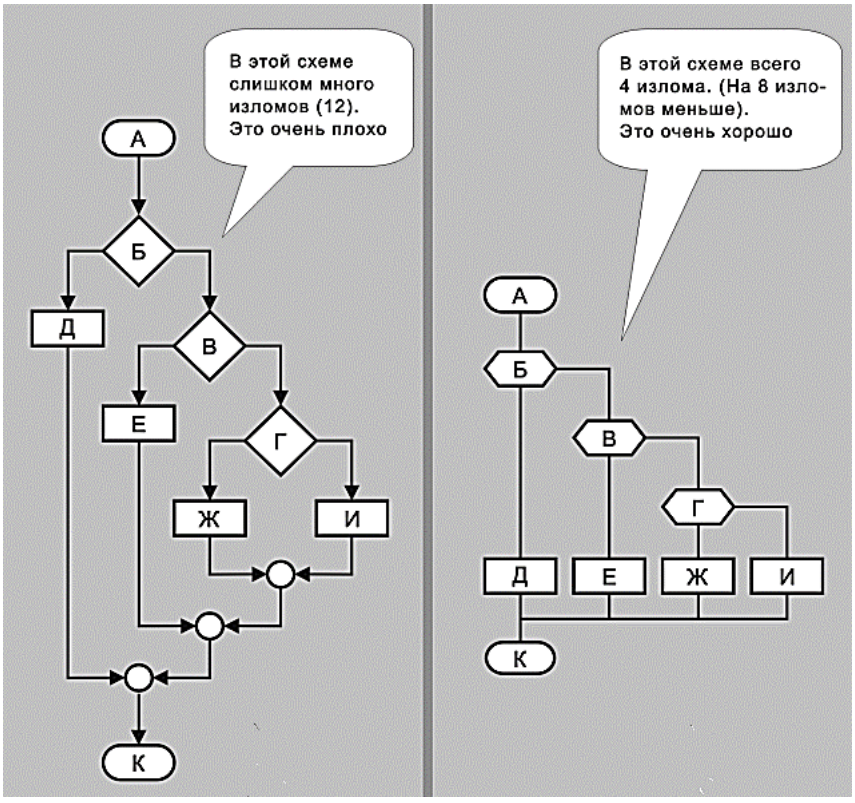

Рисунок 1. 15 - Значительное сокращение количества изломов, за счет следования правилу “шампура” [95]

Правило «запрет пересечений». При пересечении линий наглядность модели снижается, поэтому при создании «ДРАКОН» схем рекомендуется избегать их. Это обеспечивается путем правильной расстановки элементов и объединение схожих путей.

Правило «запрет повторов». Это делается путем объединения одинаковых вершин. Благодаря этому правилу значительно сокращает одинаковые блоки, увеличивается наглядность. Рекомендуется объединять не любые одинаковые иконы, а только соседние, так как в этом случае легко избежать пересечения связей.

Правило «7 ± 2». Рекомендует располагать на модели не более 7 ± 2 вершины. Остальные группируются и превращаются в одну икону, которая, при необходимости, может быть развернута.

К недостаткам платформы следует отнести закрытость системы (а следовательно ограниченное количество информации о ней) для широкого использования, поддержка только одного вида графических моделей.

Intentional Platform

Является одним из самых продвинутых инструментов, проектом руководит Чарльз Симони, известный по линейки продуктов от Microsoft. Является средой, позволяющей работать экспертам предметной области совместно с программистами. Типовой процесс разработки представлен на рисунке 1.16. Основная критика заключается в скрытности работы Intentional Software [14]. Первые публикации об инструментах относятся к 2009 году [46, 56].

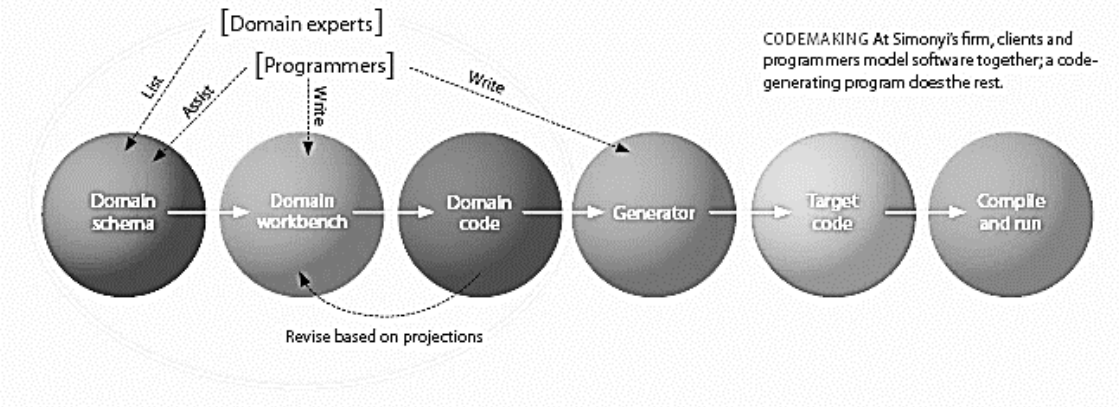

Рисунок 1. 16 - Типовой процесс в Intentional Platform

В инструменте программное обеспечение рассматривается с позиции:

$$Software = Input + Process$$

Под входом понимаются знания той предметной области, в которой разрабатывается программное обеспечение (этими знаниями обладают эксперты предметной области), а под процессом понимают объединение знаний о предметной области (domain knowledge) с процессом разработки программного обеспечения (Software Engineering Knowledge). Такое смещение процесса разработки от программистов в сторону экспертов предметной области, по мнению авторов, является закономерным и оправданным [31].

Данная платформа позволяет запускать разрабатывать Intentional приложения. Как и в других, технология обработки, таких как обработка естественного языка или обработка графической информации, данное средство позволяет преобразовывать входную информацию в полезные знания. Знание о предмете области фиксируются экспертами в специализированном редакторе с помощью предметно ориентированных языков. Его интерфейс похож на пакет Microsoft Office и позволяет работать с таблицами и графиками, встроенными элементами управления, такими как выпадающие списки, флажки и многое другое. После регистрации знаний они проверяются на полноту, и непротиворечивость, проходят встроенные тесты и в конечном итоге могут быть трансформированы в новое приложение или продукт.

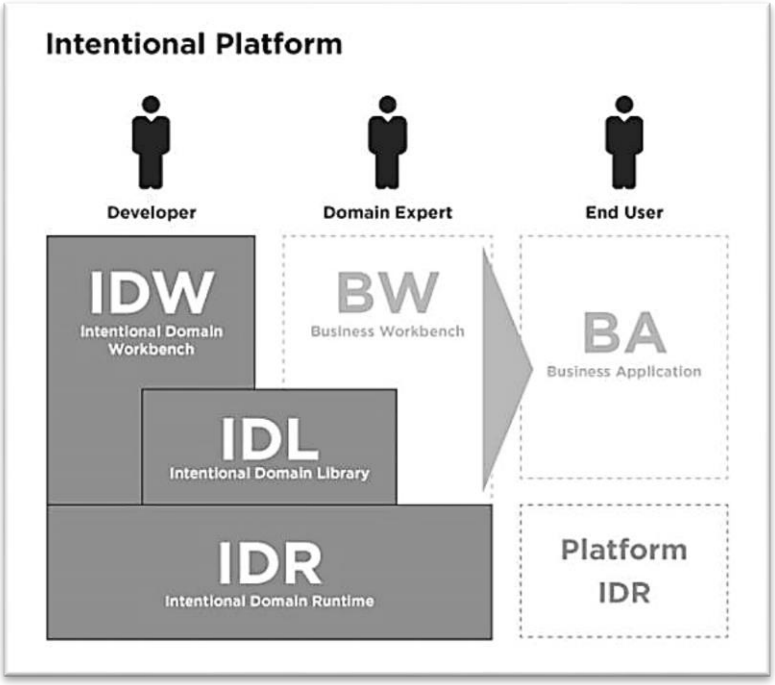

Рисунок 1. 17 - Intentional Platform ориентирована на экспертов и позволяет создавать приложения без обычного программирования [22]

Платформа состоит из 3 основных частей: Intentional Domain Workbench, Intentional Domain Library и Intentional Domain Runtime (рисунок 1.17).

Intentional Domain Workbench

Представляет собой интегрируемую среду разработки приложений, которые могут быть запущены на Intentional – платформе. Среда не только поддерживает различные DSL [10] для описания схем, трансформаций, проекций, валидаций и других артефактов, но также позволяет строить обширную среду с богатыми возможностями изменения конфигурации. Важной отличительной особенностью среды является одновременная поддержка нескольких проекций для одной семантической модели. Также существует возможность переключения между проекциями. Другими словами, неважно на каком языке описывается предметная область, важно, как она описывается (рисунок 1.18). Кроме поддержки множества языков среда имеет свой собственный язык CL1, который является подмножеством

C#. Intentional Workbench построен на базе Net Framework и большая его часть написана на C#. В связи с этим среда может легко генерировать сборки, которые будут автоматически выполняться в Net окружении. Интересной особенностью среды является то, что для представления трансформаций используется много простых трансформаций, которые выполняются последовательно, а не одна большая трансформация.

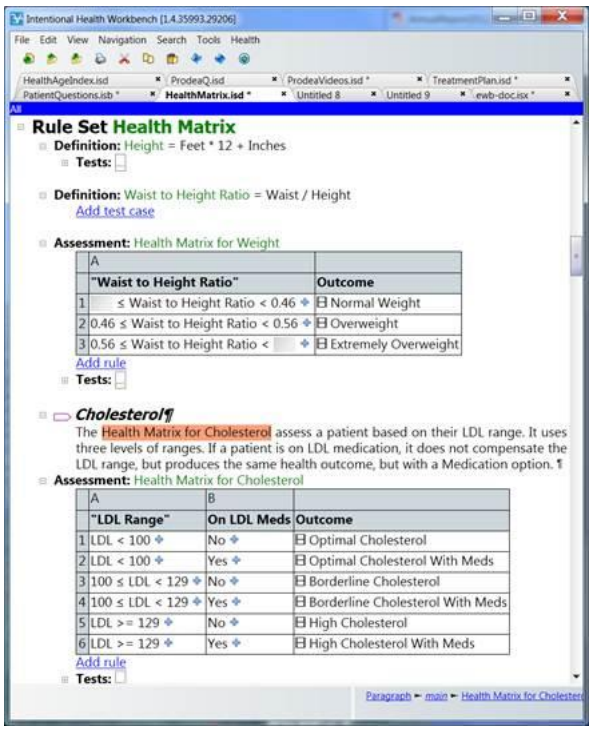

Рисунок 1. 18 - Данная среда позволяет представлять данные на естественном языке, в виде таблиц, рисунков, формул и т.д. [46]

Одной из главных проблем средств, которые работают с большим количеством моделей, является необходимость в контроле версий. “Intentional Domain Workbench” содержит встроенные механизмы контроля версий, которые логирует все изменения в “Intentional Tree” и позволяют делать фиксацию версий и объединение различных уровней дерева. Как и в большинстве средств управления, версиями существует возможность просматривать отличия между различными версиями деревьев.

Данная среда обладает следующими преимуществами:

1. Документы могут быть легко трансформированы

2. Любая нотация для описания знаний может быть принята
3. В описание предметной области закладывается возможность развития
4. Единая среда разработки и управлением версий
5. Разработка в среде включает следующие этапы:
6. Эксперт предметной области и программист договариваются относительно формата представления знаний (схемы)
7. Они вместе создают схему
8. Программист должен понять, что программа должна делать и как
9. Эксперт поддерживает код предметной области
10. Программист поддерживает реализацию через генерацию и «реализует» нотацию, используемую экспертом предметной области.

Intentional Domain Library

Представляет собой библиотеку поддерживаемых языков, которые используются при разработке приложения. Включает в себя:

- Языки программирования: C#, Ruby, Java и т.д.
- Frameworks: Rails, Net и т.д.
- DSL: SQL, HTML, BPEL/BPMN и т.д.
- Языки моделирования: UML, SysML, Modelica, SimuLink и т.д.
- Форматы файлов: Excel, Microsoft Word, PowerPoint, XML, CSV и т.д.

Intentional Domain Runtime

Представляет собой среду, в которой Intentional-приложение могут исполняться. Включает в себя:

- Intentional Tree. Единая структура, которая используется для представления всех знаний и метазнаний о предметной области и разрабатываемом приложении

- Transformation Pipeline. Представляет собой инструмент для управления обратимыми преобразованиям над Intentional Tree.
- Projection Editor. Редактор, который обеспечивает простой и интуитивно понятный эксперту предметной области интерфейс для редактирования знаний о предметной области.
- Groupware. Среда, осуществляющая контроль версий, позволяет осуществлять работу над ресурсами, поддерживает алгоритмы разрешения конфликтов.

Платформа обладает наиболее широкими возможностями (из представленных в обзоре систем) по концептуальному экспериментированию, фиксированию знаний, прототипированию. К недостаткам следует отнести закрытость платформы, ограниченное количество документации и примеров практического использования.

Activiti

Activiti – это framework на языке java для управления потоками работ [3]. Он включает в себя компоненты, позволяющие использовать его как самостоятельный продукт для, создания потоков работ в графическом режиме (web + eclipse plugin + Activiti modeler и т.д.), программирования (eclipse plugin), управления потоком работ (web), моделирования (в том числе моделирования исполнителей) и, как средство, для исполнения потока работ (рисунок 1.19).

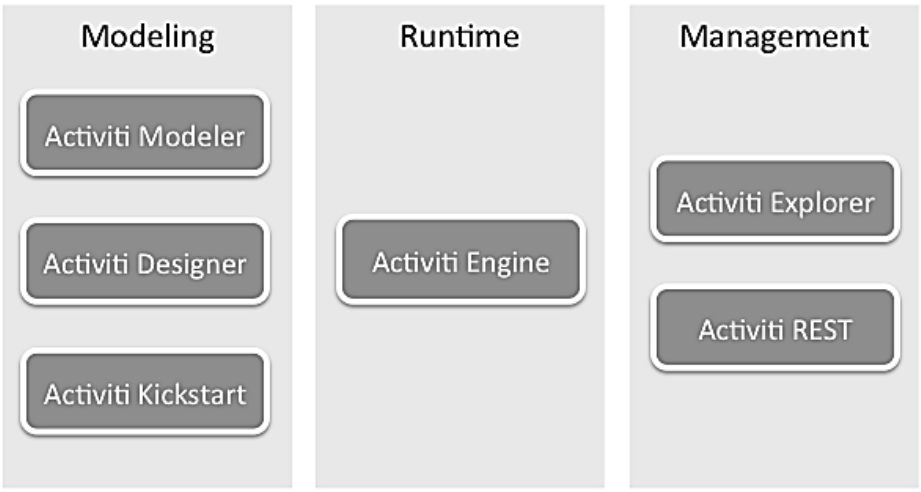

Рисунок 1. 19 - Структура framework Activiti[3]

Система очень универсальная и обладает возможностью интеграции с такими системами как Jboss [24] и Alfresco[5]. Можно выделить основные преимущества и недостатки:

Преимущества:

- добавление в поток как UserTask (задачи, выполняемые человеком), так и ComputerTask;
- возможность программирования задачи в виде бизнес правил на «естественном языке» Drools Rule Language (DRL) [11, 12];
- возможность программирования задачи на различных языках, удовлетворяющих JSR 223(также DSL);
- получение различной статистической информации о задачах и потоках (отчеты через Activiti Explorer);
- автоматизация тестирования потоков работ;
- поддержка интеграции с большим количеством систем (jboss, Alfresco и т.д.);
- тесная интеграция с web-технологиями.

Недостатки:

- Отсутствует возможность моделировать поток с пошаговым просмотром состояний. В настоящий момент не поддерживается.

- Отсутствует возможность моделировать поток с поиском узких мест (в том числе моделирование UserTask). Отсутствует соглашение среди разработчиков (на основе каких данных моделировать UserTask).
- Отсутствует возможность контроля доступа на уровне задач. Если модель доступна, то она доступна полностью.
- Не поддерживается «умное» назначение задач исполнителям (task assignment)

jBPM

jBPM (Business Process Management (BPM) Suite) – система, позволяющая реализовывать потоки работ от компании JBoss [25]. Версия продукта, которая будет рассмотрена в данном обзоре – 5.4.0 final. Средство является open-source, также позволяет выполнять бизнес процессы, описанные в нотации BPMN 2.0 (либо на jPDL).

Модели бизнес процессов позволяют описать шаги необходимые для достижения поставленных целей с помощью блок схем (BPMN 2.0) (рисунок 1.20). описанная модель может быть представлена как web приложение (имеет возможность редактирования workflow через браузер) и как framework для программного управления (или создания, например, через плагин к eclipse [21]) потоком. Основным языком является java.

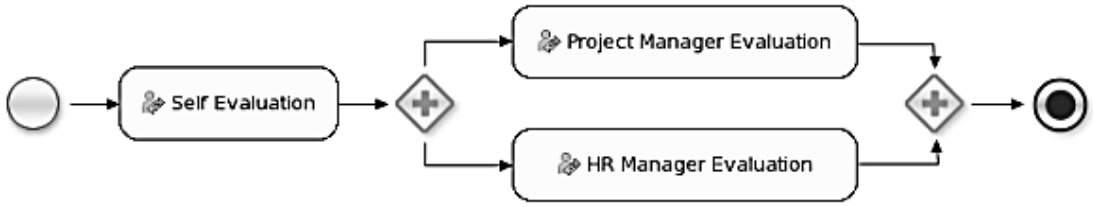

Рисунок 1. 20 - Пример бизнес процесса в JBPM (eclipse plugin)

Система позволяет включать в рабочий поток задачи, которые должен выполнять человек. Такие задачи называются UserTask.

Преимущества:

- Обладает более широким функционалом, чем Activiti, хотя они во многом похожи;
- Процесс проектирования и создания (программирования) системы может быть целиком реализован в рамках данной платформы с помощью workflow. При проектировании большей частью node будут иметь тип UserTask, тогда как при программировании ScriptTask;
- В отличие от Activity позволяет просматривать состояния workflow.

К недостаткам рассмотренных систем можно отнести узкую специализацию, связанную с управлением потоков работ, поддержку ограниченного числа моделей, отсутствие возможности перевода бизнес правил в код, приближенный к естественному языку.

Системы на базе Scientific workflows

Представляют собой серию программных систем, предназначенных для создания, выполнения потоков работ преимущественно при проведении научных опытов. «Scientific workflows» - является широко используемой парадигмой для описания и управления проведением сложных научных опытов [74]. В настоящий момент большинство из них сгруппированы вокруг среды совместных исследований myExperiment. Она была введена в 2007 году и представляет собой хранилище большого количества workflow экспериментов в различных областях науки [48]. Поддерживает большое количество различных систем управления потоками, такие как Taverna, Trident, Galaxy. Упрощенным примером, который приводится на сайте разработчиков Taverna, является графическое представление потока для определения температуры в определенном городе[71] (рисунок 1.21).

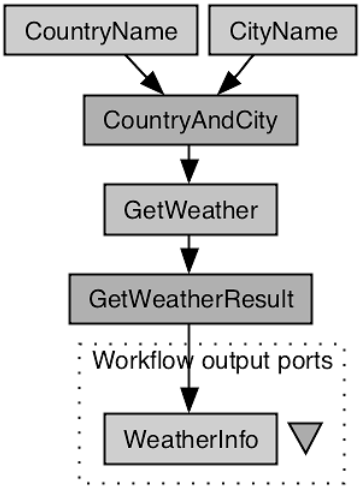

Рисунок 1. 21 - Упрощенный пример Scientific workflows

Реальные примеры, которые являются более сложными, обычно имеют ряд входных и выходных параметров. В качестве элементов потока Taverna позволяет использовать большое количество видов программных компонентов. Это могут быть различные удаленные или локальные веб сервисы, скрипты (java и т.д.), базы данных. Реальный пример потока, анализирующего последовательность белков можно посмотреть в [35].

По результатам анализа была построена таблица сравнения компьютерных систем с точки зрения учета когнитивных факторов (таблице 1.2.).

Таблица 1.1 - Сравнение программных средств

Программное средство	**Поддерживаемые графические нотации**	**Возможность концептуального экспериментирования**	**Уровень автоматизации поиска логических, фактических и т.д. ошибок в созданных моделях**	**Возможность работы с образно-семантическом и программном представлении одновременно**
Разработанное средство на базе OwnWIQA	Изобразительное представление (все нотации, для которых есть палитры), декларативное, концептуально-алгоритмическое (диаграммы активностей, вариантов использования, классов)	Динамическая визуализация (изобразительное представление), исполнение в пролог и псевдокодовом интерпретаторе	Автоматический/ автоматизированный (поиск логических и фактических ошибок)	У декларативного, концептуально-алгоритмического и изобразительного представлений есть графическая и согласованная с ней текстовая (вопросно-ответная) версия
Графит-Флокс	Процедурный язык «Графит» и декларативный язык «Флокс»	Присутствует	Нет информации	Работа происходит только в графическом режиме
Intentional Platform	UML, SysML, Modelica, SimuLink	Нет информации	Нет информации	Присутствует. Эксперт предметной области может структурировать информацию как в графическом, так и в текстовом виде
Activiti	BPMN 2.0	Отсутствует	Автоматический поиск синтаксических ошибок. Поддержка поиска других типов ошибок отсутствует.	Только графическое представление
jBPM	BPMN 2.0	Присутствует	Автоматический поиск синтаксических ошибок. Поиск других типов ошибок за счет возможности исполнения моделей	Только графическое представление

1.2. Основы сопровождение процессов решения задач

В настоящее время в разработке систем, интенсивно использующих программное обеспечение, набирает популярность подходы, в которых модели (преимущественно графические) становятся основными артефактами процесса разработки и все остальные (код, документация и т.д.) генерируются автоматически из них. Такие модели сильно отличаются от рисунков, так как удовлетворяют определенным синтаксическим правилам и содержат дополнительную семантику, выраженную через расположение элементов, используемых графических нотаций и т.д. Моделирование является гораздо более сложной деятельностью и предоставляет огромный набор дополнительных преимуществ: проверки синтаксиса, применение преобразований, исполнение и отладка моделей и т.д.

По мнению автора, основные положения данной методологии могут быть также эффективно применены и в процессе решения задач, так как эти процессы имеют много общего. Программирование, по своей сути, представляет собой процесс решения определенного рода задач (выбор и разработка архитектуры, реализации, учет возможного развития системы, отладка и т.д.), поэтому необходимо, ознакомится с основами таких подходов, известных как Model Based Engineering. В рамках MBE принято считать: "всё есть модель", т.е. все элементы будущего ПО могут быть представлены в виде моделей, в том числе и трансформации [30], которые являются движущей силой процесса разработки и позволяют уточнять (изменять) созданные модели.

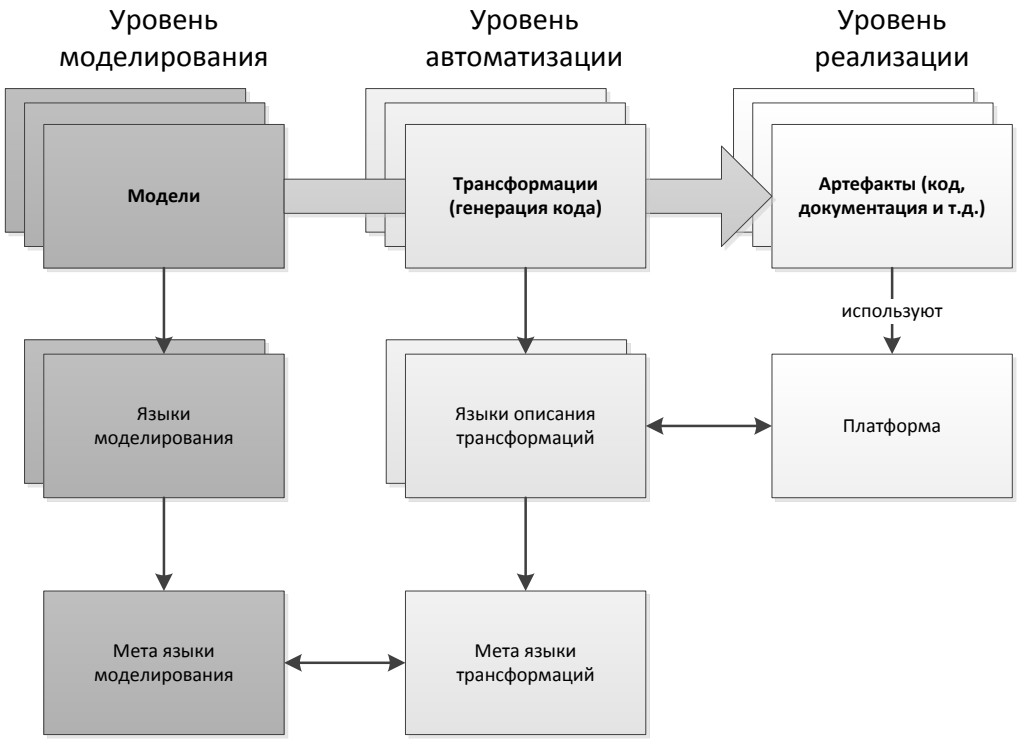

Рисунок 1. 22 - Основные сущности процесса разработки на основе моделей и связи между ними

Основное направление процесса разработки на основе моделей направлено от концептуального уровня [6, 7] (более абстрактного) к уровню реализации (менее абстрактного), посредством применения набора трансформационных правил (рисунок 1.22). Эти три уровня (концептуальный уровень, уровень автоматизации и уровень реализации), составляют ядро и описываются с помощью определенных нотаций. В качестве основы языка выступает метаязык, который тесно связан с правилами трансформаций (они создаются для определенных нотаций). Реализация, в свою очередь, сильно связана с платформой, и поэтому должна опираться на некоторую модель платформы, так же как языки трансформации опираются на нотации. Из этого следует, что MBE требует наличие множественных моделей, так как они позволяют с разных сторон взглянуть на разрабатываемую систему (или ее части).

Другим ключевым компонентом MBE являются трансформации, которые позволяют описывать преобразования. Трансформации обычно описываются на мета модельном уровне, и затем применяются на уровне моделей, позволяя преобразовывать исходную модель (source model) в целевую (target model). Трансформации обычно описываются на DSL [39], и представляют собой 2 шаблона: шаблон поиска и шаблон замены. Шаблон поиска представляет описание вершин, связей и другой информации. При нахождении шаблона поиска в моделях, он будет заменен на шаблон замены по определённому правилу.

Одним из главных преимуществ подхода MBE является автоматизация процесса разработки программного обеспечения, начиная с высокого уровня абстракции и заканчивая готовым исполняемым приложением (рисунок 1.23).

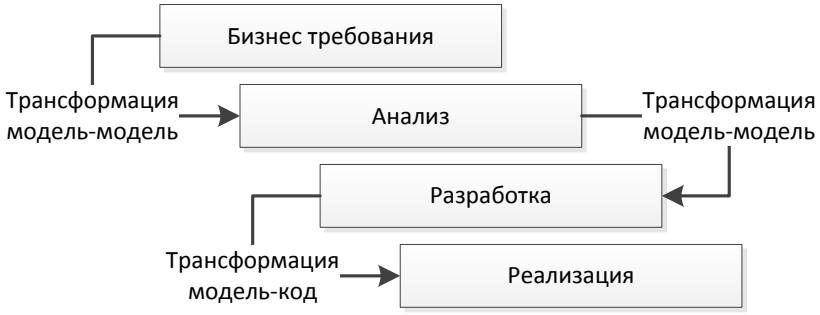

Рисунок 1. 23 - Процесс разработки с точки зрения MBE

MBE подразумевает уточнение исходных моделей, созданных аналитиками и понятных заказчику, до этапа получения исходного кода благодаря чему, контроль процесса разработки доступен всем заинтересованным сторонам. В классическом процессе, артефакты, созданные на этапе анализа требований, не участвуют напрямую в разработке, что может приводить к плохим результатам, когда поведение системы не соответствует ожидаемому.

Еще одним аргументом в пользу применения моделей является сокращение рутинных операций, код для которых может быть сгенерирован автоматически. Исследования показывают, что операции создания, чтения,

обновления и удаления занимает порядка 80% функциональности в типичных ориентированных на данные приложениях [29].

Определение процесса решения задач

Под процессом решения задач следует понимать такие мыслительные операции, которые направлены на достижение целей, заданных в рамках задачи. Такой процесс возникает, если субъект сталкивается с ситуацией, когда он не в состоянии использовать предыдущий опыт для нахождения решения. С точки зрения когнитивного подхода процесс решения задач является наиболее сложной из всех функций интеллекта и определяется как когнитивный процесс более высокого порядка [91, 27]. Детальное исследование этого процесса выходит за рамки САПР и служит предметом исследования психологии и когнитивных наук, но без понимания основ такого процесса невозможно разработать эффективную систему поддержки решения задач. Большой вклад в этой области внес Карл Дункер [27, 79] - немецкий психолог, представитель гештальтпсихологии, занимающийся исследованием процесса продуктивного мышления и решения задач. Основным положением его исследований является то, что между моментом, когда задача появляется и моментом, когда достигается окончательное решение, существует ряд определенных стадий, организованных в определенном порядке (от обобщенного решения к более конкретному решению). В упрощенном виде процесс решения согласно Дункеру можно разбить на следующие шаги:

1. Процесс понимания проблемной ситуации, понимание внутренних связей, восприятие задачи как целое, заключающее в себе некий конфликт.
2. Нахождение "функционального значения" решения, которое, по словам Дункера, “находится на основе внутренних и очевидных связей с условиями проблемной ситуации” [91].

3. Конкретизация функционального значения с целью нахождения решения. “Понять какое-либо решение как решение — это значит понять его как воплощение его функционального значения” [79]. Поэтому окончательное решение будет иметь вид функционального значения, примененного к конкретным данным.
4. В случае если решение окажется неудовлетворительным, субъект осуществляет возврат, чтобы отыскать другое решение. Так как субъект является активной силой, то при необходимости на этом этапе может происходить адаптация задачной ситуации (изменение или дополнение), с целью упрощения или изменения целей.

Процесс отыскания решения можно представить, как иерархическую последовательность операций конкретизации, с возможными возвратами. Кроме того, процесс решения (а также результат) зависят от накопленного субъектом опыта (прецедентов) и не носят случайный характер. Также на каждой фазе решения ставится вопрос о причинах конфликта, позволяющий глубже проникнуть в природу конфликта и приблизиться к решению. Дункер называет это “анализом конфликта”. Параллельно процессу конкретизации может идти процесс перемещения между различными функциональными значениями. По мнению Дункера, субъект ищет «в рамках прежней постановки вопроса другой зацепки для решения» или уточняет саму постановку вопроса [79]. Параллельно с анализом конфликта, может происходить и анализ цели, который может приводить к обобщению цели (или даже к ее переосмыслению).

Понятие образ имеет большое количество определений, но в основе большинства из них лежит определение как "форма представления чего-либо". В данном случае об определенной форме представления задачи в виде (преимущественно) графических изображений (в англоязычной литературе слово образ часто переводится как Image - изображение, картина). Часто понятие образа противопоставляется понятию вещи, причем оно может, как

предшествовать (прообраз, прототип), так и создаваться на основе объекта. В любом случае следует понимать, что образ находится в изоморфном отношении с объектом, и всегда описывает ограниченное количество свойств оригинала. В данном контексте понятие образа созвучно понятию графической модели. Графические модели могут представляться в форме пригодной для построения формальной математической модели (и разбора согласно грамматике), то есть в форме языка программирования. В этом случае можно говорить о некой семантики (предании смысловых значений составляющих модель конструкций), тем самым вовлекать средства исполнения, верификации, трансформации и т.д. Под процессом сопровождения понимается процесс поддержки субъекта на различных этапах решения задач. При анализе понятия "процесс решения" было выявлено, что процесс содержит несколько этапов, каждый из которых обладает определенной спецификой. Поэтому процесс сопровождения можно разбить на несколько этапов согласованных с процессом решения.

Таким образом, можно сделать следующие выводы, которые будут учитываться при проектировании системы поддержки решения задач на основе моделей:

1. Процесс решения задач является наиболее сложным из функций интеллекта. Из процесса решения можно выделить несколько этапов:
2. Этап понимания проблемной ситуации в целом (необходимо представление системы как целого состоящего из различных элементов и связей).
3. Этап нахождения, обобщенного "функционального значения". Акцентирование на различных функциональных зависимостях между элементами задачи (решения).
4. Применение "функционального значения" к конкретным данным задачи. Получение окончательного решения.

5. В случае не успешности полученного решения - переосмысление полученных результатов (анализ цели, анализ конфликта и т.д.), возвращение к предыдущим этапам решения.
6. Оценивание решение задачи и формирование единицы опыта (прецедента) в различном виде (например, в виде логической модели). Дункер не выделяет этот этап как самостоятельный, так как он выходит за рамки решения задачи. Но в наших исследованиях этап формирования базы опыта является важным процессом, так как разрабатываемые средства будут ориентированы на повторное использование опыта.
7. При разработке системы поддержки решения задач нельзя игнорировать этапы решения, которые рассматриваются в когнитивных науках.
8. Процесс сопровождения должен быть привязан к этапу решения задачи и учитывать их специфику.
9. Реализации сопровождения на основе графических моделей, которые строятся на основе определенной семантики (правил) позволит управлять процессом решения.

1.3. Обобщенная постановка задачи

На основе проведенного анализа необходимо сформулировать обобщенную постановку задачи, которая имеет следующий вид:

Z. 1. Разработать комплекс средств, обеспечивающих пошаговое формирование постановки проектной задачи, состояние которой представляется взаимосвязанной совокупностью текстовых, графических и концептуально-алгоритмических моделей.

Z. 2. В основу пошагового формирования положить механизмы разработок, управляемых моделями, в процессе которых положить итеративное согласование текстов с графикой и алгоритмической

составляющей, нацеленное на обеспечение необходимого понимания и повторного использования.

Z. 3. Разработка должна осуществляется в инструментально моделирующей среде WIQA, так чтобы построенный комплекс средств был включен в нее как расширение.

Вопросно-ответный анализ обобщенной постановки задачи

Формулировка задачи представлена в виде, который необходим для применения вопросно-ответного анализа. Данный вид анализа принято начинать с извлечения вопросов.

Из обобщенной постановки задач следует, что основной акцент в работе делается на разработку образно-семантические моделей, которые должны быть типизированы и специфицированы так, чтобы они были согласованы с нормативной моделью прецедента, допуская полезные преобразования между типами и переходы к их программным представлениям.

Рассматривая Z.1. как источник неопределенности, можно выделить следующие составляющие неопределенности: «задача», «состояние задачи», «постановка задачи», «ряд моделей, специфицирующих состояние задачи». Из этих составляющих логично, в первую очередь, рассмотреть специфику тех задач, для которых разрабатываются средства поддержки, то есть начать пошаговую детализацию с вопросов:

Q.1. С каким классом задач в диссертации будут связаны ее теоретические и практические рекомендации и решения?

Q.2. Что подразумевается под состоянием задач?

Q.3. Каков жизненный цикл задач, рассматриваемых в диссертации?

Q.4. Каким образом происходит формулирование и решение задач?

Q.5. Каким образом происходит согласование текстов с графикой и алгоритмической составляющей?

В рамках исследовательской группы, в которой выполнялась диссертация, особый интерес представляют задачи разработки систем автоматизированного проектирования. В процессе проектирования таких систем коллектив проектировщиков формулирует и решает задачи различных классов для достижения поставленных целей. При разработке АС, как и при разработке других сложных систем, существует класс задач, которые появляются неожиданно и требуют скорейшего решения. Для таких задач важным является как можно быстрее разобраться с проблемой (а возможно и с системой, в которой возникла проблема) и решить ее за ограниченное время. Поэтому, ответом на Q 1. является:

A. 1. В диссертационных исследованиях делается акцент на проектных задачах, которые появляются неожиданно и требуют скорейшего решения.

В тоже время ответ на вопрос A. 1. порождает следующий ряд неопределенностей-вопросов:

Q. 1.1. Кто занимается решение проектной задачи, которая появляется неожиданно в процессе проектирования?

В процессе решения проектной задачи основная роль отводится коллективу проектировщиков во главе с руководителем. В некоторых случаях для решения задач используются экспертные системы, которым может быть отведена важная роль. В небольших организациях часто руководитель занимается постановкой задачи, тогда как решением обычно занимается один исполнитель. В таких случаях говорят об индивидуальной активности. В связи с этим ответом на вопрос Q. 1.1 будет:

A. 1.1 Решением проектной задачи занимается коллектив проектировщиков во главе с руководителем (коллективная деятельность). В некоторых случаях в процесс решения вовлекаются экспертные системы. В небольших группах решением занимается один исполнитель. В этом случае говорят об индивидуальной активности.

Ответ А.1.1. содержит неопределенность, а именно:

Q. 1.1.1. Чем отличается индивидуальная деятельность, направленная на решение проектной задачи, от коллективной деятельности?

Коллективная деятельность является намного более сложным образованием и включает не только активность каждого члена команды, направленную на решение поставленной задачи (или подзадач, из которых она состоит), но и активность между членами коллектива (руководитель-подчиненный и т.д.). Она обладает рядом преимуществ, по сравнению с индивидуальной деятельностью (параллельное выполнение, взаимозаменяемость сотрудников и т.д.), но требует наличия развитого набора средств, позволяющих решать задачу несколькими проектировщиками параллельно с возможность объединения полученных результатов. В процессе коллективной разработки программного обеспечения одной из важных проблем является необходимость наличие средств контроля версий исходного кода.

Ответом на вопрос Q 1.1.1 будет:

A 1.1.1 Коллективная деятельность является намного более сложным образованием чем индивидуальная, и включает в себя проблемы совместного использования ресурсов, разрешения конфликтных ситуаций и т.д. В рамках диссертации акцент будет делаться преимущественно на коллективную деятельность.

Важно отметить, что автоматизированная система, как достаточно сложное образование не имеет одного неизменного состояния, а постоянно эволюционирует во времени. Поэтому, задачи, которые появляются при проектировании АС (или при последующем развитии системы) также проходят определенные состояния, то есть подвержены некоторой динамике.

Таким образом, при анализе вопроса Q 2. можно выделить:

Q 2. 1. Каким образом меняется состояние задачи во времени (динамика)?

В популярных SCM под состоянием задачи подразумевается одно из следующих значений: «Открыта», «В процессе», «Решена», «Пере открыта», «Закрыта». Между этими состояниями существуют переходы. В совокупности они представляют собой жизненный цикл задачи (workflow).

В рамках диссертационного исследования, под состоянием подразумевается снимок (snapshot) артефактов (моделей, описывающих решение задачи), в определенный момент времени.

Так как задача развивается во времени и постоянно идет приращение новых знаний (например за счет использования метода пошаговой детализации) и сокращение неопределенности, то состояние задачи можно представить в виде формулы, где текущее состояние представляет собой предыдущее состояние и некоторое приращение ΔS

$$S(Z_i t_j) = S(t_{j-1}) + \Delta S_j$$

В свою очередь состояние задачи представляется состоянием моделей, которые описывают задачу (решение). В рамках диссертационного исследования (см. Z. 1.) выделяют совокупность текстовых, графических и концептуально-алгоритмических моделей. Именно состоянием этих моделей и определяется состояние всей задачи:

$S(Z,t) = S(T(t), G(t), C(t))$, где S – состояние задачи Z в определенный момент времени t

В свою очередь каждая модель находится в состоянии развития и изменяется во времени. Поэтому ответом на вопрос Q. 2. и Q 2.1. будет:

A. 2., A 2.1. Под состоянием задачи подразумевается совокупность состояний текстовой, графической и концептуально-алгоритмической моделей. Состояние задачи меняется во времени и может быть представлено в виде: $S(Z,t) = S(T(t), G(t), C(t)) + \Delta S$

Данный ответ порождает следующий вопрос:

Q. 2.1.1 Зачем необходимо представлять задачу различными моделями?

В настоящий момент наблюдается тенденция к увеличению сложности программных систем. Из этого следует, чтобы нагрузка на разработчика также увеличивается и ему приходится держать больше информации в голове. Увеличение нагрузки на мозг разработчиков - это увеличение вероятности возникновения ошибок, просчетов, что в конечном итоге приводит к увеличению стоимости разработки. На первый план выходят проблемы понимания (understandability). Проблемы понимания невозможно решить без вовлечения в процесс разработки когнитивных технологий и исследования того как человеческий мозг воспринимает и обрабатывает различного рода информацию.

Следует отметить, что в настоящее время большое количество исследований направлены на изучение возможности использования образов в программной инженерии [64]. Среди этих работ можно отметить: использование воображения в разработке программ [42], в совместной работе [41], а также использование образов в визуальном программировании [36] и в достижении понимания [62].

Человеческий мозг ориентирован на обработку графической информации. Это связано с тем, что большую часть информации об окружающем мире мы получаем с помощью зрения, и текст является лишь подвидом графической информации и не может быть выделен как самостоятельный вид. Принципиальным отличием текстовой информации от графической является доступность графической информации большинству людей, тогда как для понимания текста необходимо владеть определенным словарем или языком, на котором эта информация выражена. В мировой практике в международных командах, которые состоят из инженеров и программистов происходит отказ от текстовых форм представления

информации и переход к моделям или графическим записям. Обычно для выражения используют большое количество типов моделей.

Поэтому, ответом на вопрос Q 2.1 является:

А 2.1.1 Использование текстовых, графических и концептуальных моделей вытекает из принципа понимаемости (understandability), а также необходимо для активации феномена «mental imagery».

Феномен «mental imagery» может быть представлен как когнитивный процесс фильтрации поступающей информации (или поступившей ранее) и формированию образа, который может быть зафиксирован в виде графической модели. Без активации воображения проектная деятельность невозможна, так как проектировщикам особенно на концептуальном этапе проектирования, постоянно приходится «интеллектуально воображать», представляя сложившееся в процессе проектирования (или необходимые для процесса или его результата) сущности без опоры на информационные потоки, порождаемые органами чувств при взаимодействии с физической реальностью, то есть без опоры на естественное предназначение органов чувств.

Процесс пошагового формирования, который лежит в основе предложенного метода (пошаговой детализации в прецедентно-ориентированном решении), подразумевает итеративный процесс нахождения решения задачи путем манипулирования различными представлениями (текстовое, графическое, логическое, вопросно-ответное и т.д.), которое может, осуществляется как в прямом направлении, так и путем возврата на несколько шагов назад с целью корректировки. Отправной точкой для такого процесса может служить текстовая модель (техническое задание). Сформулированная в виде вопросов-ответов (после проведения вопросно-ответного анализа), такая модель может быть транслирована в графическую. Графическая модель находится в согласование с текстовой, поэтому такой переход является обратимым. Из-за этого становится

непринципиальным, какая модель (текстовая, графическая или концептуально-алгоритмическая) станет отправной точкой, формирование решения может происходить в одной из представленных проекций. Таким образом, ответом на вопрос Q. 4. будет:

> A. 4. Формирование решение задачи обычно начинается с вопросно-ответного анализа (построение текстовой модели), затем путем согласованных преобразований (в графическую или концептуально-алгоритмическую модели) и уточнений может быть получено решение. Не является принципиальным, какая модель является отправной точкой. Процесс формирования является итеративным и согласованным.

При ответе на вопрос A. 2. рассматривался «классический» жизненный цикл задачи, применяемых во многих workflow системах. Также говорилось, что задача может быть представлена в виде снимков состояний (snapshots) всех моделей, описывающих задачу в конкретный момент времени. Таким образом, целесообразно говорить о древовидной структуре жизненного цикла задачи. Вершинами в таком дереве являются снимки моделей, входящих в типизированный набор в момент времени t_j. В общем случае в дереве существуют циклы, или возвраты, которые схожи с понятием откат или сброс (rollback, reset) в системах контроля версий. Поэтому, ответом на вопрос Q. 3:

> A.3. Жизненный цикл задачи в общем виде представляет собой граф, вершинами которого являются снимки (snapshots) текстовой, графической, концептуально-алгоритмической моделей в конкретный момент времени t_j. В частном случае граф может вырождаться в дерево, но наличие обратных связей (циклов) необходимы для представления возвратов (rollback, reset), схожих с теми, что происходят в системах контроля версий.

При ответе на вопрос Q. 5. необходимо определить, что подразумевается под согласованием моделей. Согласование моделей – это процесс, при котором согласовываемые модели находятся в некоторой

зависимости и изменение одной из них влечет соответствующее изменения в зависимой модели. В методологии разработки на основе моделей под согласованием часто подразумевают процесс трансформаций моделей. Трансформация – это автоматическая или автоматизированная генерация целевых моделей из исходных, согласно заданному набору трансформационных правил, описывающих, каким образом конструкции языка исходных моделей могут быть представлены в языке целевых моделей. Такой процесс может быть обратимым и протекать в обоих направлениях, как от текстовых моделей к графическим (алгоритмическим), так и наоборот. Ручное изменение моделей также считается трансформацией с низким уровнем автоматизации. К такому виду можно отнести, например, уточнение моделей, когда в нее вносятся дополнительная информация, уточняющая модель. Детально процесс трансформаций, а также их виды будут рассмотрены во 2 главе. Таким образом, ответом на вопрос Q. 5. является:

A. 5. Согласование текстовой информации (моделей) с графикой и алгоритмической составляющей осуществляется за счет применения трансформаций (в терминах разработки на основе моделей) к исходным и целевым моделям. Такие трансформации могут осуществляться в обоих направлениях, и обладать различным уровнем автоматизации (ручной, автоматизированный, автоматический).

Мотивационно-целевая диаграмма

К числу важных результатов диссертационного исследования относятся положительные аспекты, которые можно получить от внедрения в практику предложенных методов. В данном случае в процесс коллективной разработки прототипа системы интенсивно использующих программное обеспечение (SIS). В связи с этим целесообразным является выделение ожиданий от разрабатываемых подходов, и затем, путем их анализа, выделение мотивов и целей, которые планируется достичь.

Для определения мотивационно-целевых установок, необходимо рассмотреть обобщенное представление задачи Z. 1. И сформулировать на его основе первый мотив:

М. 0. Снизить существующее существенное различие между естественным взаимодействием человека с физическим миром и его взаимодействием с компьютеризованными приложениями.

Мотив М. 0. связан с тенденцией к увеличению сложности программных систем. Из этого следует, что нагрузка на разработчика также увеличивается и ему приходится держать больше информации в голове. Увеличение нагрузки на мозг разработчиков - увеличение вероятности возникновения ошибок, сбоем расчетов, что в конечном итоге приводит к увеличению стоимости разработки. С другой стороны, узким местом при разработке сложных систем становятся не вычислительные, а человеческие ресурсы. Поэтому на первый план встает проблема производительности систем человек-компьютер, где человек является наиболее непредсказуемой и подверженной сбоям системой.

Снизить эти негативные факторы можно путем учета когнитивных факторов при разработке средств поддержки проектирования САПР. Для этого необходимо анализировать разрабатываемое средство с помощью GOMS и учитывать эргономические и когнитивные факторы. Таким образом, цели могут быть сформулированы следующим образом:

Ц0. Обеспечить проектировщика средствами представление моделей в двух проекциях, учитывающих различия в интеллектуальных механизмах правого и левого полушарий.

Ц1. Обеспечить проектировщика средствами динамической визуализации графических моделей.

Ц2. Обеспечить проектировщика средствами, автоматизирующих процессы пошагового формирование постановки проектной задачи.

Ц3. Обеспечить проектировщика средствами позволяющими строить модели прецедентов по ходу решения проектных задач

Ц4. Автоматизировать мысленные эксперименты, проводимые в процессе решения задач

Достижение каждой из поставленных целей приведет к следующим положительным эффектам:

- Достижение Ц 0. позволит вовлекать в процесс проектирования ранее использующиеся модели, что позволит повысить производительность труда проектировщика.
- Достижение Ц 3. позволит вовлекать в процесс проектирования всех заинтересованных сторон (stakeholders). В основном акцент делается на команду проектировщиков, но использование моделей в качестве основных артефактов делает возможным вовлечение в процесс проектирования заинтересованных лиц, не обладающих навыками программирования (заказчики, менеджеры и т.д.).
- Реализация Ц 1. и Ц 2. направлена на предотвращение и нахождения ошибок. Преимущественно акцент делается на концептуальных ошибках, противоречиях, несогласованном поведении системы. Такие ошибки необходимо устранять преимущественно на этапах проектирования, так как потом стоимость их исправления значительно возрастает.
- Достижение Ц 0. и Ц 3. позволит минимизировать затраты на поддержку системы. Наличие моделей обладающих алгоритмической простотой и вовлеченных в процесс разработки (благодаря этому модель не может противоречить тому, как функционирует реальная система) позволит уменьшить стоимость поддержки системы в будущем. Так как на этапе проектирования создаются и отлаживаются модели, которые непосредственно

станут основой будущей системы (а не просто документацией), невозможны случаи, когда модель будет противоречить реальному функционированию программы. В существующей практике очень часто модели не имеют воплощения в коде и часто, с течением времени, вступают в противоречие с тем, как система реально функционирует.

- Достижение Ц 0. – Ц 4. Приведет к сокращению сроков разработки системы, уменьшение вероятности возникновения ошибок, и как следствие, увеличение степени успешности разработок.

Каждая из поставленных целей включает в себя ряд требований, которые должны быть исполнены для ее разрешения. Весь перечень требований представлен на рисунок 1.24.

Современный процесс разработки невозможно представить без создания визуальных моделей, описывающих бизнес требования, алгоритмы и структуры данных. Часто эти модели не имею прямой связи с кодом и, со временем, вступают с ним в противоречие. Наличие такой несогласованности удорожает процессы развития, тестирования и вовлечения в процесс разработки новых лиц.

Современную коллективную разработку невозможно представить без средств контроля версий исходного кода. Вовлечение в процесс проектирования графических моделей делает необходимым адаптацию/доработку существующих решений для работы с графическими моделями. В тоже время учет специфики диаграмм упрощает процессы разрешения конфликтов, которые являются очень частыми в коллективной проектной деятельности.

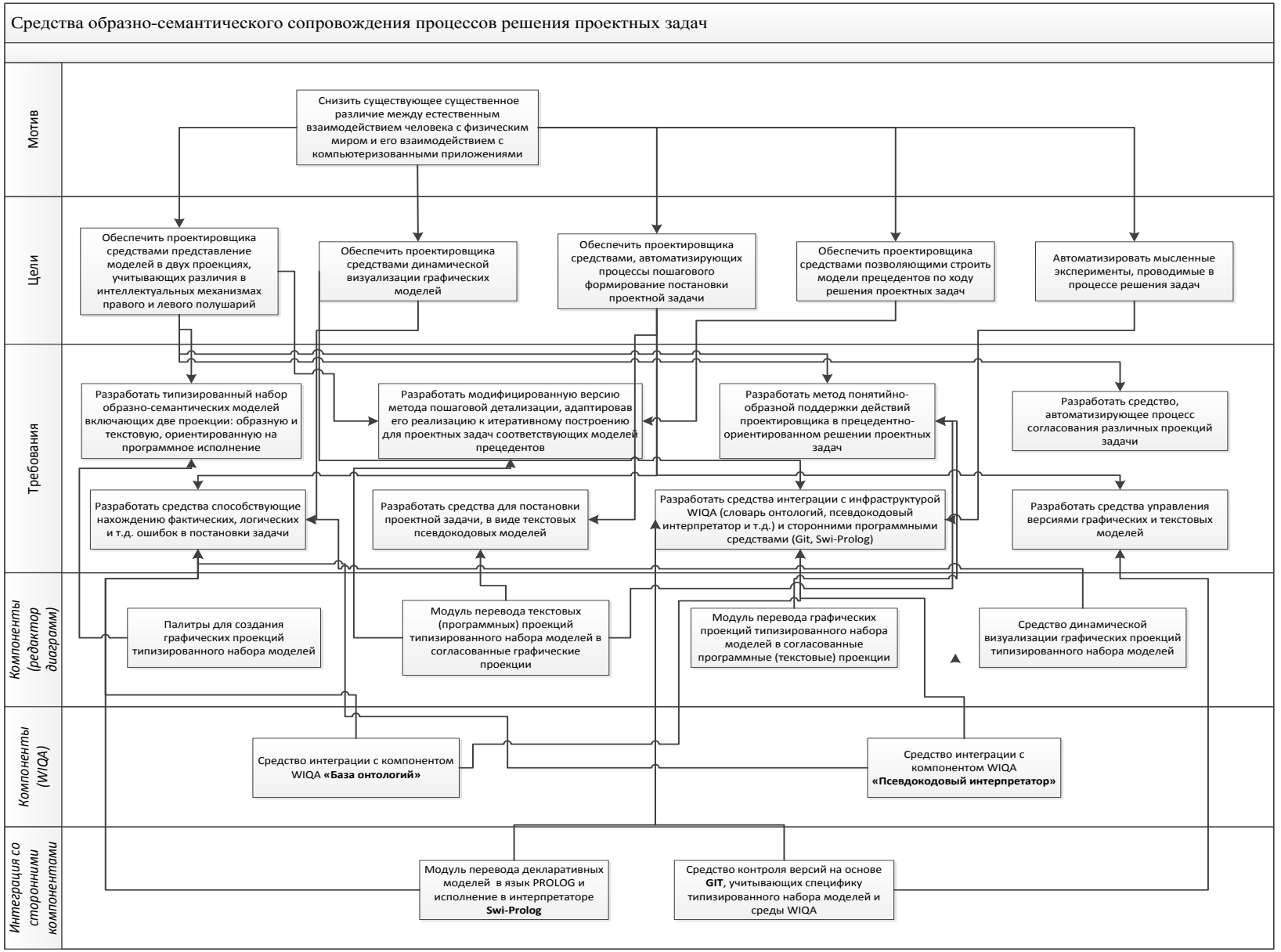

Рисунок 1. 24 - Мотивационно-целевая диаграмма

1.4. Выводы

1. Современные тенденции в разработке средств визуализации ориентированы на учет когнитивных факторов деятельности проектировщика и поддержку ряда согласованных представлений (intentional platform, “Графит-Флокс”) или проекций решаемой задачи.

2. Среди графических моделей (mind mapping, fishbone diagram, influence diagram и т.д.), рассмотренных в обзоре и базирующихся на «block and line» диаграммах содержаться те, которые имеют изобразительную, декларативную (констатацию фактов) и императивную (алгоритмы) семантику.

3. В тоже время большинство современных средств визуализации не позволяют работать с несколькими проекциями задачи и обеспечить формирование целостного понимания о её структуре и связей между составными частями.

4. Одним из принципиальнейших средств, сопровождающих процесс решения задачи, является схематизированная графика, раскрывающая процесс решения с полезных точек зрения.

5. В прецедентно-ориентированном решении проектной задачи следует использовать итеративное формирование модели прецедента, нормативная схема которого наполняется понятийно-образным содержание в процессе пошагового формирования постановки задачи в результате вопросно-ответного анализа её состояния.

6. В процессе решение необходимо иметь возможность (по ходу решения задачи) комплексировать её различные представления и управлять процессом решения, по ходу которого строится модель соответствующего прецедента, интегрирующая текстовую, графическую и алгоритмическую информацию.

7. Построение необходимых образно-семантических моделей должно осуществляться и использоваться в автоматизированных мысленных экспериментах, каждый из которых по определённым причинам затребован в

прецедентно-ориентированном пошаговом решении определённой проектной задачи.

8. Используемые образно-семантические модели должны быть типизированы и специфицированы так, чтобы они были согласованы с нормативной моделью прецедента, допуская полезные преобразований между типами и переходы к их программным представлениям.

9. Положительные эффекты от применения методов, моделей и средств визуализации должны быть нацелены на повышение эффективности поиска семантических ошибок, возможность контроля этапов решения задачи, динамическую визуализацию процессов создания моделей и возможность автоматического перехода к их программным версиям, ориентированную на повторное использование (наполнение модели прецедента).

Глава 2. Формализация процесса решения задач

Главной особенностью взаимодействия человека с физическим миром является явное или неявное использование им естественного опыта (E^N), формирование которого происходит в процессе интеллектуальной обработки поведенческих действий. В общем случае, любой новый элемент E^N начинает свой жизненный цикл с умственной обработки потоков информации, поступающих от органов чувств. В тех случаях, когда информационные потоки ввода отсутствуют или явно не используются, процесс формирования начинается с активации феномена «mental imagery». Суть «mental imagery» подробно изложена в публикациях [63] и [34]. Теоретическое описание этого феномена основано на двух теориях: изобразительной [51] и описательной [63]. Основой первого взгляда является его обусловленность первой сигнальной системой (органы чувств) в то время как вторая фокусируется на второй сигнальной системе (естественный язык). Как указывается во многих работах [49], обе эти точки зрения являются взаимодополняющими друг друга, так как, в действительности, человек получает информацию от обеих сигнальных систем.

Таким образом, существует множество определений этого феномена и его функций, но наши разработки фокусируются на следующих функциях:

1. позволяет моделировать «произвольную» реальность по своему желанию для предсказания того, что будет происходить в конкретной ситуации или после выполнения определенных действий;
2. позволяет делать конкретные предсказания, основанные на предыдущем опыте.

Как правило, активация этого явления происходит как в процессе взаимодействия с физическим миром, так с компьютеризированной средой.

В представленной работе делается акцент на задачах в проектировании SIS, которые являются новыми. Подобная ситуация возникает, когда проектировщик работает с одной из назначенных задач в компьютеризированной среде, а в это время на мониторе, появляется сигнал о новой задаче. При взаимодействии с

компьютером, основную часть информации человек получает через программную среду, в виде аудио и видео информации. Формы такого взаимодействия, созданные программистами (создателями данной программной среды) и основанные на использовании соответствующих решений из теории и практики HCI, обычно выражаются посредством интерфейсных форм, направленных на поддержку ментальных образов, которые позволят продолжить компьютерную обработку. Таким образом, процесс взаимодействия с компьютером (средой) состоит из двух компонентов естественного и искусственного типов, и конечные результаты взаимодействия будут существенно зависеть от способа, который используется для координации этих компонентов. При первоначальном осознании новой задачи, мышление пытается выразить ее в понятной форме, проводя действия с концептуальными компонентами (образами) с целью формирования определенного (начального) уровня понимания; такой процесс может быть интерпретирован как мысленный эксперимент.

В главе рассмотрены теоретические основы прецедентно-ориентированного процесса решения проектных задач посредством согласования графических и текстовых проекций типизированного набора образно-семантических моделей. Представлен общий подход к сопровождению процессов решения проектных задач, приводится формализация графичесуих и програмных проекций этих моделей (декларативная, концептуально-алгоритмическая и изобразительная), обобщенно представляются метод итеративного согласования постановки задачи и метод пошаговой детализации в прецедентно-ориентированном решении.

2.1. Подход к образно-семантическому сопровождению процессов решения проектных задач

Разработка данного подхода не возможна без рассмотрения когнетивных процессов происходящих в процессе решения задач, в основе которых лежат интеллектуальные механизмы правого полушария, ориентированных на обработку образной информации и/или имитирующие такую активность, и механизмы левого полушария, включающие в интеллектуальную обработку

использование естественного языка. Предположение о том, что правое и левое полушария мозга, обладают разными функциями было высказано более 140 лет назад (статья Jackson, J. H. “On the nature of the duality of the brain”, 1874).

Многие исследователи сходятся во мнении, что полушария мозга работают параллельно, и могут быть схематично представлены в виде двухпроцессорной системы, каждая из которой имеет свой набор команд, т.е. ориентирована на обработку определенного вида информации. Левое полушарие может выполнять такие операции как прогнозирование, наименование, классификация, вывод, абстрагирование, правое отвечает за озарение, движение, синтез [48, 40, 78, 86, 87].

Схематичное изображение процессов концептуального решения проектных задач на семантическую память инструментальной среды WIQA, приведено на рисунке 2.1.

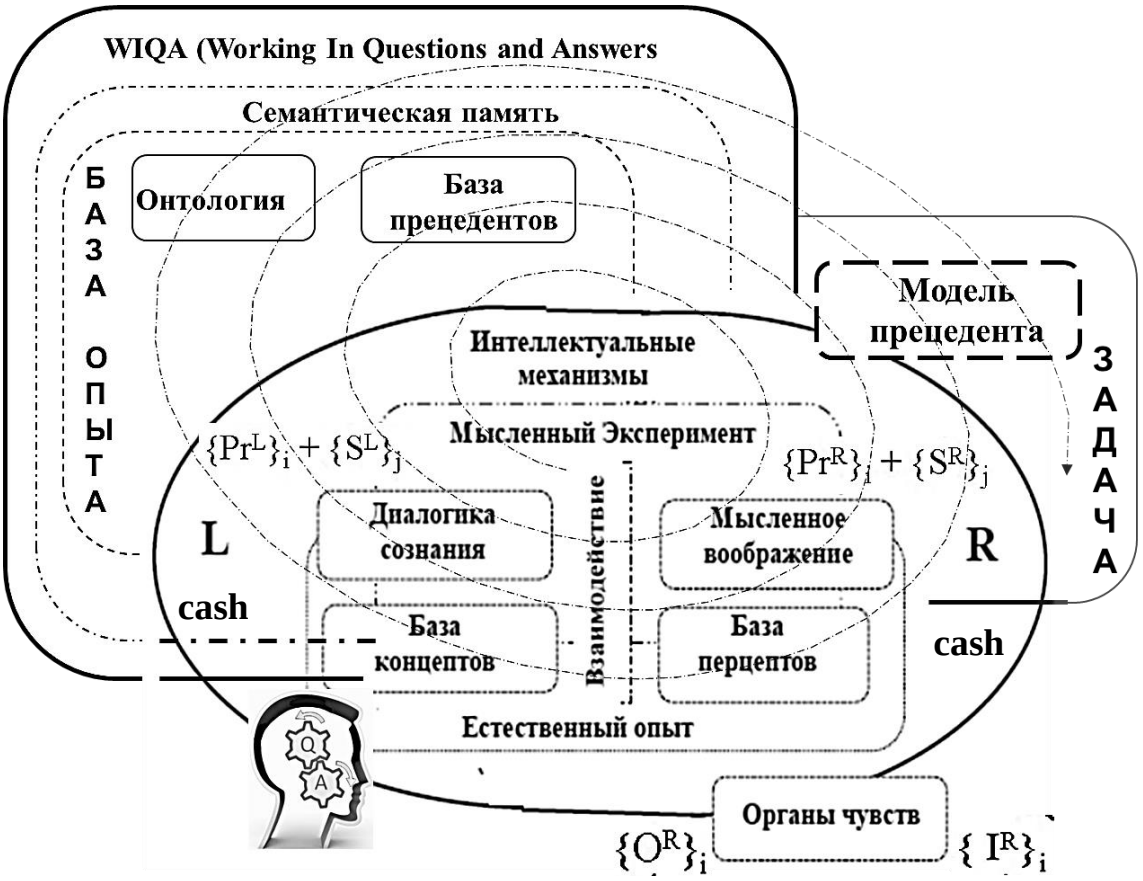

Рисунок 2. 1 - Взаимодействие проектировщика со средой в процессе концептуального решения проектных задач

На схеме показано, что в процессе решения задачи проектировщик проводит автоматизированные мыленные эксперименты, используя базу (моделей) опыта, включающую «Онтологию» проектирования и «Базу прецедентов». В автоматизацию любого из мысленных экспериментов конструктивно вовлечены диалогика сознания и мысленное воображение, взаимодействующие с естественным опытом, включающим прообразы базы концептов (онтология, левое полушарие L) и базы перцептов (правое полушарие R). Одним из важных результатов такой деятельности являются естественно-языковые модели, на основе которых могут быть построены модели прецедента, ориентированная на возможность повторного использования.

За «спиралью» на схеме стоит реализация метода понятийно-образной поддержки процесса пошаговой детализации в прецедентно-ориентированном решении задач, где центральное место занимает нормативная структура прецедента P, включающая текст постановки задачи P^T, вопросно-ответную модель P^{QA} его анализа, логическую модель P^L, графическое представление P^G, исходный исполняемый псевдокод P^I и программное представление P^E прецедента.

Библиотеки образов и понятий состоят из множества единиц опыта, которые характеризуются набором признаков $\{Pr_i\}$ и свойств $\{S_j\}$. Признак отличается от свойства тем, что свойство относится к конкретному объекту, тогда как по признаку можно догадываться о том, что является субъектом. При поиске идет учет только признаков, причем, если по Pr_i в библиотеке $\{Pr\}^R$ или $\{Pr\}^L$ не найдено элементов (либо поиск занял слишком большое время), то в соответствующую библиотеку добавится новый элемент с признаком Pr_i и свойством S_i. Следует отметить, что на скорость поиска влияет “важность” признака – более важные признаки ищутся быстрее и их свойства трудно изменить.С помощью алгоритмов ввода $\{I_j\}^R$ осуществляется преобразование полученной информации (внешних воздействий) в совокупность признаков, то есть происходит преобразование (распознавание) входных сигналов. Обратное преобразование полученной информации в выходные данные осуществляется

также через правое полушарие, но уже с помощью алгоритмов $\{O_i\}^R$, которые преобразуют ответ на языке признаков в набор внешних действий.

Жизненный цикл новой задачи начинается с обнаружения проектировщиком определенного сигнала, активизирующий правое полушарие через органы чувств. Семантика сигнала определяется проектировщиком как появление новой задачи и происходит активация феномена «mental imagery», который способствует формированию первоначального «видения» задачи путем фильтрации и связывания соответствующих концептуальных компонентов, извлеченных из опыта E^N. В этом итерационном процессе, формирование понимания обусловлено проверкой задачи на полноту W, которая может быть представлена как в вербальных и графических формах, так и в виде их комбинаций. В процессе перехода от внутренних форм W к внешним версиям (в виде моделей M), становится возможным воспринимать их через органы чувств (привлекать к решению других проектировщиков). Любая внешняя версия модель M, которая представляет W, может быть использована для достижения следующих целей:

1. Попытки активировать W в любое время, даже в тех случаях, когда проектировщик забыл W.
2. Использование M для фильтрации внутренних и внешних информационных потоков, влияющих на W.
3. Использование M для исследования W и любых ее компонентов или их группы.
4. Понимание W посредством понимания M.
5. Манипуляции с M (в материализованном виде) вместо манипуляций с W (в сознании).
6. Использование M для модификации W за счет модификации M или создания его новой версии.
7. Управление психическими процессами путем организации потока взаимодействий с M.

На практике, проектировщики активно используют перечисленные функции различных моделей M_i при решении проектных задач, особенно на этапах концептуального проектирования. На этом этапе проектировщик для решения задачи Zi использует соответствующий набор $\{M^V_J\}$ текстовых (вербальных) описаний (моделей) и графических моделей $\{M^G_K\}$, формируя таким образом, свое концептуальное представление. Особенности такого типа решений образно представленных на рисунке ниже, где также показан итерационный процесс формирования постановки St (Zi, T) задачи Zi в процессе её концептуального решения. Как сказано выше, жизненный цикл новой задачи Zi начинается с реакции на её семантику. Такую семантику лучше зарегистрировать с помощью набора знаков $\{W_Q\}$ в виде набора ключевых слов. Следующим шагом является осознание этого множества, в результате чего дизайнер формулирует первоначальную постановку задачи St (Zi, t_0). С этого момента времени t0 жизненного цикла, состояние задачи становится наблюдаемой в виде St (Zi, t). Далее, шаг за шагом, с использованием концептуальных действий, описанных выше, и других необходимых действий, проектировщик решает задачу, и регистрирует состояние процесса в соответствующих St (Zi, t). Любое состояние St (Zi, tn) этого описания обусловлено набором текстовых и графических моделей, которые были созданы и использованы проектировщиком до момента времени tn.

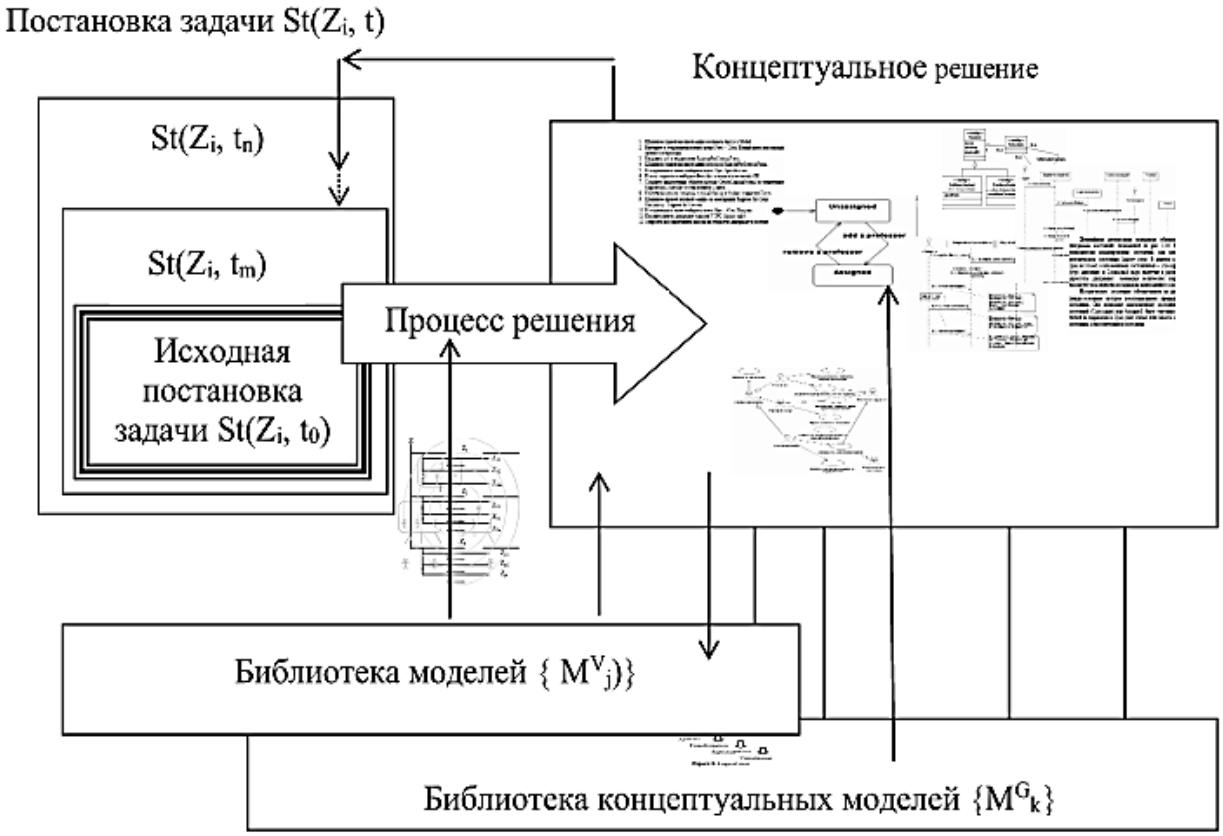

Рисунок 2. 2 - Концептуальное решение задачи

На рисунке 2.2 показаны два источника концептуальных моделей, которые применяются в процессе решения. Они условно называются библиотеками, чтобы подчеркнуть, что такие модели могут быть использованы для хранения и повторного использования моделей. В данном исследовании, данный подход реализован на базе инструментальной среды WIQA, где проектировщик создает и изменяет постановку St (Zi, t) путем использования метода пошаговой детализации на основе QA-анализа и моделирования (рисунке 2.3).

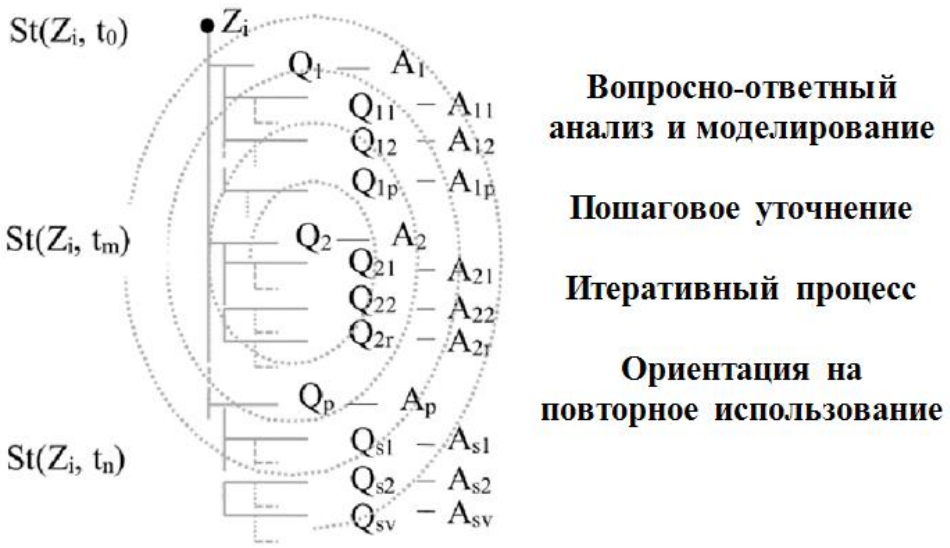

Рисунок 2. 3 - Уточнение постановки задачи в процессе решения

В свою очередь, постановки St (Zi, t) отражают процесс концептуального решения, ориентированного на повторное использование в качестве прецедента.

В процессе решения задачи происходит постепенное приращение модели прецедента (которая лежит в основе), т.е. можно говорить о применении проектировщиком подхода MDD для этой модели. На рисунке показаны особенности этого подхода.

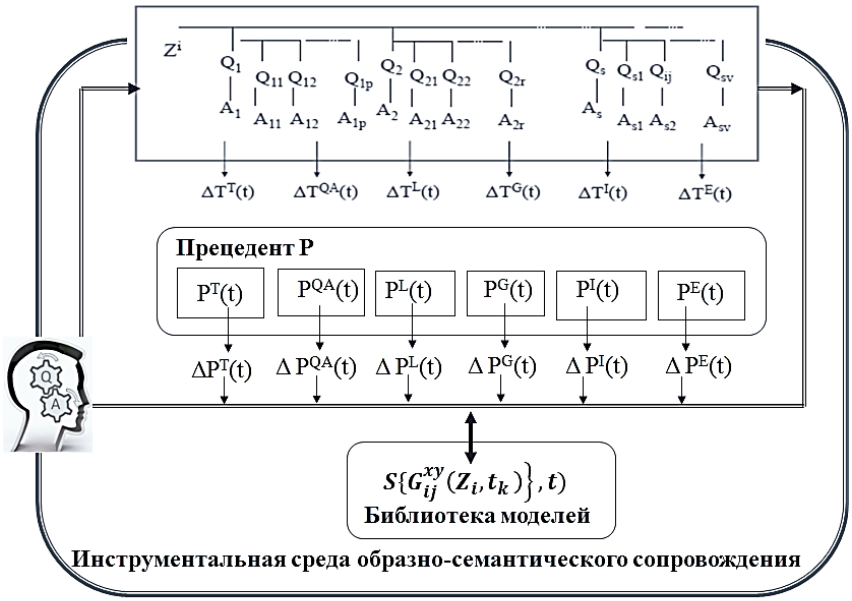

Рисунок 2. 4 - Итеративное создание и использование модели прецедента

На рисунке 2.4, постановка задачи St (Zi, t) в форме QA-дерева выполняет роль источника текстовых приращений $\{\Delta T_{nu}^{mx}(Z_i, t_j)\}$, каждое из которых (с индексом u) формулируется на m-уровне и n-стадии пошагового уточнения для создания следующего приращения $\Delta P_{nu}^{mX}(Z_j, t_k)$ X-компоненты прецедента модели P(t). В рамках данного исследования предлагается использовать структуру прецедента, которая неоднократно рассматривалась в работах Соснина П.И. [99, 100, 101] и представляет собой совокупность следующих X-компонентов: текстовое описание $P^T(t)$ прецедента; вопросно-ответное описание $P^{QA}(t)$ в виде зафиксированных QA-рассуждений; логические формулы $P^L(t)$; графическая (диаграмма) представление $P^G(t)$; псевдокод модель $P^I(t)$ в виде программы псевдокода; и модель, которая представляет свой исполняемый код $P^E(t)$. Этот выбор компонентов обусловлен использованием подобия между интеллектуальной обработкой при решении задачи и интеллектуальной обработкой условных рефлексов.

Предположим, что в момент времени T_j проектировщик подготовит следующую порцию $\Delta T_{nu}^{mx}(Z_i, t_j)$. Как уже было сказано выше, этот текстовый элемент используется в качестве модели M^V-типа, которые должны быть концептуально проверены. Первая проверка должна выполняться путем использования умственного экспериментирования. Когда текст $\Delta T_{nu}^{mx}(Z_i, t_j)$ визуально наблюдаем, проектировщик явно или неявно активирует феномен «mental imagery», который приведет к созданию соответствующей графической модели в качестве следующего приращения $\Delta P_{nu}^{mX}(Z_j, t_k)$ компонента P^G. Формально такой переход может быть представлен выражением:

$$\Delta T_{nu}^{mG}(Z_i, t_j) \overset{R_1^G}{\rightarrow} \Delta P_{nu}^{mG}(Z_j, t_k)$$

где R_1^G является отображением обеспечивающим переход «текст-графика», и p является уникальным индексом этого перехода. Если целью перехода является достижение понимания, тогда

$$\Delta P_{nu}^{mG}(Z_j, t_k) = M_v^{Gy}(Z_j, t_k)$$

где $M_v^{Gy}(Z_j, t_k)$ является графическая модель с уникальным индексом v и у это тип подчиненной модели, с которым изображение $M_v^{Gy}(Z_j, t_k)$ должно быть связано. В качестве элемента повторного использования, этот образ должен быть встроен в библиотеку $S(\{M_v^{Gy}(Z_j, t_k)\}, t)$.

Далее приращение графики может привести к следующему отображению:

$$\Delta P_{nu}^{mG}(Z_i, t_k) \overset{R_2^x}{\rightarrow} \Delta St_p^G(Z_j, t_r)$$

в результате которого, будет изменено текущее состояние постановки задачи

$$St(Z_i, t_r) = St(Z_i, t_r) \cup \Delta St_p^G(Z_j, t_r)$$

Отметим, переходы между другими видами приращений $\Delta T_{nu}^{mx}(Z_i, t_j)$ и $\Delta P_{nu}^{mG}(Z_j, t_k)$ также приводят к сходным результатам и эффектам. Таким образом, в процессе пошагового уточнения, с использованием обратной связи, разработчик создает модель прецедента и использует её для построения решения соответствующей задачи.

Таким образом, специфику подхода определяют:

1. Итеративная реализация метода управляется текущими результатами $\{\Delta T^X(t),\ X=(T,\ QA,\ L,\ G,\ I,\ E)\}$ пошаговой детализации постановки задачи $TP(Z,\ t)$, регистрация которых проводится в дереве $P^{QA}(t)$ вопросно-ответного анализа.
2. Каждое приращение $\Delta T^X(t)$ способно привести к приращению $\Delta P^X(t)$ соответствующей составляющей прецедента P, среди которых для образно-семантической поддержки принципиальное место занимают образно-семантические модели, которые аккумулируются в библиотеку моделей.
3. Каждая графическая модель, в зависимости от её типа (изобразительный, декларативный, концептуально-алгоритмический) создаётся проектировщиком в соответствующей операционной среде специализированного графического редактора.

4. Каждую графическую модель, если в этом имеется необходимость, проектировщик может подключить к соответствующему блоку постановки задачи.
5. Для повышения эффективности визуального моделирования, проектировщику предоставлена возможность автоматизированного преобразования типов (рисунок 3) для создаваемых или созданных графических моделей.

2.2. Типизированный набор образно-семантических моделей и система их согласованных преобразований

Основными целями получения концептуального решения задачи являются:

1. Представление постановки St (Zi, t) задачи Zi в понятной форме, для которой проверяется возможность алгоритмической реализации (существование решения).

2. Разработка и тестирование концептуально алгоритмическое решение.

Для достижения первой цели, проектировщик включает необходимые текстовые и графические модели от $\{M^V_J\}$ и $\{M^G_K\}$ в St (Zi, t), а также встроенные компоненты. При повторном использовании это обеспечит активацию феномена «mental imagery», без которого невозможно добиться понимания. Инструментарий WIQA поддерживает создание и преобразование визуальных моделей для типов, представленных на рисунке 2.5.

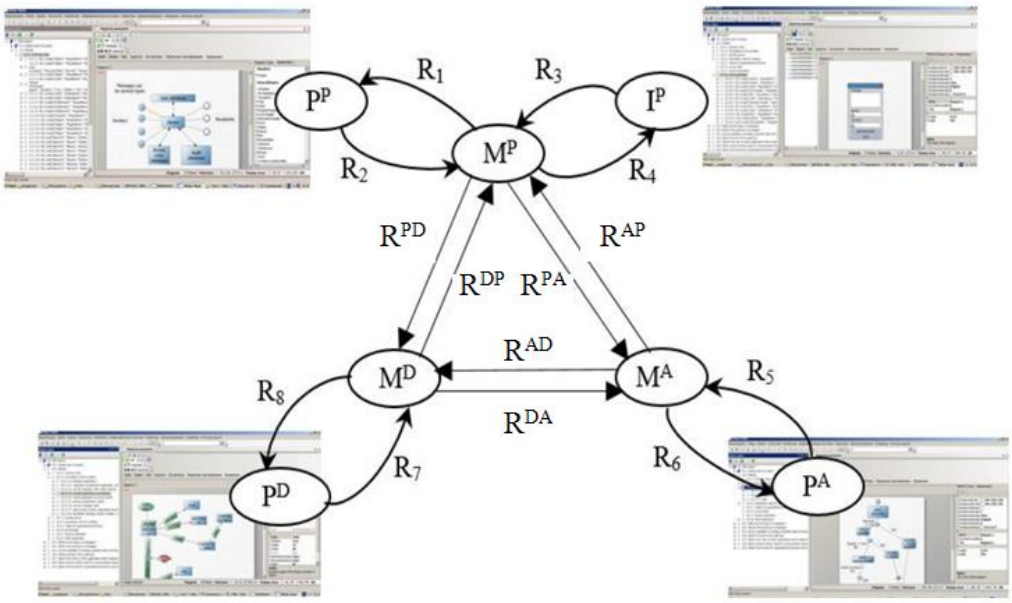

Рисунок 2. 5 - Типы используемых моделей и преобразования между ними

Множество типов включает:

- Изобразительный тип M^P, модели которого, представляют произвольные типы требуемых графических моделей. Для обеих проекций модели, используется графический редактор, позволяющий автоматически изменять программные версии, при изменении соответствующих графических проекций. Такая возможность реализуется для всех 3 типов моделей.
- Декларативный тип M^D предназначен для визуализации семантики текстовых моделей $\{M^V\}$, которая важна для достижения необходимого уровня понимания. Основной причиной для использования этого типа является проверка St (Zi, t) и его словесных компонентов с помощью механизмов декларативного программирования. Таким образом, версия P^D имеет PROLOG-подобное описание и может быть исполнена в интерпретаторе ПРОЛОГ.
- Концептуально-алгоритмический тип M^A обеспечивает визуальное моделирование алгоритмических компонентов решаемой задачи. Для

этого, разработчик может использовать псевдокодовый язык LWIQA, который построен для семантической памяти инструментария WIQA. В текущей версии WIQA, тип M^A поддерживает работу с диаграммами вариантов использования, диаграммами активностей и классов. Для перехода к программной версии P^A используется подход, известный как Model Driven Development [15, 8], ядром которого являются модели и преобразования (т.е. множественные операции над моделями). Одним из главных преимуществ данного подхода является автоматизация процесса разработки программного обеспечения, который начинается с концептуального этапа и, путем итерационных уточнений, заканчивается готовым исполняемым приложением.

На схеме на рисунке также показаны отношения между (возможности автоматического или автоматизированного перевода) тремя видами. Множество отношений состоит из пар, представляющих трансформации между соответствующими моделями различных видов (переходы $[R^{PD}, R^{DP}]$, $[R^{PA}, R^{AP}]$, $[R^{DA}, R^{AD}]$), а также между моделями и их программных версий ($[R_1, R_2]$, $[R_3, R_4]$, $[R_5, R_6]$, $[R_7, R_8]$):

- $[R^{PD}, R^{DP}]$ – автоматический перевод графической проекции декларативного представление в программу рисования на псевдокоде и исполнение её в интерпретаторе (R^{PD}).
- $[R^{PA}, R^{AP}]$ – автоматический перевод графической проекции концептуально-алгоритмических моделей в программу рисования на псевдокоде и исполнение её в интерпретаторе (R^{PA}).
- $[R^{DA}, R^{AD}]$ – автоматизированное преобразование (согласование) концептуально алгоритмических (диаграммы классов, активностей и вариантов использования) и семантической блок-схемы (графическая проекция декларативного представления).

- [R_1, R_2] - автоматическое преобразование произвольной диаграммы (созданной в рамках поддерживаемых палитр) в программу рисования с последующим исполнением в интерпретаторе псевдокода.
- [R_3, R_4] – автоматизированное преобразование интерфейсных прототипов (подмножество изобразительного представления), с возможностью вызова псевдокодовых функций (которые могут быть созданы в рамках концептуально алгоритмического представления) при возникновении различных событий (клик, ввод текста и т.д.).
- [R_5, R_6] – автоматический перевод графической проекции концептуально-алгоритмического представления в псевдокодовую программу (соответствующую типу созданной графической модели), с возможностью её отладки в интерпретаторе.
- [R_7, R_8] – автоматический перевод семантической граф схемы в прологоподобное текстовое описание (программу) с возможностью исполнения в интерпретаторе ПРОЛОГ.

Любая версия графической модели соответствует исполняемой программе, которая может быть изменена проектировщиком, например, чтобы исправить ошибки или уменьшить неопределенность в визуальной модели. Такие трансформации могут идти в нескольких направлениях:

- $St_k \xrightarrow{Tst_{k+1}} St_{k+1}$ - от более абстрактного уровня к менее абстрактному (синтез). Наиболее популярный вид трансформаций; за счет уточнения моделей и процесса пошаговой детализации в итоге происходит получение исполняемого псевдокодового прототипа решения.
- $St_{k+1} \xrightarrow{Tst_k} St_k$ - от менее абстрактного уровня к более абстрактному (реверс инжиниринг). За счет интеграции процесса с системой контроля версии и фиксации состояний St_0, St_1 и т.д., трансформация к более абстрактному уровню представляет собой переключение на предыдущее состояние (commit), то есть можно считать трансформацию St_k также обратимой.

На рисунке 2.6 ниже представлены рассмотренные трансформации в разрезе общепринятой классификации, использующейся в MDD:

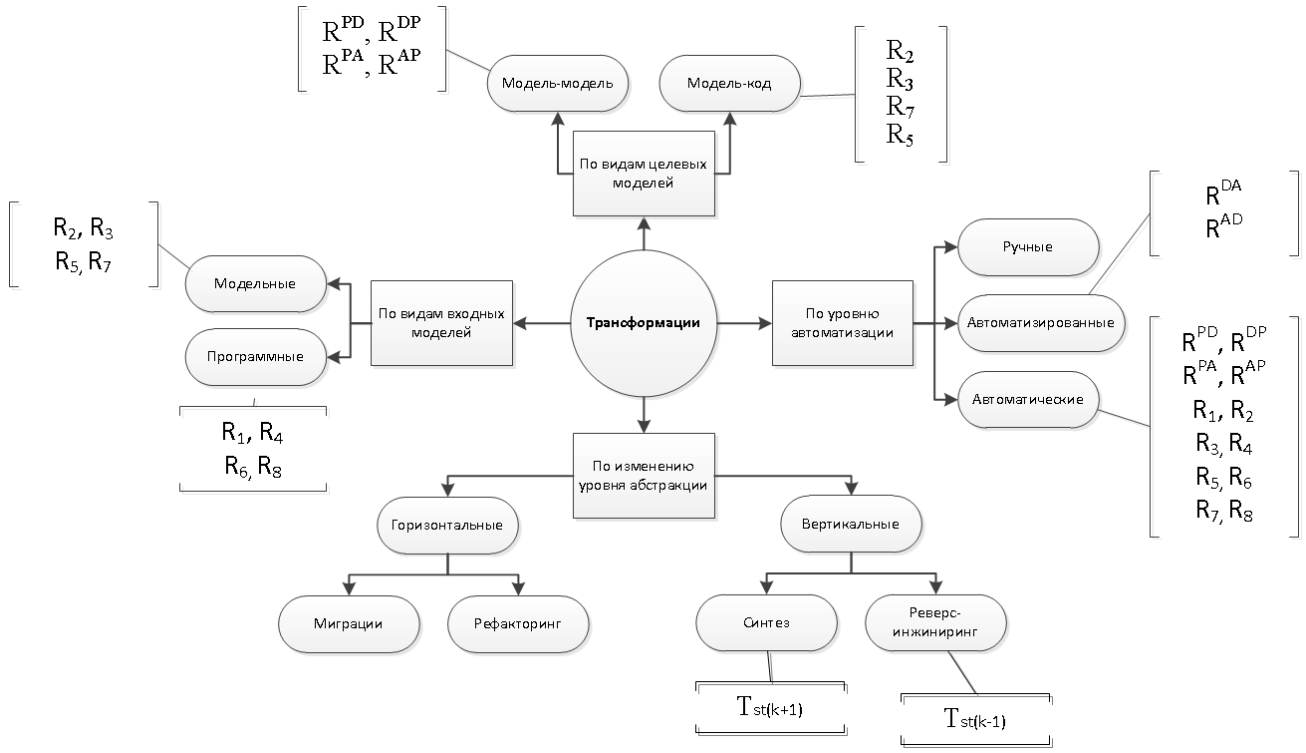

Рисунок 2. 6 - Классификация используемых трансформаций

Альтернативной версия формализации процесса трансформаций моделей является формализация на языке РБНФ грамматик, которая дополнительно раскрывает вопрос порождения единиц.

Постановка задачи = Текстовое описание , [Графическое описание] ;
Текстовое описание = Описание цели , [Описание алгоритма] , [Описание среды] ;
Графическое описание = Граф. Диаграмма | Графическое описание ;
Граф. диаграмма = Графическая проекция И | Графическая проекция КА | Графическая проекция Д ;
Графическая проекция И = И диаграмма , Программная проекция И ;
Графическая проекция КА = КА диаграмма , Программная проекция КА ;
Графическая проекция Д = Д диаграмма , Программная проекция Д ;
КА диаграмма = Диаграмма активностей | Usecase диаграмма | Диаграмма классов ;
Программная проекция КА = Лексема псевдокода | Вопросно-ответная проекция КА ;
Программная проекция Д = Предикатная форма | Вопросно-ответная проекция Д ;
Программная проекция И = Вызов псевдокодовой функции рисования | Вопросно-ответная проекция И ;
Предикатная форма = Предикат , Дополнение ;

Структура памяти для хранения программных представлений

Для отображения декларативного, концептуально алгоритмического и изобразительного представлений в вопросно-ответную память WIQA необходимо

сначала формально описать такое представление. Вопросно-ответная память представляет собой иерархично организованное хранилище вопросно-ответных единиц:

$$\text{Вопросно} - \text{ответная память} = \{\text{Вопросно} - \text{ответная единица}\};$$

Единица представляет собой пару, которая состоит из вопроса и ответа, причем последний может отсутствовать (т.к. вопросы появляются раньше ответов).

$$\text{Вопросно} - \text{ответная единица} = (\text{Вопрос}, \text{Ответ});$$

Вопрос и ответ может состоять из-под вопросов и ответов на эти под вопросы. Такое разбиение дает большую гибкость и позволяет логично организовывать вопросно-ответные единицы (которые по своей сути представляются ветвями вопросно-ответного дерева).

$$\text{Вопрос} ::= Q \mid (Q, \{SUBQ\});$$

$$\text{Ответ} ::= A \mid (A, \{SUBA\});$$

С другой стороны, каждая вопросная единица имеет совокупность атрибутов, к которым относится:

- Дата создания, Dt
- Текстовое описание, $Desc$
- Идентификатор, Id
- Статус, St
- Тип единицы (проект, задача, вопрос, ответ и т.д.), $Type$
- Идентификатор родительской вершины (для корневой вершины он будет не определен), $Parent$
- Массив произвольных атрибутов в виде пар ключ, значение, Ext

Таким образом, вопросно-ответная единица может быть представлена в виде:

$$QA \coloneqq (Id, Dt, Desc, St, Type, Parent)$$

Средство образно-семантической поддержки процесса решения задач может работать с любой структурой, сохраненной в вопросно-ответном протоколе, так

как при трансляции указывается вершина, которая станет корневой для вопросно-ответного представления одной из моделей. Но всё-таки рекомендуется организовать работу с вопросно-ответным деревом, следующим образом. Множество задач должно быть сгруппировано по проектам, а каждая из задач разбита на 3 составляющих (на практике, иногда требуется представлять задачу в виде большего количества согласованных моделей)

$$P \coloneqq \{\mathrm{Z}\}$$

$$Z_i \coloneqq (D_i, A_i, V_i \mid i = 1..n)$$

где D_i, A_i, V_i – декларативная, концептуально-алгоритмическая и изобразительная проекции задачи.

2.3. Обобщенное представление методов понятийно-образной поддержки

В первом параграфе были представлены детали итеративного процесса решения проектной задачи с использованием пошаговой детализации только с позиции формирования и использования графики. В тоже время, в диссертационном исследовании, пошаговая детализация, дополненная итерациями, используется для управления по всем составляющим модели прецедента, а через эти составляющие и для управления процессом решения задачи (по образцу model driven development, то есть «разработки управляемой моделью» прецедента). Другими словами, формируемая модель прецедента используется для дополнительного вклада в управление процессом решения задачи.

При разработке сложных автоматизированных систем часто возникают ситуации, когда в процессе решения одной задачи происходит событие (которое влечет за собой создание задачи), требующее скорейшего решения. В таком случае проектировщик должен оценить негативные факторы и переключится с задачи, которую он решал Z_i на новую задачу Z_j. В общем виде такая ситуация показана на рисунке 2.7.

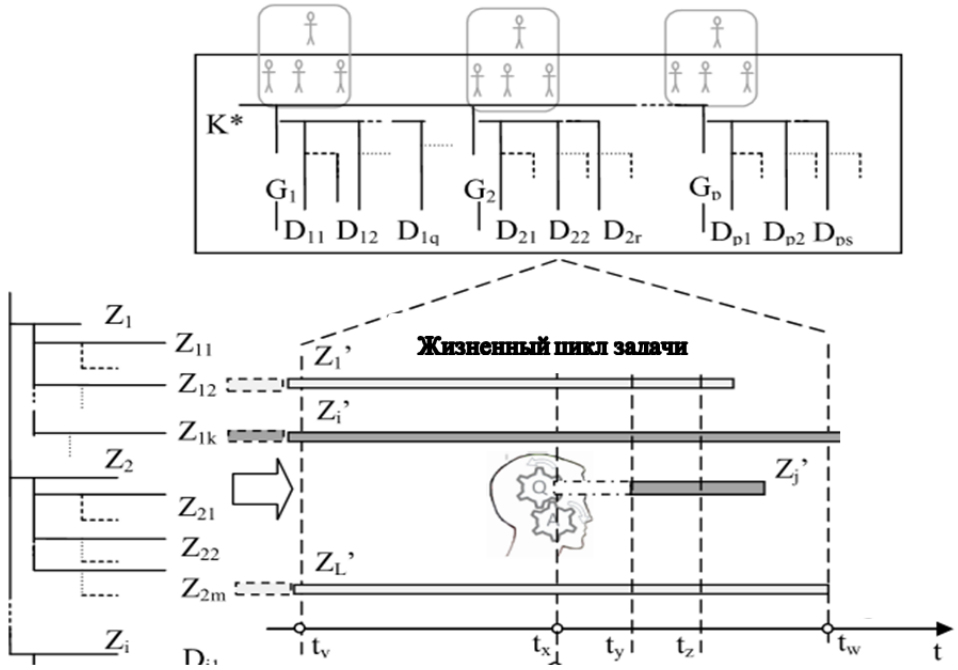

Рисунок 2. 7 - Представление деятельности проектировщика по решению задач

На рисунке видно, что в момент времени t_x проектировщик обнаруживает появление новой задачи Z_j и запускает новый жизненный цикл для нее. В этот момент времени когнитивные процессы приводят к формированию феномена, известного как “mental imagery”, активизирующий формирование представление задачи в виде совокупности образов. В момент времени с t_x по момент времени t_y проектировщик осуществляет предварительный анализ задачи с целью принятия решения о прекращении работы над задачей Z_j (оценивается важность новой задачи). На рисунке показан случай, когда произошло переключение на задачу и начинается отображение задачи на вопросно-ответную память.

Именно такая идея положена в основу «Метода понятийно-образной поддержки пошаговой детализации», обобщённая схема которого раскрывается на рисунке 2.8.

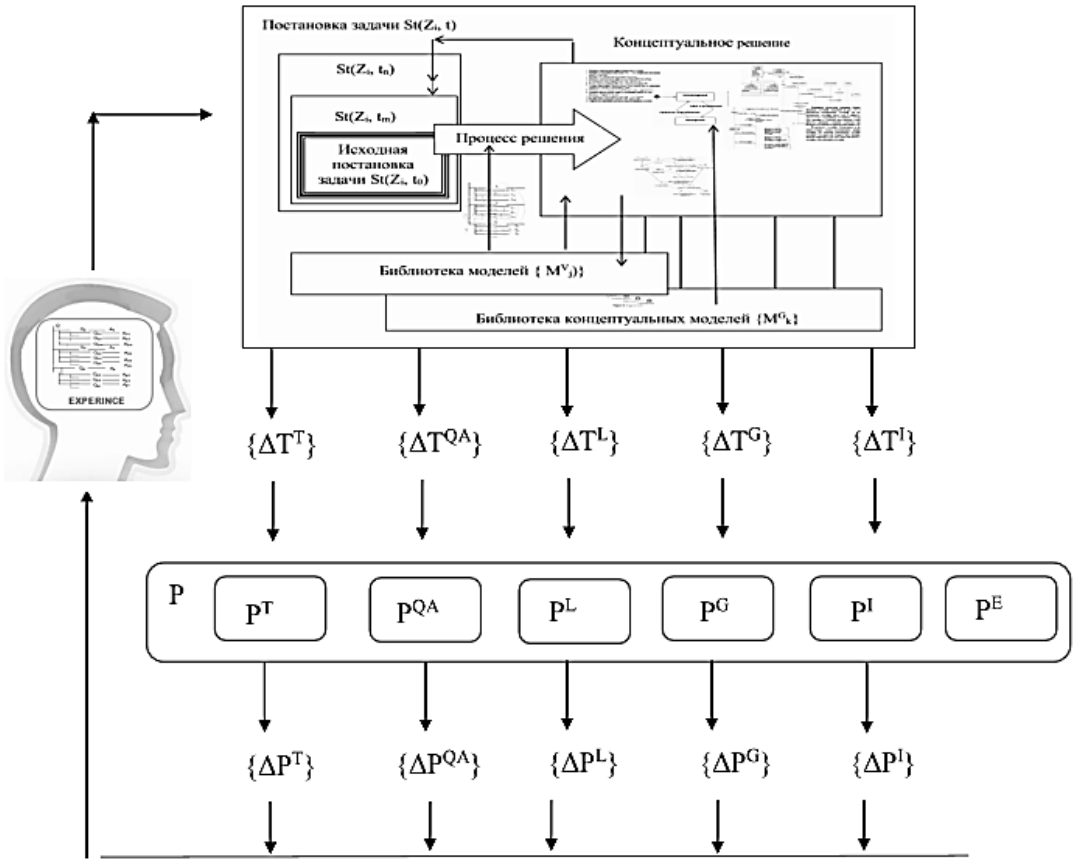

Рисунок 2. 8 - Обобщенная схема метода понятийно-образной поддержки пошаговой детализации

Реализация метода начинается с создания текста T исходной постановки St(Zi, t0) задачи Zi, после чего осуществляется первый шаг детализации, в основу которой (в методе) положен вопросно-ответный анализ текста. Примером такого анализа может служить его применение к задачи диссертационного исследования, приведённое в последнем параграфе первой главы диссертации. Суть метода основана на формирования и пошаговому уточнению типизированному набору моделей (декларативной, концептуально-алгоритмической и изобразительной), отличающихся одновременным представлением в виде графических диаграмм и текстовых (псевдо кодовых) моделей, тем самым ориентированных на когнитивные факторы процесса решения. Рассмотрим метод пошаговой детализации более детально с этапа зарождения задачи.

В результате анализа задачи на первом шаге формируется группа ответов, каждый из которых может выполнить роль приращения ΔT^X, где X как и в предыдущем пункте принадлежит множеству (T, QA, L, G, I, E).

Каждое из таких ΔT^X используется для формирования соответствующего приращения ΔP^X соответствующей модели P^X , а также, через текущее состояние $P^X (Z_i, t)$, для осуществления следующего шага детализации.

Названные действия метода повторяются в циклах итеративного формирования интегральной модели прецедента

$$P = P^S \cup P^T \cup P^{QA} \cup P^L \cup P^G \cup P^I \cup P^E,$$

в которой для каждой составляющей шаг за шагом строится её текущее состояние.

На рисунке выше не приведены связи с составляющей P^E, поскольку её формирование выходит за рамки построения концептуального решения задачи. Но именно концептуальное решение, через его концептуально-алгоритмическую составляющую P^I будет определять разработку высокоуровневой программной версии задачи Z_i. Для остальных составляющих их текущие версии будут определяться интегральными сборками

$$P^T = \{\Delta P^T\}, \text{ где}$$

$\{\Delta P^T\}$ множество приращений текстовое описание прецедента P

$$P^{QA} = \{\Delta P^{QA}\}, \text{ где}$$

$\{\Delta P^{QA}\}$ множество вопросно-ответное описание в виде зафиксированных вопросно-ответных рассуждений.

$$P^L = \{\Delta P^L\}, \text{ где}$$

$\{\Delta P^L\}$ множество логических формул,

$$P^G = \{\Delta P^G\}, \text{ где}$$

$\{\Delta P^G\}$ множество графических представлений (диаграмм)

$$P^I = \{\Delta P^I\}, \text{ где}$$

$\{\Delta P^I\}$ множество приращений псевдо кодовых исполняемых моделей

$$P = \{\Delta P^T\} \cup \{\Delta P^{QA}\} \cup \{\Delta P^L\} \cup \{\Delta P^L\} \cup \{\Delta P^G\} \cup \{\Delta P^I\},$$

Таким образом, Формирование прецедента формируется за счет приращений его составляющих.

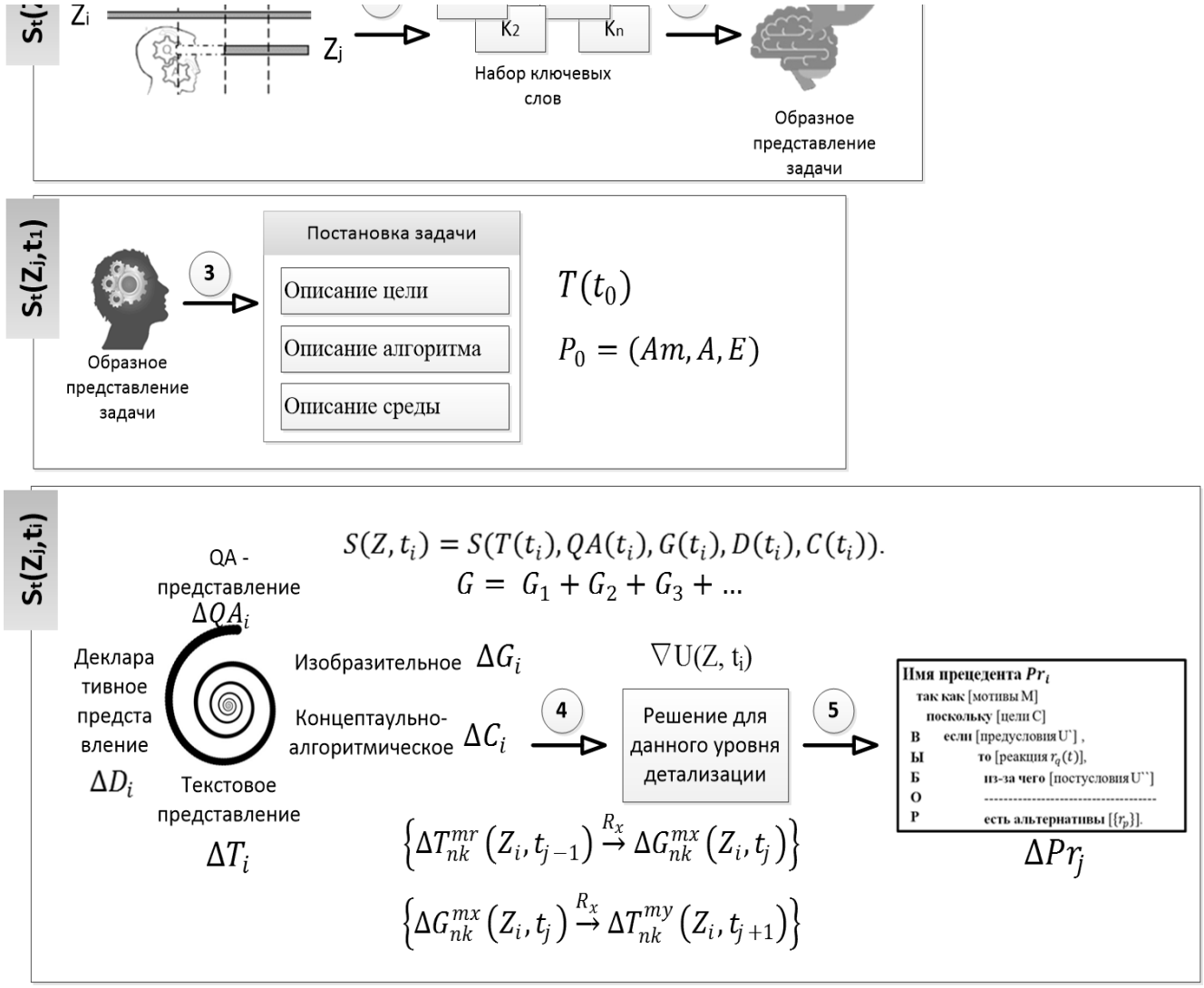

Рисунок 2. 9 - Процесс пошагового формирования решение проектной задачи

Как было сказано раньше на первом этапе после переключения на задачи она представляется набором ключевых слов (рисунок 2.9). Далее начинает формироваться текстовое представление постановки задачи $T(t_0)$, которое обычно (рекомендуется) состоит из 3 предложений. Первое предложение описывает цели, которые планируется достичь, а также может описывать возможности, которые это даст. Второе предложение делает акцент на идеях, которые описывают то как цели можно достичь, по сути здесь описывается примерный алгоритм решения задачи. Третье предложение обычно описывает

среду, в которой проектировщик будет решать задачу. В этот момент времени состояние задачи может быть описано состоянием каждой из составляющих его представлений $S(Z,t_1) = S(T(t_1),Q(t_1))$. В момент времени t_2 появляются рисунки, или изобразительное представление $G(t_2)$, которое содержит множество графических моделей. Такие модели не обладают определенной структурой, и поэтому могут быть представлены только в изобразительном представлении. Далее обычно происходит появление декларативного представления, на основе первоначальной постановки задачи, которое необходимо для "восприятия задачи как единое целое" согласно Дункеру, а также для исследования внутренних связей. Таким образом можно говорить о формировании декларативного $D(t_3)$ представления. Следующим этапом происходит формирование "функционального значения" или концептуально-алгоритмического представления $C(t_4)$. На этом этапе состояние задачи может быть представлено в виде $S(Z,t_4) = S(T(t_4),Q(t_4),G(t_4),D(t_4),C(t_4))$.

Состояние задачи S (Z, t_n) конструируется путем прохождения ей состояний, начиная от начального S (Z, t_0) (при регистрировании задачи) до S (Z, t_n) посредством преобразований, которые будут более подробно рассмотрены далее. При этом "объем" неопределенности уменьшается с каждой новой итерацией ∇U(Z, t_k). Такое уменьшение неопределенности необходимостью принимать определенные решения, каждое из которых будет конкретизировать (уточнять) состояние задачи. Обычно вначале времени t_0 неопределенность достигает своей наивысшей отметки ∇U(Z, t_0). Таким образом, движущей силой для пошагового формирования служит начальный объем неопределенности, который сокращается с каждым шагом ΔS(Z_j, t_0), ΔS(Z_j, t_2), …, ΔS(Z_j, t_{K-1}).

Преобразование декларативной и концептуально-алгоритмической моделей в изобразительные модели

Изобразительное представление является наиболее универсальным, позволяет сохранять произвольные модели в вопросно-ответном протоколе в виде программы рисования на языке LWIQA причем количество поддерживаемых

типов моделей обусловлено набором доступных палитр. Для декларативного представления доступна палитра включающая набор необходимых элементов: субъекта, объекта, атрибута и ассоциативной связи (поддерживается 14 типов). Поддержка концептуально-алгоритмического представления включает в себя три вида палитр для создания диаграммы классов, диаграммы активностей и диаграммы вариантов использования. При переводе происходит получение всех вершин $\{V_{\mathrm{d}}\}$ и связей $\{E_{\mathrm{d}}\}$ графической модели и преобразование в соответствующие конструкции языка LWIQA. Связи преобразуются в вызов функции

DD_Link(ShapeNameSrc, ShapeNameDst, LinkTemplate), где

ShapeNameSrc, ShapeNameDst идентификаторы вершин их которых выходит связь и в которую она входит и LinkTemplate – тип отношения с возможностью задания 14 типов для декларативной модели (и одного обобщенного в случае если элементами присутствует несколько типов отношений). Вершины преобразуются в функции вида:

DD_Create(TemplateName, ShapeName, [PropertyValue]), где

TemplateName – имя примитива из соответствующей палитры (в данном случае палитры PredicateShapes). Сочетание имени палитры и имени примитива должно быть уникально в рамках всех палитр. ShapeName – представляет строку, которая является текстовым наполнением вершины, а PropertyValue соответственно представлено массивом свойств, которые подробно рассматриваются в 4 главе и позволяют управлять цветом, положением углом поворота и т.д. Были представлены основные псевдокодовые методы рисования, полный набор также содержит функции необходимые для реализации динамической визуализации, и различных вариантов расстановки, которые применяются при переводе текстовых проекций в графические модели.

Преобразования декларативных моделей

Вопросно-ответное представление декларативной модели представляется набором предикатов, имеющих следующий вид:

Предикат (Субъект, [Свойство]; Объект, [Свойство]) ассоциативная составляющая.

Свойства являются необязательными параметрами и могут отсутствовать. В случае если несколько предикатов ссылаются на один субъект, будет создан только один субъект и все предикаты будут ссылаться на него. Для удобства навигации при выделении предиката в вопросно-ответном протоколе выделяется подграф, соответствующий выделенному предикату; при выделении вершины на графе выделяется соответствующий предикат. При редактировании графа автоматически перестраивается вопросно-ответное представление; при редактировании связанного вопросно-ответного представления, перестраивается графическая проекция. При семантической граф схемы происходит поиск всех вершин имеющих тип «предикат» и для каждой из низ ищется субъект и объект (вершины соответствующих типов), связанных с данным предикатом. Далее предикат помечается как обработанный и происходит переход к следующему предикату. При обратном переводе возникает проблема расстановки семантической граф схемы, так как программное представление не содержит информацию. В графическом редакторе реализовано несколько видов представлений (которые интегрированы в псевдокод и доступны через вызов функции DD_AutoLayout), которые более подробно рассматриваются в 4 главе, но наибольшее качество расстановки показывают комбинации этих алгоритмов, так как некоторые из них позволяют минимизировать количество пересечений, тогда как другие позволяют явно задать ориентацию графа (например, FlowLayouter позволяет осуществлять расстановку сверху вниз или справа налево что применимо для диаграмм активностей и декларативных граф схем).

Преобразования концептуально-алгоритмических моделей

Одним из основополагающих понятий подхода MDD являются “графовые трансформации” [8, 18, 28]. Трансформация – это автоматическая или автоматизированная генерация целевых моделей из исходных, согласно заданному набору трансформационных правил, описывающих каким образом

конструкции языка исходных моделей могут быть представлены в языке целевых моделей. Правила для графовых трансформаций представляют собой пары графов, имеющих обозначения Left Hand Side(LHS) и Right Hand Side (RHS) [28]. При применении правила на граф G происходит поиск всех подграфов, соответствующих LHS и замена найденных по соответствующему правилу на RHS.

Ориентированный граф, которым являются декларативная, концептуально алгоритмическая и образная модели, может быть представлен в следующем виде:

$$G \coloneqq (V, E), V - \text{множество вершин}, E := \left(V_i, V_j\right), \text{где } V_i, V_j \in G$$

Множество связей E, которые описывают граф, принято задавать в виде пар вершин. Как было отмечено выше, правило трансформации представляют собой пару граф поиска/граф замены и имеют вид:

$$Rule \coloneqq (G_1, G_2), \text{где}$$
$$G_1 \coloneqq (V_1, E_1, V_{н1}, V_{к1}), V_{н1}, V_{к1} \in V_1$$
$$G_2 \coloneqq (V_2, E_2), E_2 = \emptyset$$

В нашем случае, как видно из описания, граф для замены является вырожденным (множество связей E_2 графа G_2 является пустым множеством). Это необходимо для того, чтобы на каждом этапе получать более простой граф и в конечном итоге, путем ряда трансформаций получить граф, состоящий из одной вершины (содержащий вопросно-ответное представление).

Сам процесс трансформации может быть представлен в следующем виде:

$$\text{если } V_1 \subseteq \text{V } \wedge E_1 \subseteq E, \text{то}$$
$$G' \coloneqq (V', E'), \text{где } V' = V \cup V_2 \setminus V_1, E' = E \cup E'_{н2} \cup E'_{к2} \setminus E_1 \setminus E'_{н1} \setminus E'_{к1}, \text{где}$$

G' - граф, полученный после применения правила Rule к графу G,

$E'_{н2}$ - множество связей исходящих из любой вершины графа G и входящих в $V_2(V_{н1})$,

$E'_{к2}$ - множество связей исходящих из $V_2(V_{к1})$ и входящих в любую вершину графа G,

$E'_{н1}$ - множество связей исходящих из любой вершины графа G и входящих в $V_{н1}$,

$E'_{к1}$ - множество связей исходящих из $V_{к1}$ и входящих в любую вершину графа G.

Графовая трансформация может быть неоднозначной, если существуют связи, которые ведут к вершинам подграфа (LHS). Данное противоречие представлено на рисунке 2.10 на примере простой диаграммы активностей.

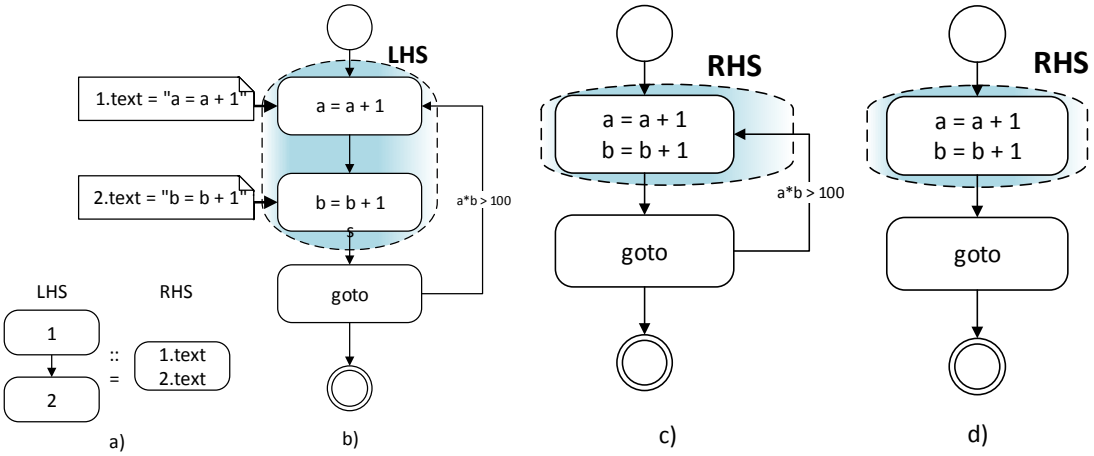

Рисунок 2. 10 - Пример применения графовой трансформации (a) на граф (b) с сохранением связей (c) и без сохранения (d)

Такое противоречие, решается несколькими способами. В нашем подходе мы исключаем возможность применения трансформации, если внутренне вершины LHS графа содержат связи с вершинами, не входящими с LHS. Для этого в формальное описание трансформации были введены $V_{н1}, V_{к1}$ – соответственно начальная и конечная вершины для правила. Под начальной вершиной понимается та, которая может иметь входящие связи из любых вершин графа не входящих в LHS, а под конечной – та, которая может иметь исходящие связи в любые вершины графа, не входящие в LHS.

Завершая параграф, отметим, что методическая структура «Метода понятийно-образной поддержки пошаговой детализации», через подчинённый ему «Метод итеративного согласования понятийного и образного содержания текстовых единиц» и другие методики, раскрывается в третьей главе.

2.4. Выводы

В заключение главы сформулированы следующие выводы:

1. Графическую поддержку прецедентно-ориентированного решения задач в достаточной мере обеспечивает типизированный набор образно семантических

моделей, включающий изобразительный, декларативный и концептуально-алгоритмический типы.

2. Изобразительный тип должен содержать возможности обеспечения визуальной поддержки мысленного воображения и «усиливать» его эффекты в прецедентно-ориентированном решении проектных задач.

3. Декларативный тип должен быть ориентирован на акты понимания в формировании постановки задачи и логическую проверку их приращений в пошаговом вопросно-ответном анализе.

4. Концептуально–алгоритмический тип должен обслуживать ни только визуальную поддержку в построении концептуальных программных решений, но и переход к прототипированию решения.

5. Каждый из видов моделей может быть представлен в двух вариантах одновременно: в виде образно-семантического рисунка (графическое представление) и в виде программной проекции, зафиксированной в QA-протоколе.

6. Разработаны методы и средства образно-семантической поддержки процесса решения проектных задач осуществляют поддержку всех этапов концептуального экспериментирования, начиная от этапа $S_t(Z_i, t_0)$, когда задача представляется набором ключевых слов и постановка задачи еще несформулированно и заканчивая этапами прототипирования пользовательского интерфейса и исполнения и отладки разработанных алгоритмов. На рисунке 2.9 показаны основные этапы в процессе решения задач, где этап $S_t(Z_i, t_i)$, является шагом метода пошаговой детализации, т.е. этот шаг повторяется до получения решения на определенном уровне детализации.

Глава 3. Методологическое обеспечение процесса решения задач на основе моделей

Представление задачи в декларативной, концептуально-алгоритмической и изобразительной проекции позволяет «рассмотреть» её со всех сторон засчет проведения мысленного экспериментирования и активации феномена «интеллектуальное воображение» для ее решения. Логично рассмотреть процессы представления и согласования проекций в разрезе метода итеративного согласования и метода понятийно-образной поддержки, завершив главу примерами задач, которые решались в рамках внедрения.

Каждая графическая модель, в зависимости от её типа (изобразительный, декларативный, концептуально-алгоритмический) создаётся проектировщиком в соответствующей операционной среде специализированного графического редактора, выбор которого доступен в соответствующем пункте меню (рисунок 3.1).

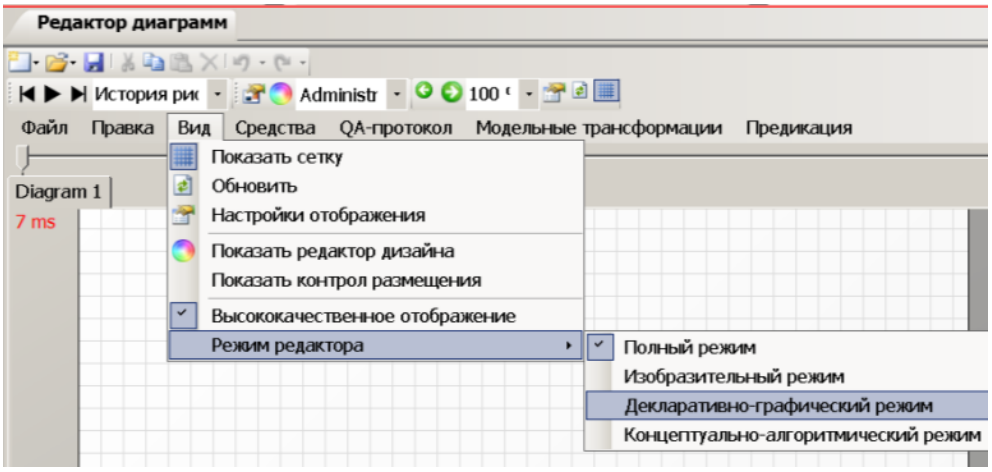

Рисунок 3. 1 - Выбор режима в специализированном графическом редакторе

При выборе режима происходит загрузка соответствующих палитр, изменения списка доступных опций и операций (например, меню «Предикация» доступно при выборе декларативно-графического режима, а меню «Модельные трансформации» для концептуально-алгоритмического). По умолчанию выбирается полный режим, которые включает в себя возможности всех режимов.

Средства образно-семантической поддержки решения проектных задач применимы с момента зарождения задачи, когда постановка еще не сформулирована. С этого момента начинается псевдопараллельное использование средства семантической поддержки процесса решения в контексте прецедента (ориентация на повторное использование). Исследования показали, что для этого потребуется набор моделей, который в диссертации назван типизированным набором и позиционируется как одно из научных положений. Представленные методы раскрываются совокупностью методик, которые являются для них принципиальными; перечень основных из них представлен на рисунке 3.2.

Рисунок 3. 2 - Используемые методики в разработанном комплексе инструментально-технологических средств

Приведённая совокупность основных методик служит основой метода пошаговой детализации, а также подчинённого ему метода согласования понятийного и образного содержания текстовых единиц с использованием их преобразования в прологоподобную форму. К общей специфике методик

относится то, что их использование автоматизировано с активным использованием псевдокодового языка, встроенного в инструментарий WIQA

3.1. Метод итеративного согласования понятийного и образного содержания текстовых единиц с использованием их преобразования в прологоподобную форму

При рассмотрении метода понятийно-образной поддержки процесса пошаговой детализации в прецедентно-ориентированном решении проектных задач не рассматривался вопрос формирования постановки проектной задачи, которая, естественно, уточняется в процессе решения. Второй метод, выносимый в качестве научного результата - метод итеративного согласования понятийного и образного содержания текстовых единиц с использованием их преобразования в прологоподобную форму, специфику которого определяет взаимодополняющее итеративное уточнение графического и текстового представления требований и спецификаций с использованием автоматизированного взаимодействия с онтологией, как раз ориентирован на этот этап и дополняет первый метод.

Рассматривая проектную деятельность, можно сказать, что она направлена, главным образом, на решение задач, которые по своей природе имеют символьно-логическую форму, то есть, ориентированы на представление логических отношений. Это определяет характер когнитивных процессов деятельности проектировщик, когда активируется главным образом левое, логическое полушарие мозга. Для эффективного восприятия проблемной ситуации необходима активация обоих полушарий, поэтому автоматизированный перевод текстового описания проектной задачи в графическую проекцию декларативного представление позволяет лучше понять проблему и, следовательно, решить задачу более эффективно. Такое графическое представление может содержать информацию, недоступную в текстовом описании, за счёт чего достигается ряд положительных эффектов:

1. Выделения предикатов, субъекта, объекта и ассоциативной составляющей позволяет выделить суть постановки задачи, убрать

лишний текстовый «шум», тем самым сделав формулировки более компактными.

2. Образное представление позволяет отображать семантику, недоступную в текстовом представлении, такую как позиционное выделение наиболее значимых объектов (имеющих наибольшее количество входных/исходящих связей; при применении правил расстановки располагаются ближе к центру).
3. Представление субъектов и объектов в виде рисунков позволяет вовлекать образы из ассоциативной памяти и тем самым улучшить процесс запоминания. Метод ассоциаций является одним из основных метод развития памяти и становится основой для большинства техник запоминания [76, 93].

Как сказано выше, любой переход может быть интерпретирован как мыслительный эксперимент, во время которого ментальные образы помогают достичь определенной цели. Когда переход будет направлен на графическую проекцию декларативного типа (при y = D), проектировщик будет пытаться понять пункт $\Delta T_{nr}^{mG}(Z_i, t_j)$ и исправить ее, как показано на рисунке 3.3.

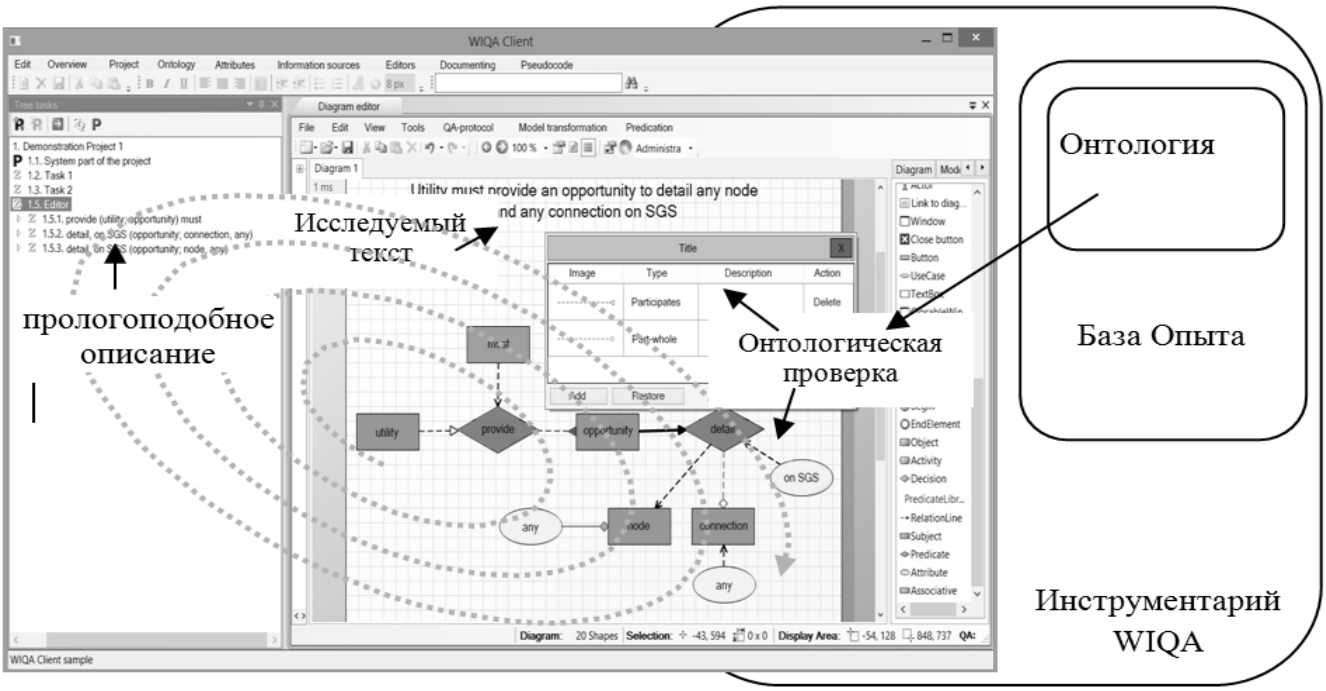

Рисунок 3. 3 - Метод итеративного согласования

Взаимодействуя с текстом $\Delta T_{nr}^{mG}(Z_i, t_j)$, проектировщик создает первоначальную версию своей семантической схемы $M_v^{Gy}(Z_i, t_k)$, что автоматически отражается на соответствующем Пролог-подобном описании. Тогда это описание проверяется с использованием интерпретатора Пролога и, если оно содержит ошибки, то они исправляются проектировщиком. Все изменения программного описания автоматически отражаются в семантической схеме. Проектировщик повторяет эти действия, до того состояния, когда изменения больше не требуются. Это состояние сигнализирует о том, что модели $T_{nr}^{mG}(Z_i, t_j)$ и $M_v^{Gy}(Z_j, t_k)$ являются взаимно согласованными, ошибки отсутствуют, а проектировщик понял текст $\Delta T_{nr}^{mG}(Z_i, t_j)$. Достижение необходимого уровня понимания является основной целью концептуального экспериментирования.

Процес формирование постановки задачи, как было отмечано выше, является итерационным и может начинаться как с формирования графической модели, так и с формирования предекатной формы в вопросно-ответном протоколе. Предикатная форма в вопросно ответном протоклеимеет вид:

<Предикат> (<Субъект> ; <Объект>) <Ассоциативная составляющая>

Для предиката, субъекта и объекта через запятую можно указать свойства – слова (словосочетание) отвечающие на вопрос «Какое?», «Какой?» и т.д.:

<имя субъекта>, <свойство1, свойство2, …>.

В настоящий момент выделение предикатных форм из текста постановки задачи происходит не автоматически, но это принципиально возможно, так как сводится к анализу структуры предложения (выделения сказуемого, подлежащего, дополнения и т.д.), которое успешно проводится для ограниченного набора слов русского языка. Дальнейшее развитие уточнение модели сводится к автоматической трансформации ее в графическую проекцию (текстовую) декларативного представления, уточнение и согласование с исходной моделью. Примеры применения указанных методов в процессе решения проектной задачи приведены в конце главы.

Формальное описание вопросно-ответной проекции декларативного представления

Определим грамматику языка L_d, вопросно-ответной проекции декларативного представления задачи и определим допустимый для них синтаксис с помощью БНФ нотации.

Грамматика языка L_d будет иметь вид:

$$L_d =< S, P, A, Pr, R >$$

где S – множество субъектов и объектов;

P – множество предикатов;

A – множество ассоциативных составляющих;

Pr – множество свойств;

R – множество правил грамматики, определяющих синтаксис языка.

В БНФ форма она может быть представлена:

<ДВО> ::= <ПР> '(' <СБ> ')' | <ПР> '(' <СБ> ')' <АС>

<ДВО> ::= <ПР> '(' <СБ> ';' <ОБ> ')' | <ПР> '(' <СБ> ';' <ОБ> ')' <АС>

<АС> ::= <СЛОВО> | <СЛОВО> ',' <АС>

<СБ> ::= <СЛОВО> | <СЛОВО> ',' <СВОЙСТВО>

<ОБ> ::= <СЛОВО> | <СЛОВО> ',' <СВОЙСТВО>

<СВОЙСТВО> ::= <СЛОВО> | <СЛОВО> ',' <СВОЙСТВО>

<СЛОВО> ::= ‘a’..’Z’ | ‘a’..’Я’ | 0..9

Где ДВО – вопросно-ответное представление декларативной модели, ПР – предикат (в простом предложение часто имеет вид сказуемого), СБ – субъект (часто представляется подлежащим), ОБ – объект (дополнение), АС – ассоциативная составляющее, реализующая функции связи в исследуемом тексте.

Средства поддержки декларативного представления

Средства поддержки декларативного представления обладают широкими возможностями, начиная от создания графической модели и перевода ее в вопросно-ответное представление, и заканчивая взаимодействием с базой онтологий и исполнением в пролог-интерпретаторе. Основой согласования

служит обратная связь между текстом и графикой, способствующая переносу любых «позитивных изменений» от текста на графику и наоборот. Для проверки корректности прологоподобных описаний из программы эксперимента вызывается версия Swi-Prolog интерпретатора Пролога. Для проверки корректности графики и её уточнения используются обращения к Онтологии проекта и Базе Опыта. Диаграмма вариантов использования представлена на рисунке 3.4.

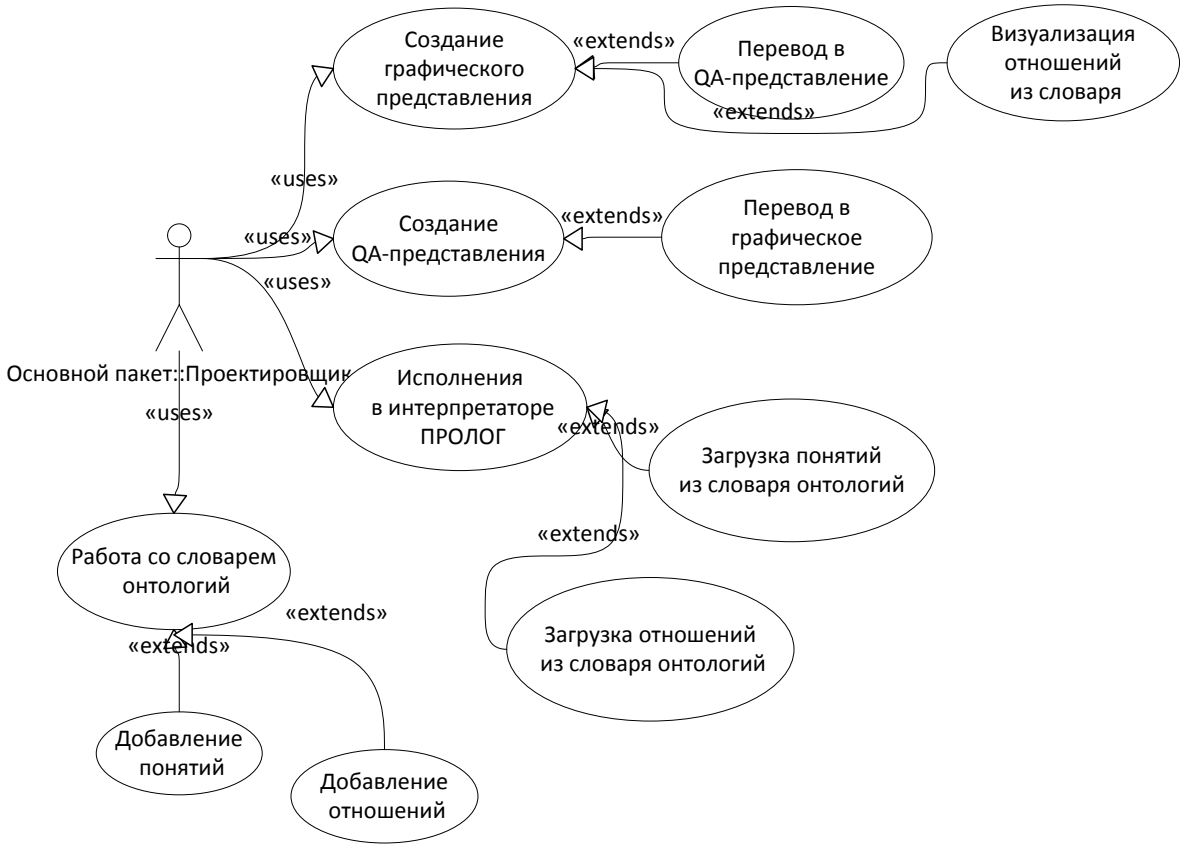

Рисунок 3. 4 - Диаграмма вариантов использования средства поддержки декларативного представления

В общем виде алгоритм создания декларативного представления и сохранение его в вопросно-ответном протоколе состоит из следующих шагов:

1. Выбор соответствующего режима в специализированном графическом редакторе
2. Создание декларативной модели с помощью палитры, которая состоит из:
    a. Предикат. Параллелограмм зелёного цвета.

   b. Субъект и объект. Прямоугольник синего цвета.
   c. Свойства объекта или субъекта. Эллипс жёлтого цвета
   d. Ассоциативная составляющая. Ромб красного цвета
3. Редактирование свойств вершин через панель «Свойства»
4. Создание отношений между элементами. В онтологическом словаре WIQA существуют 14 отношений [102], доступных в графическом редакторе:
   a. часть-целое;
   b. участвует;
   c. синонимы;
   d. реализация;
   e. пространственная ассоциация;
   f. причина-следствие;
   g. по сходству;
   h. по смежности;
   i. по контрасту;
   j. наследование;
   k. инструмент;
   l. выполняет;
   m. временная ассоциация;
   n. атрибут.
5. Отношение между понятиями могут состоять из нескольких видов, причем пользователь может создавать свои типы отношений и записывать их в онтологию. Группы отношений будут представляться «жирной» стрелкой при клике, на которую будет происходить показ всех отношений между понятиями.
6. Перевод и сохранение полученной диаграммы в вопросно-ответный протокол через пункт меню «Предикация».

7. В случае наличия ошибок при сохранении в вопросно-ответный протокол будет показано соответствующее окно. В таком случае процесс исправления ошибки может быть произведен по текущей методике (начиная с пункта 1).

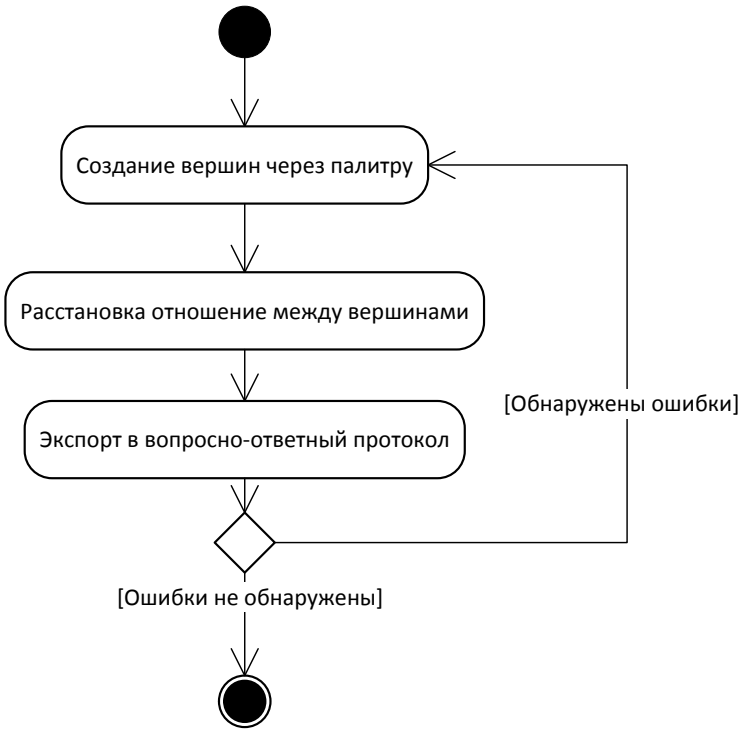

Рисунок 3. 5 - Методика создания графической проекции декларативного представления

Диаграмма активностей описанной методики представлена на рисунке 3.5.

Методика исполнения декларативного представления в интерпретаторе ПРОЛОГ.

После перевода графического представления в вопросно-ответное представление, становится возможным автоматический перевод и исполнение представленных фактов и правил на интерпретаторе пролог с целью изучения (экспериментирования) предметной области.

При проектировании очень важно иметь быстрый доступ к словарю предметной области или базе знаний. В среде WIQA были реализованы средства поддержки работы со словарем онтологий. В словаре онтологий хранятся связи между различными терминами предметной области. Поиск отношений в словаре

происходит динамически (при выполнении запроса), но возможно и ручное добавление отношений в словарь путем выполнения команд вида:

```
assert(отношение(“целое-часть”, ”совокупность моделей”, ”графическая модель”)).
assert(отношение(“включает”, ”совокупность моделей”, ”графическая модель”)).
…
```

На этапе экспериментирования с постановкой задачи может возникнуть ситуация когда проектировщику неизвестны некоторые термины предметной области или отношения между ними. В данном случае можно вызвать следующую команду и получить ответ (пример из рис. 3):

```
отношение(X, ”совокупность моделей”, ”графическая модель”).
X=“целое-часть”
X=“включает”
```

Диаграмма вариантов использования интерактивной консоли представлена на рисунке 3.6

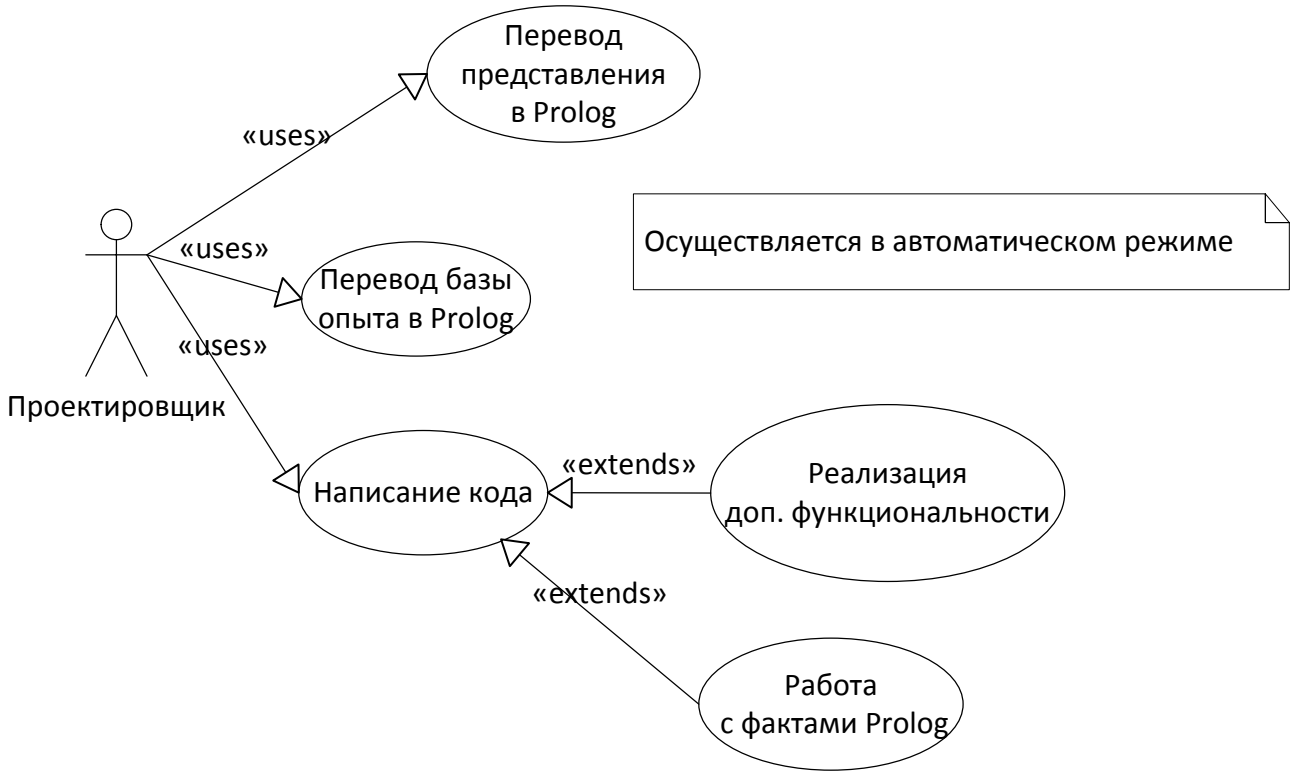

Рисунок 3. 6 - Диаграмма вариантов использования исполнения декларативного представления в интерпретаторе ПРОЛОГ

Методика работы в интерактивной консоли ПРОЛОГ представляет собой итерационный процесс и состоит из следующих шагов:

1. Экспериментирование с предметной областью
    a. Добавление новых фактов. Для добавления новых фактов в предметную область используется стандартный синтаксис;
    b. Добавление новых правил (отношений);
    c. Использование стандартных языковых конструкций языка ПРОЛОГ (вывод на консоль, циклы, массивы и т.д.).
2. Обращение к словарю онтологий
    a. Поиск определений для различных терминов;
    b. Поиск отношений между сущностями.

На рисунке 3.7 показан общий вид редактора с согласованным вопросно-ответным и графическим представлением и интерактивной консолью.

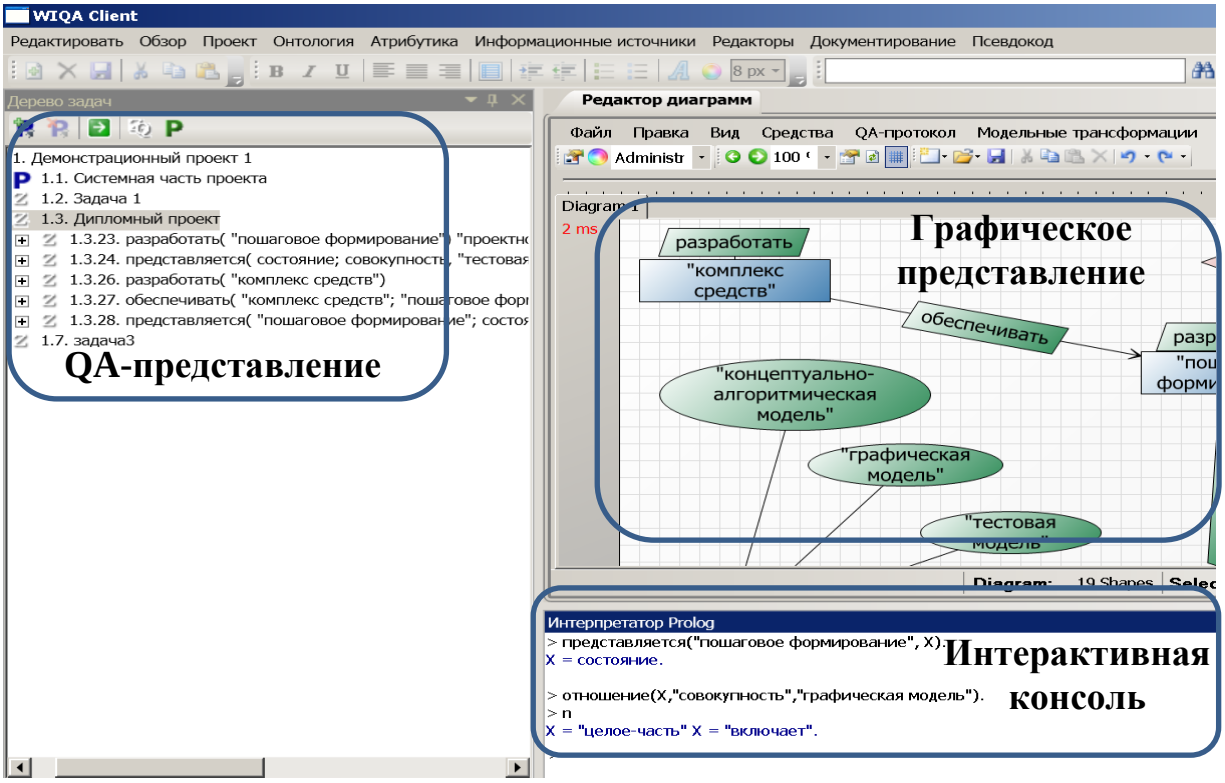

Рисунок 3. 7 - Общий вид редактора в декларативном режиме

3.2. Метод понятийно-образной поддержки процесса пошаговой детализации в прецедентно-ориентированном решении проектных задач

В основе метода положено отображение процессов концептуального решения проектных задач на семантическую память инструментальной среды WIQA, обобщённое представление которой приведено на рисунке 3.8.

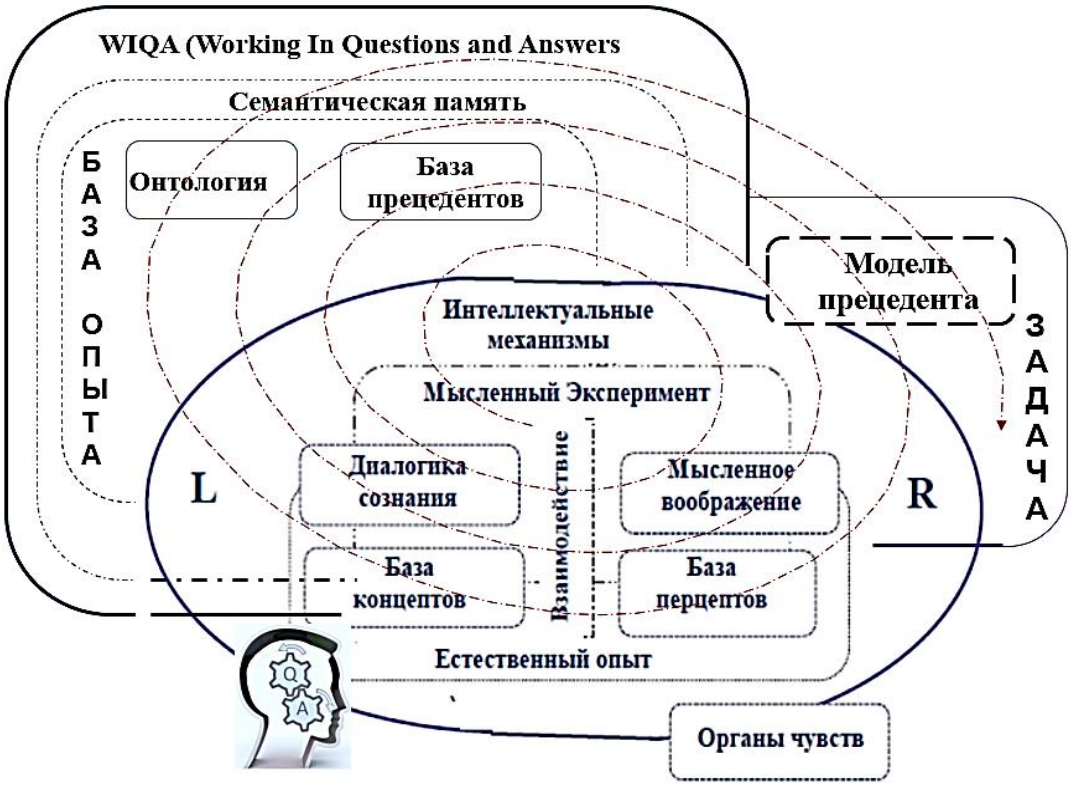

Рисунок 3. 8 - Среда концептуального решения проектных задач

На схеме показано, что в процессе решения задачи проектировщик, решая задачу и создавая для неё модель прецедента, проводит автоматизированные мыленные эксперименты, используя базу (моделей) опыта, включающую «Онтологию» проектирования и «Базу прецедентов». В автоматизацию любого из мысленных экспериментов конструктивно вовлечены диалогика сознания и мысленное воображение, взаимодействующие с естественным опытом, включающем прообразы базы концептов (онтология, левое полушарие L) и базы перцептов (правое полушарие R).

За «спиралью» на схеме стоит реализация метода понятийно-образной поддержки процесса пошаговой детализации в прецедентно-ориентированном

решении задач, обобщённое представление которого приведено на рисунке 2, где центральное место занимает нормативная структура прецедента P, включающая текст постановки задачи P^T, вопросно-ответную модель P^{QA} его анализа, логическую модель P^L, графическое представление P^G, исходный исполняемый псевдокод P^I и программное представление P^E прецедента.

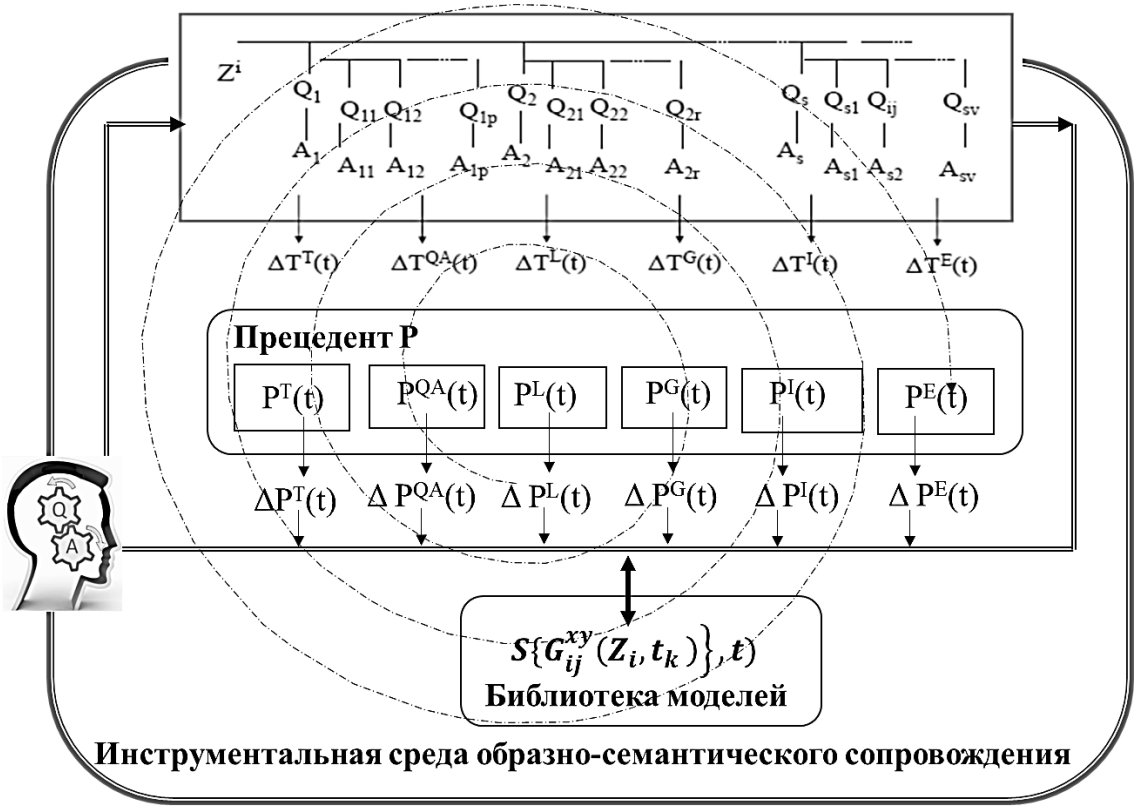

Рисунок 3. 9 - Метод пошаговой детализации постановки задачи

Специфику метода определяют следующие его составляющие:

1. Итеративная реализация метода управляется текущими результатами $\{\Delta T^X(t),\ X=(T, QA, L, G, I, E)\}$ пошаговой детализации постановки задачи $T^P(Z, t)$, регистрация которых проводится в дереве $P^{QA}(t)$ вопросно-ответного анализа.
2. Каждое приращение $\Delta T^X(t)$ способно привести к приращению $\Delta P^X(t)$ соответствующей составляющей прецедента P, среди которых для образно-семантической поддержки принципиальное место занимают образно-семантические модели, которые аккумулируются в библиотеку моделей.

3. Каждая графическая модель, в зависимости от её типа (изобразительный, декларативный, концептуально-алгоритмический) создаётся проектировщиком в соответствующей операционной среде специализированного графического редактора.
4. Каждую графическую модель, если в этом имеется необходимость, проектировщик может подключить к соответствующему блоку постановки задачи.
5. Для повышения эффективности визуального моделирования, проектировщику предоставлена возможность автоматизированного преобразования типов (рисунке 3.9) для создаваемых или созданных графических моделей.

Формальное описание вопросно-ответного представления концептуально-алгоритмической проекции

Определим грамматику псевдокодового языка L_{wiqa}, который используется для вопросно-ответного представления концептуально-алгоритмической проекции задачи, и определим допустимый для них синтаксис с помощью РБНФ нотации [100, 80]:

<letter> ::= a | b | ... | z | A | B | ... | Z
<digit> ::= 0 | 1 | 2 | 3 | ... | 9
<ident> ::= <letter>|<ident><letter>|<ident><digit>
<identifier> ::= &<ident>&
<var_list> ::= <identifier>|<var_list>,<identifier>
<type> ::= int|float|bool|string
<statement-list> ::= <statament>|<statement-list>; <statament>
<label> ::= label <identifier>
<statement> ::= <assign>; | <input st>; | <output st>; | <defined loop>; | <indefined loop>; | <condinion st>; | <label>; | <identifier>;
<goto>::= goto <ident>
<statement_list> ::= <statement>;|<statement_list><statement>;
<body> ::= BEGIN <statement_list> END | <statement>;
<assign> ::= <ident> := <expression>
<condition st> ::= IF <comparison> THEN <body> | IF <comparison> THEN <body> ELSE <body>
<defined loop>::= FOR <identifier> := <expression> TO <expression> DO <body>
<indefined loop> ::= WHILE <comparison> DO <body> | REPEAT <body> UNTIL <comparison>
<expression> ::= <term> | <expression> + <term> | <expression> - <term> | <expression> or <term> | <expression> and <term> | not <expression>
<term> ::= <factor> | <term> * <factor> | <term> / <factor>
<factor> ::= <ident> | int | (<expression>) |<function_name> () | <function_name>|(<parameter_list>)

<parameter_list> ::= <expression>|<parameter_list>, <expression>
<comparison> ::= <expression> <relation> <expression>
<relation>::= < | <= | >= | <> | =
<procedure>::=PROCEDURE <identifier> (<var_list>) {<statement_list>}

Этот язык встроен в WIQA и используется как основное средство для программирования решения, так как включает средства интеграции со всеми компонентами.

Методика создания концептуально-алгоритмического представления

Концептуально-алгоритмическое представление ориентировано на использование 3 типов диаграмм UML 2 версии: диаграмм активностей (деятельности), диаграмм вариантов использования и диаграмм классов. Оно позволяет переводить их в псевдокодовое представление и выполнять в интерпретаторе, который встроен в WIQA. Для удобства расширения возможностей поддерживаемых видов диаграмм, правила трансформаций загружаются из текстового файла, который доступен для непосредственного редактирования.

Данная функциональность позволяет переводить диаграммы в алгоритмический код (псевдокод), путем выявления различных семантических блоков, таких как условия, циклы, потоки и т.д. Под семантическими блоками не следует понимать только структуры диаграмм активностей. Таким блоком может выступать любая совокупность графических элементов (подграф), имеющая сколько угодно много связей между собой, но обладающая входной (может иметь входы снаружи блока) и исходящей (может иметь выход из блока) вершинами. Правила для графовых трансформаций представляют собой пары графов, имеющих обозначения Left Hand Side(LHS) и Right Hand Side (RHS). Алгоритм поиска и свертки правил представлен на рисунке 3.10.

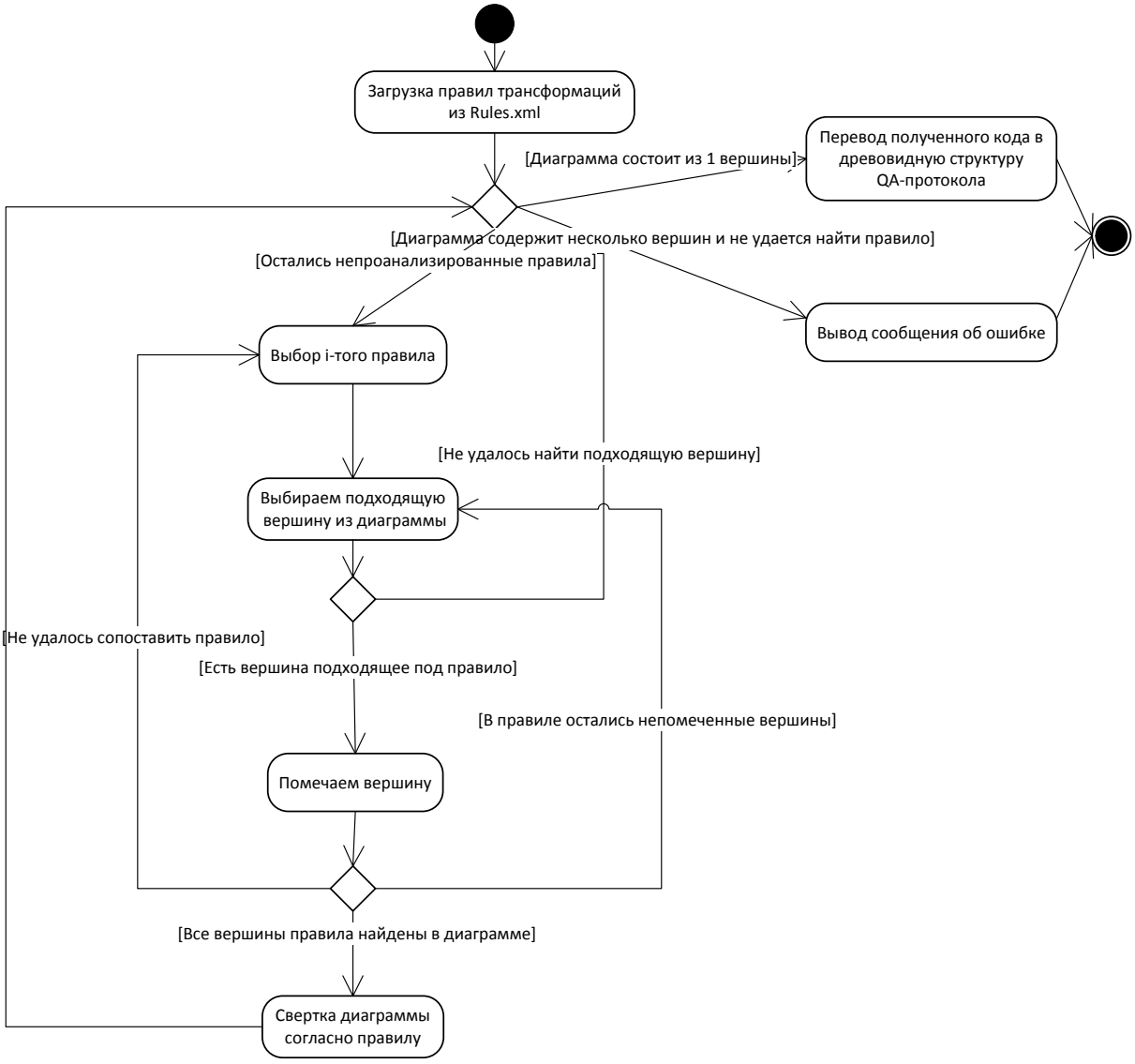

Рисунок 3. 10 - Алгоритм поиска и свертки правил в диаграмме

При применении правила происходит поиск всех подграфов, соответствующих LHS и замена найденных по соответствующему правилу на RHS. В нашем случае, граф для замены является вырожденным (множество связей является пустым множеством). Это необходимо для того, чтобы на каждом этапе получать более простой граф и в конечном итоге, путем ряда сверток получить граф, состоящий из одной вершины. Пример простой трансформации приведен на рисунке 3.11:

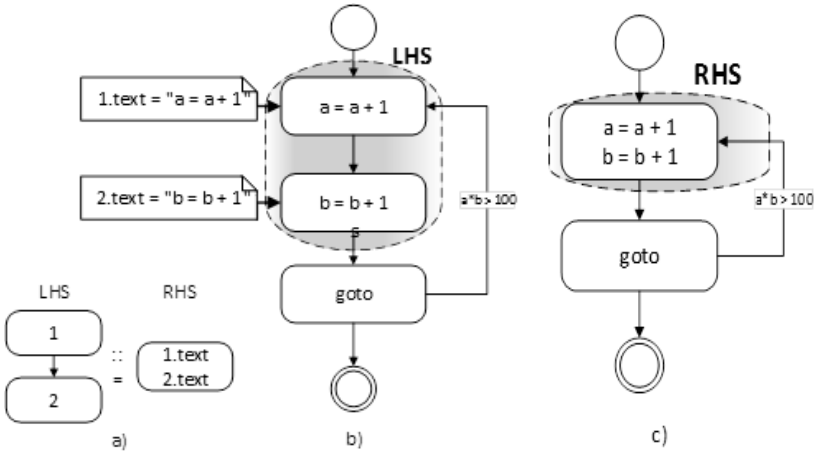

Рисунок 3. 11 - Пример применения простого правила трансформации: a – правило трансформации, b – найденное правило, c – граф, получившийся после применения правила

Для поиска концептуальных ошибок предусмотрено исполнение вопросно-ответного представления в псевдокодовом интерпретаторе среды WIQA. На рисунке 3.12 представлена диаграмма вариантов использования.

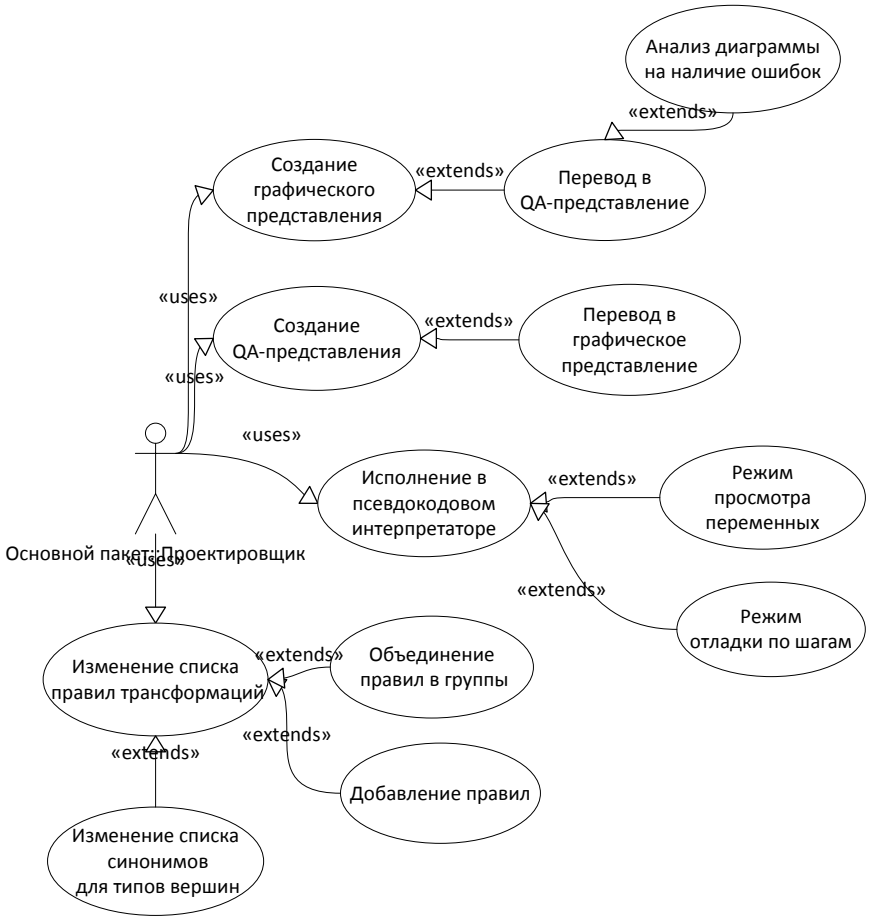

Рисунок 3. 12 - Диаграмма вариантов использования для средства концептуально-алгоритмической поддержки решения задач

Таким образом методика формирования концептуально-алгоритмического представления состоит из следующих шагов:

1. Разработка графического или вопросно-ответного представления и сохранение его в QA-протокол.
2. Выполнение QA-протокола в псевдокодовом интерпретаторе WIQA с целью нахождения концептуальных и иных ошибок.
3. Корректировка решения для исправления найденных ошибок.
4. Визуализация полученного результата.
5. Шаги 1-4 обычно выполняются несколько раз с целью последовательного уточнения (улучшения) и исправления ошибок.

Методика реализации дополнительных правил трансформации

При разработке средства делался акцент на простоту реализации поддержки других видов диаграмм и правил трансформаций. Как было отмечено выше, правила трансформации представляют собой пары графов LHS и RHS. На практике правила трансформации содержат дополнительную информацию, необходимую для сохранения данных о семантическом блоке после его свертки. Такой информацией является псевдокод, полученный после применения трансформации. На рисунке 3.13 представлена небольшая часть файла правил для трансформации блока «IF» диаграммы активностей.

```
- <rule name="if 2 action">
    - <items>
        <item id="1">ActyvityDiagram_Pal.AD_Condition_Element</item>
        <item id="2">ActyvityDiagram_Pal.AD_Action_Element</item>
        <item id="3">ActyvityDiagram_Pal.AD_EndCondition_Element</item>
        <item id="4">ActyvityDiagram_Pal.AD_Action_Element</item>
      </items>
      <in>1</in>
      <out>3</out>
    - <connections>
        <connection id="1cn" note="" to="2" from="1"/>
        <connection id="" note="" to="3" from="2"/>
        <connection id="2cn" note="" to="4" from="1"/>
        <connection id="" note="" to="3" from="4"/>
      </connections>
      <code>IF (1cn.text) THEN\r\n\t2.text\r\nelse if(2cn.text) THEN\r\n\t4.text\r\nEND</code>
      <collapseElementType>ActyvityDiagram_Pal.AD_Action_Element</collapseElementType>
  </rule>
```

Рисунок 3. 13 - Часть файла с описанием правила трансформации для блока “IF”

Каждое правило описывается следующими тегами:

- Тег items. Содержит массив всех вершин семантического блока с указанием их типов (в палитре) и уникальных идентификаторов (атрибут id). Атрибут «id» используется далее для описания связей, входных и выходных вершин;
- Тег in. Содержит идентификатор входной вершины – вершины, которая имеет входы «снаружи»;
- Тег out. Содержит идентификатор выходной вершины – вершины, которая имеет выход «наружу»;
- Тег connections. Содержит информацию о связях внутри блока. Связь выходит из вершины «from», входит в вершину «to», имеет идентификатор «id» (необходим для формирования псевдокода) и подпись «note». Если подпись не задана, то считается, что она может принимать любое значение. Значение id рекомендуется задавать вида “<число>cn”;
- Тег code. Содержит правило для получения кода вершины. Может содержать макрос “<id>.text”, вместо которого будет подставлено текстовое значение связи или вершины;
- Тег collapseElementType. Содержит тип вершины, которая будет создана после того как будет применена трансформация. Тип вершины рекомендуется выбирать в рамках используемой палитры.

Правила можно объединять в группы (конвекторы), задавать синонимы для типов вершин, объединять трансформации в цепочки. Последнее необходимо для того, чтобы при наличии правил трансформации моделей («A» в «B») и («B» в «C») получить преобразование из («A» в «C»). Формат описания синонимов представлен на рисунке 3.14:

```
<?xml version="1.0"?>
- <convectors>
    + <convector text_regexp="([А-Яа-я]+[А-Яа-я0-9\ \t]+[\?]?)" name="diagram to code">
    + <convector text_regexp="([А-Яа-я]+[А-Яа-я0-9\ \t]+)" name="diagram to PROMELA">
    + <convector text_regexp="([А-Яа-я]+[А-Яа-я0-9\ \t]+)" name="oracle BPEL to diagram">
    - <groupConvector name="oracle BPEL to PROMELA">
          <convector name="oracle BPEL to diagram"/>
          <convector name="diagram to code"/>
      </groupConvector>
    - <synonyms>
          <synonym to="ActyvityDiagram_Pal.AD_Begin_Element" from="u_m_l_activity_shapes._begin"/>
          <synonym to="ActyvityDiagram_Pal.AD_End_Element" from="u_m_l_activity_shapes._end_element"/>
          <synonym to="ActyvityDiagram_Pal.AD_Condition_Element" from="u_m_l_activity_shapes._decision"/>
          <synonym to="ActyvityDiagram_Pal.AD_EndCondition_Element" from="u_m_l_activity_shapes._end_decision"/>
          <synonym to="ActyvityDiagram_Pal.AD_Action_Element" from="u_m_l_activity_shapes._activity"/>
      </synonyms>
  </convectors>
```

Рисунок 3. 14 - Формат описания синонимов

Таким образом методика добавления нового правила трансформации состоит из следующих шагов:

1. Анализ существующих правил трансформации. Часто возникает ситуация, что правило трансформации уже есть, но описано для других типов вершин. В таком случае необходимо добавить соответствующее описание в раздел синонимов;
2. Описание правила трансформации
    a. Описание всех вершин из LHS. При описании для каждой вершины задается уникальный идентификатор и тип;
    b. Указание идентификаторов входной и выходной вершины;
    c. Описание связей между вершинами. При описании связей используются идентификтаторы вершин, проставленные ранее;
    d. Указание правил трансформации текста из исходных вершин. Доступны макросы “text”, как для вершин так и для связей;
    e. Указание типа вершины, которая получится после применения правила;
3. Проверка правильности нового правила. При загрузке трансформаций, в случае возникновения ошибок, будет выведено соответствующее сообщение.

На рисунке 3.15 представлен общий вид редактора в концептуально-алгоритмическом режиме.

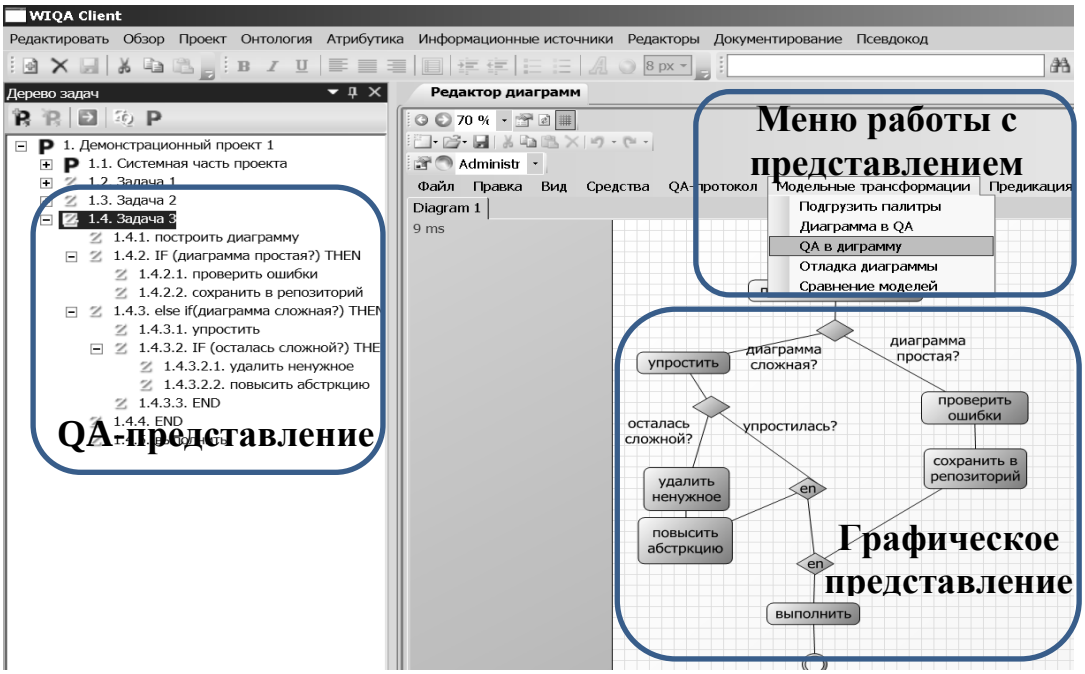

Рисунок 3. 15 - Общий вид редактора в концептуально-алгоритмическом режиме

Методика формирования изобразительного представления

Изобразительное представление существенно отличается от декларативного и концептуально-алгоритмического, так как не требует использования специальных палитр и средств – в изобразительное представление (QA - протокол) можно перевести любую диаграмму в любой палитре. Поэтому, изобразительное представление выступает как “основа”, и все остальные представления могут быть переведены в него. На рисунке 3.16 диаграмма вариантов использования, описывающая возможности проектировщика

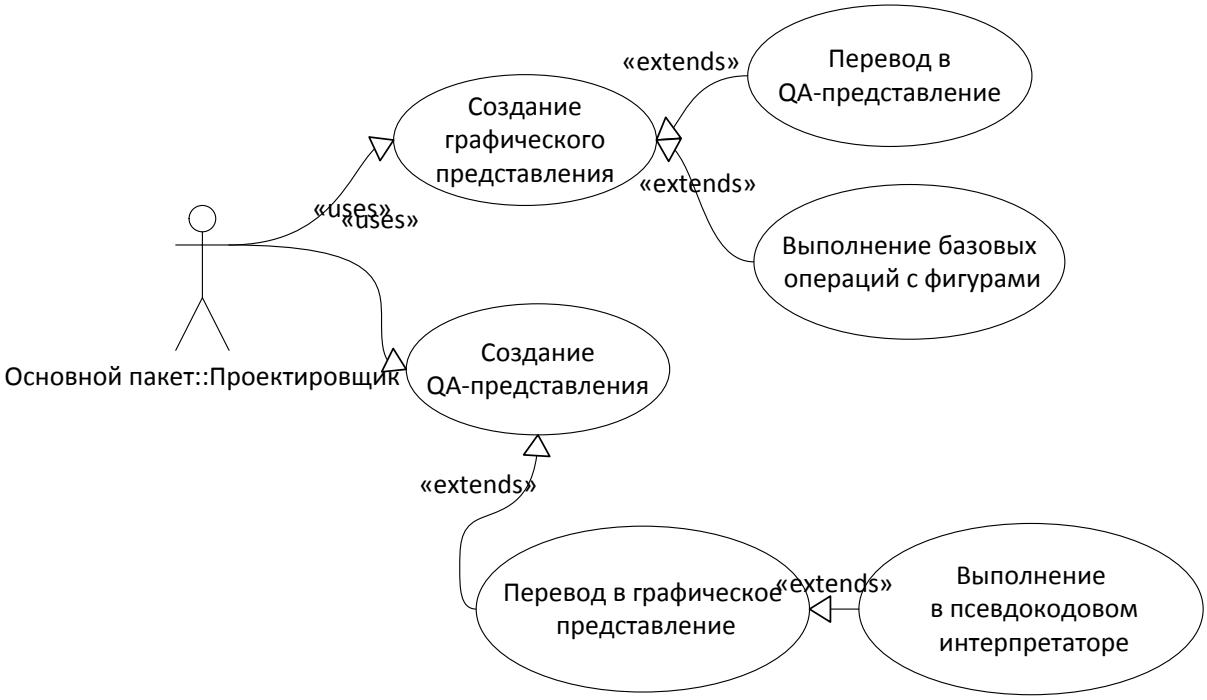

Рисунок 3. 16 - Диаграмма вариантов использования для изобразительного представления

Как и для остальных типов представлений, изобразительное может формироваться как через QA-протокол (с последующим созданием диаграммы), так и через создание примитивов в специализированном редакторе (и последующей генерации из него QA-представления).

Для представления в вопросно-ответном протоколе предусмотрены следующие псевдокодовые функции, которые были реализованы в рамках диссертационного исследования [85] и адаптированы для использования в изобразительном, декларативном и концептуально-алгоритмическом представлениях (этот список ограничен только функциями, которые были реализованы в рамках других исследований; весь перечень с учетом новых функций приведен в главе 4):

- DD_Create – функция используется для создания нового графического элемента на основе существующей палитры и позволяет задавать различные параметры:
  - положение на холсте (X, Y);
  - размеры фигуры (Width, Height);

    - угол поворота фигуры (Angle);
    - текст, присутствующий на фигуре (Text);
    - стиль текста (доступно большое количество свойств);
    - стиль заливки графического примитива (доступно большое количество свойств);
    - стиль линий графического примитива (доступно большое количество свойств);
- DD_Update – функция обновления существующего графического элемента. Доступно изменение тех же свойств, что и при создании примитива. В качестве идентификатора выступает имя фигуры.
- DD_Delete – удаление графического элемента по имени.
- DD_Link – связывание двух графических фигур, определенной связью (в палитрах доступные различные типы связей)
- DD_LinkDiagram – функция позволяет связать фигуру с произвольной диаграммой (файлом) таким образом, что при клике на элемент откроется связанная диаграмма. Принимает следующие параметры:
    - наименование фигуры, при клике на которую должен быть осуществлен переход на другую диаграмму;
    - идентификатор вопросно-ответной единицы, соответствующей диаграмме, на которую должен быть осуществлен переход.

На рисунке 3.17 показан пример программной проекции изобразительного представления в вопросно-ответном протоколе.

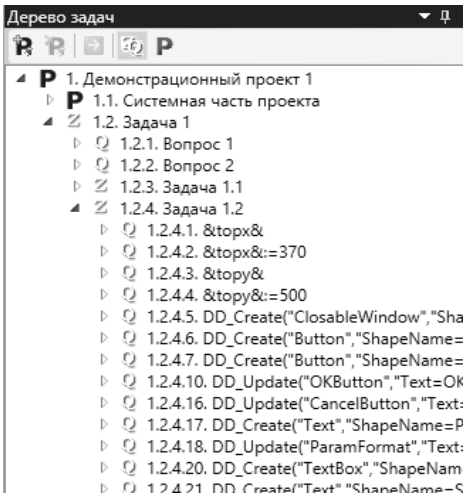

Рисунок 3. 17 - Пример изобразительного представления в вопросно-ответном протоколе

Методика концептуального экспериментирования с представлениями

В основу разработанного комплекса средств положен процесс пошагового формирования (stepwise refinement), который подразумевает движение «сверху-вниз» путем уточнения и представляет собой переход от абстрактного описания системы к подсистемам, из которых она состоит, и далее, до необходимого уровня детализации. Подход хорошо себя зарекомендовал как при разработке систем (принцип ООП), так и в подходах Model Driven Development. Процесс пошагового формирования основан на итеративном уточнении различных представлений (текстовое, декларативное, вопросно-ответное и т.д.), который может осуществляться как в прямом направлении, так и путем возврата на несколько шагов назад с целью корректировки. Отправной точкой для такого процесса может служить текстовая модель (техническое задание). Сформулированная в виде вопросов-ответов (после проведения вопросно-ответного анализа) такая модель может быть транслирована в графическую. Графическая модель находится в согласование с текстовой, поэтому такой переход является обратимым. Из-за этого становится не принципиальным, какая модель (текстовая, графическая или концептуально-алгоритмическая) станет отправной точкой, формирование решения может происходить в одной из представленных проекций.

В общем виде процесс согласования состоит из следующих этапов:

1. Согласование декларативного представления

a. Создание декларативного представления и сохранением его в QA-протокол.
b. Выполнение протокола в интерпретаторе ПРОЛОГ, работа со словарем онтологий. Целью является экспериментирование, выявление противоречий, неполноты и т.д.
c. Изменение вопросно-ответного представления для исправления найденных ошибок.
d. Визуализация вопросно-ответного представления с целью образного восприятия постановки задачи.
e. Повторение этапов a-d до полного согласования.

2. Согласование концептуально-алгоритмического представления
   a. Создание концептуально-алгоритмического представления и сохранением его в QA-протокол.
   b. Выполнение задачи в псевдокодовом интерпретаторе WIQA. Целью является экспериментирование, выявление алгоритмических (концептуальных ошибок).
   c. Изменение вопросно-ответного представления согласно найденным ошибкам.
   d. Визуализация вопросно-ответного представления с целью образного восприятия алгоритма.
   e. Повторение этапов a-d до полного согласования.

Следует отметить, что задача может содержать несколько представлений одного вида (например, в рамках концептуально-алгоритмического представления можно работать с диаграммами классов, активностей и вариантов использования). В таком случае после согласования каждой модели, возможно, потребуется согласование моделей друг с другом.

Методика работы с интегрированным средством контроля версий

Наличие большого количества моделей требует вовлечение средств контроля версия в процесс согласования и уточнения моделей. Как описывалось

ранее, процесс уточнения может идти не только “сверху-вниз”, но и осуществлять возвраты на несколько шагов назад. Без средств контроля версий такое поведение было бы трудно реализовано.

В качестве основы было выбрано средство с открытым исходным кодом git. В главе 4 рассматриваются принципы интеграции данного средства в инструментальную среду WIQA, здесь будут рассмотрены только методики работы с интегрированным средством. Доступные следующие операции:

1. Создание репозитория.
2. Выбор ранее созданного репозитория.
3. Фиксация изменений.
4. Переключение на определенную версию (зафиксированную ранее).

Инициализация репозитория

Ключевым понятием любого контроля версий является понятие “репозиторий”. Под ним понимается то место (папка на диске), в которое будут складываться и забираться все версии определенной системы. Рекомендуется для каждого проекта иметь отдельный репозиторий между моделями. Методика инициализации состоит из следующих шагов:

1. Выбор соответствующего пункта меню;
2. Выбор папки на диске, в которой будет инициализирован репозиторий.

Следует отметить, что интеграция средства контроля версий в WIQA построена таким образом, что позволяет непосредственно (с помощью системной утилиты git) просматривать и изменять данные в репозитории, что дает дополнительную гибкость в выборе средств работы с моделями. После проделанных шагов система проинициализирует репозиторий и запомнит путь до него.

Фиксация и откат изменений

Для фиксации изменений необходимо выполнить соответствующий пункт меню и ввести сообщение, описывающее сделанные изменения. После этого

произойдет запись изменений в репозиторий и выведется сообщение о количестве зафиксированных изменений.

Для отката на определенную версию в прошлом необходимо выбрать пункт меню “Переключится на версию” и выбрать необходимую версию. Пример диалогового окна для выбора версии представлен на рисунке 3.18

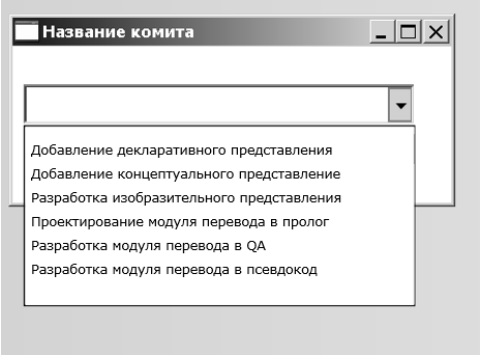

Рисунок 3. 18 - Диалоговое окно выбора предыдущей версии

Средства контроля версий реализованы для активного проекта, таким образом, они не привязаны к типам моделей и диаграмм, используемых в проекте. Это обеспечивает гибкость, и позволяет хранить согласованные модели совместно с другими артефактами (текстовым описанием, внешними файлами и т.д.).

Методика поиска некоторого вида ошибок

При решении задачи могут возникать разного рода ошибки:

- Речевые. К ним относятся лексические ошибки, неверное использование фразеологизмов, использование диалектизмов, устаревших слов, использование многозначных слов там, где это неприменимо и т.д. Перечень ошибок характеризуется языком, на котором описывается задача. Несмотря на свою простоту, такие ошибки могут искажать смысл постановки задачи и приводить к ошибкам, связанным с непониманием.
- Синтаксические. Характеризуются нарушением структуры словосочетаний (предложений).
- Грамматические. Появляются при несоблюдении норм формообразования, норм синтаксической связи между словами.

- **Фактические**. Эти ошибки проявляются как искажение фактов, и связаны с недостаточностью знаний об описываемом объекте, неверная трактовка событий и т.д.
- **Логические ошибки**. Обусловлены нарушением правил логического мышления.

Ошибки, выделенные жирным цветом, представляют особый интерес, так как приводят к значительно более тяжёлым последствиям в случае их появления. Рассмотрим эти ошибки, а также способы их обнаружения в средстве образно-семантической поддержки процессов решения задач.

Фактические ошибки

Как было отмечено ранее, их появление обусловлено неполнотой знаний о предметной области, неправильной трактовки событий, непониманием смысла слов или вызвано наличием речевых ошибок, исказивших семантику задачи.

С точки зрения разработанного подхода эти ошибки могут быть найдены по следующим признакам:

1. Отсутствие отношений между объектами (отсутствие фактов).
    a. Графический вид декларативного представления. Этот вид ошибки означает отсутствие вершины и/или связи между объектом и субъектом (объектом и дополнением). При большом количестве фактов такую ошибку трудно найти путем сравнения текстового описания и декларативной модели. Это значительно проще сделать в прологоподобном виде декларативном представления, выбрав для перевода соответствующий пункт меню, представленный на рисунке 3.19.

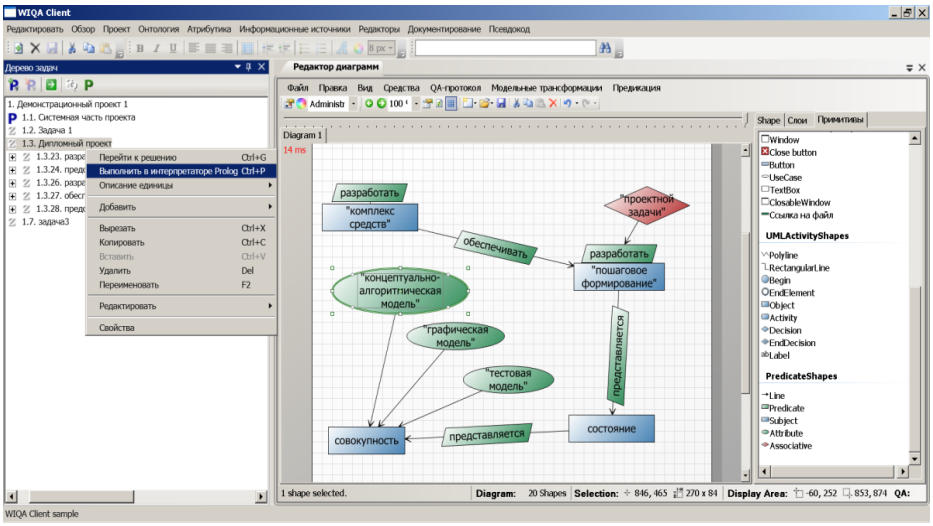

Рисунок 3. 19 - Перевод программной проекции декларативного представления в графическую

8. При исполнении декларативной модели в пролог интерпретаторе с последующим экспериментированием, данный вид ошибок может быть найден по следующим характеристикам:

i. При отсутствии факта будет возвращена ошибка и предложена другие факты (например, другой арности, уже описанные в предметной области задачи). Для примера, который был рассмотрен в главе 2 и представлен на предыдущем рисунке, при вызове метода “разработать” будет возвращена ошибка, так как был описан только метод с арностью 1.

```
> разработать("пошаговое формирование", "базу данных").
ERROR: Undefined procedure: разработать/2
ERROR:     However, there are definitions for:
ERROR:         разработать/1
false.)
```

ii. Возвращение значения false, если факт требуемой арностью существует (был описан ранее), но имеет другое описание (форму).

```
> разработать("базу данных").
false.
```

2. Наличие нескольких решений (фактов) при необходимом наличии только 1 решения (определенного набора фактов). Из примера выше требуется наличие двух фактов:

```
> разработать(X).
X = "пошаговое формирование" ;
X = "комплекс средств".
```

Удалять факты из консоли можно с помощью метода “retract”.

```
> retract(разработать("комплекс средств")).
true.

> разработать(X).
X = "пошаговое формирование".
```

При экспериментировании со сложными предметными областями, состоящими из большого количества фактов и правил, удобно использовать текущий подход. Например, при описании решение классической задачи Эйнштейна в декларативном виде, при исключении части правил возникает ситуация, когда решением являются несколько ответов. Так как из условий задачи вытекает наличие только одного решения, то такое поведение говорит о наличии неполноты представленных фактов (которую, в данном случае, мы сделали специально).

Логические ошибки

В ряду логических ошибок можно выделить две большие группы, связанные с причиной возникновения семантической ошибки. Причины могут быть внешнего характера, то есть предметная область содержит противоречия, которые являются существенными, так как не позволяют найти решение и внутреннего, которые возникают в процессе решения задачи и обусловлены недопониманием задачи проектировщиком. Второй вид ошибок является распространенным при разработки сложных систем и его трудно выявить.

Как отмечалось во второй главе, при постановке задачи необходимо придерживаться определенной структуры, в основе которой лежит наличие логических связей между предложениями и фактами. Структура постановки задач может отличаться, но наличие такой связи является обязательным. Обычно наличие такой связи между соседними предложениями легко прослеживается при чтении, но при увеличении расстояния между фактами в тексте, наличие или отсутствие таких связей уже не так очевидно. При представлении задачи в декларативном виде, расстояния между фактами в тексте перестает иметь смысл и

отсутствие логической связи определяется отсутствием соединения между объектами. На рисунке 3.20 ниже представлен пример, который рассматривался выше, но в нем была допущена орфографическая ошибка, которая привела к отсутствию логической связи (с точки зрения среды — это разные факты) и появлению “висячих вершин”:

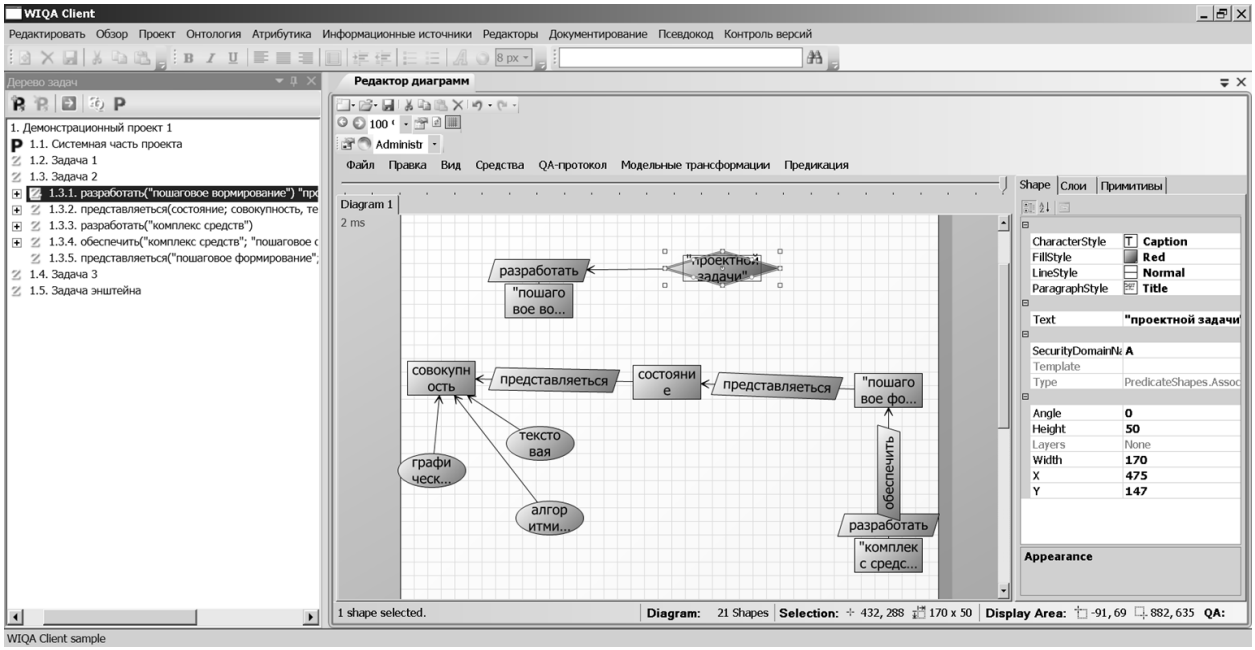

Рисунок 3. 20 - Процесс согласование задачи с декларативным представлением

Следует отметить, что наличие “висячих вершин” не всегда является логической ошибкой, но может также указывать на неполноту (пропуск фактов) или неправильную формулировку задачи. Разработанное средство позволяет упростить нахождение таких мест, требующих повышенного внимания.

3.3. Примеры решения проектной задачи с помощью разработанных методов и средств

Пример решенной задачи в рамках внедрения в ОАО НПО «Марс»

В рамках внедрения в ОАО НПО «Марс» предложенные модели методы и средства были использованы при решении проектной задачи разработки имитатора радиолокационной информации. Программный комплекс предназначен для имитации радиолокационной обстановки, прототипами для которой могут служить различные виды корабельных навигационных РЛС и обзорных РЛС.

Рассмотрим процесс применения разработанных моделей методов и средств, начиная с этапа зарождения проектной задачи и заканчивая этапом получения рабочего прототипа на примере реализации модуля имитации движения объектов.

Шаг 1 Формирования первоначальной постановки задачи и наполнение словаря понятий

Первоначальная постановка задачи может быть сформулирована следующим образом:

Необходимо разработать модуль имитации движения объектов являющийся частью модуля имитации внешней среды и позволяющий моделировать различные виды движений (как прямолинейное так и с произвольной траекторией) в трехмерном пространстве (в том числе с ускорением по все координатам).

Построим декларативную модель постановки задачи в виде программной проекции декларативного представления. Для удобства восприятия введем сокращения:

- МИД – модуль имитации движения
- МИВС – модуль имитации внешней среды

разработать (МИД) позволяет моделировать (МИД; движение) бывает (движение; с ускорением) бывает (движение; прямолинейное) бывает (движение; с произвольной траекторией) осуществляется (движение; в пространстве, трехмерном)

Семантическая граф схема будет иметь вид:

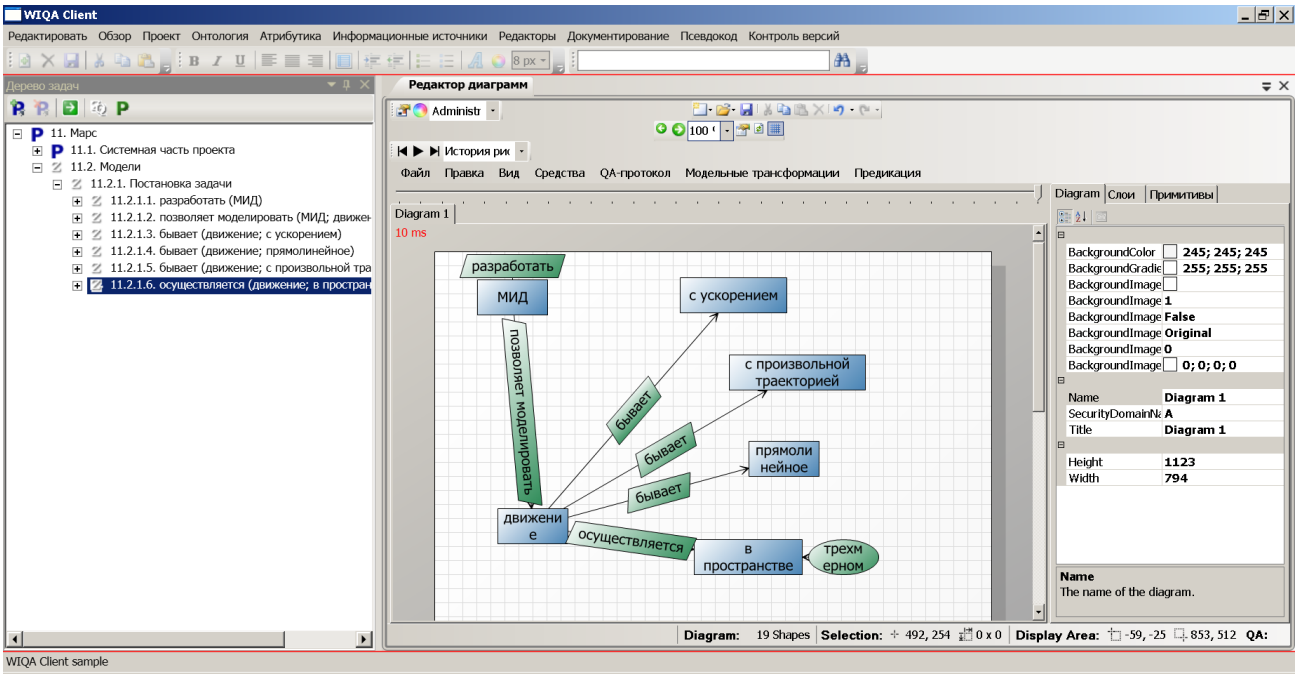

Рисунок 3. 21 - Семантическая граф схема текущей постановки задачи

Так как предметная область является достаточно сложной и содержит большое количество терминов на этом этапе важным является формирование словаря понятий с указанием отношений между ними. Как будет показано далее, это позволит проектировщику в случае необходимости получать определения незнакомых терминов непосредственно через итеративную консоль интерпретатора ПРОЛОГ (SwiPrologProxy). Добавим определения и отношения для терминов из постановки задачи в словарь онтологий WIQA.

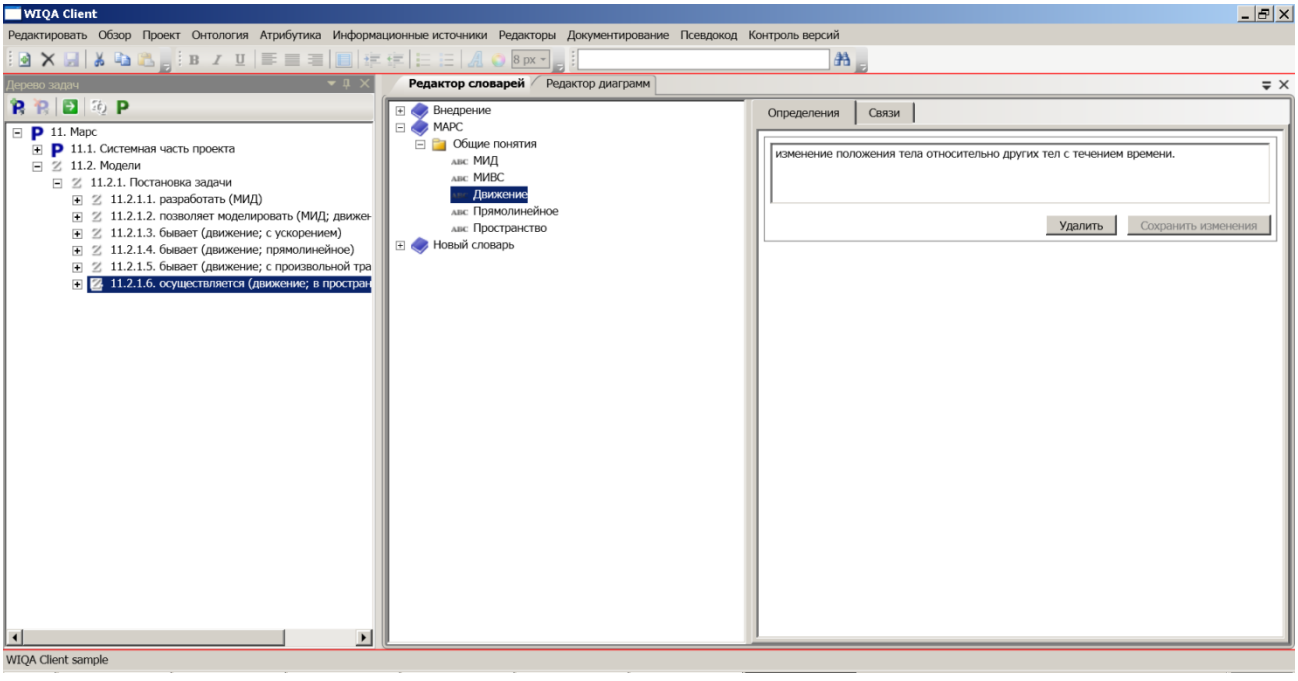

Рисунок 3. 22 - Наполнение словаря понятий в WIQA

При этом связи между понятиями удобно изменять в семантической граф схеме, как показано на рисунке ниже. Следует отметить, что каждый вид отношения имеет собственное представление на графической диаграмме; в случае наличия нескольких отношений (количество отношений между 2 объектами не ограничено) оно будет выделено жирным цветом.

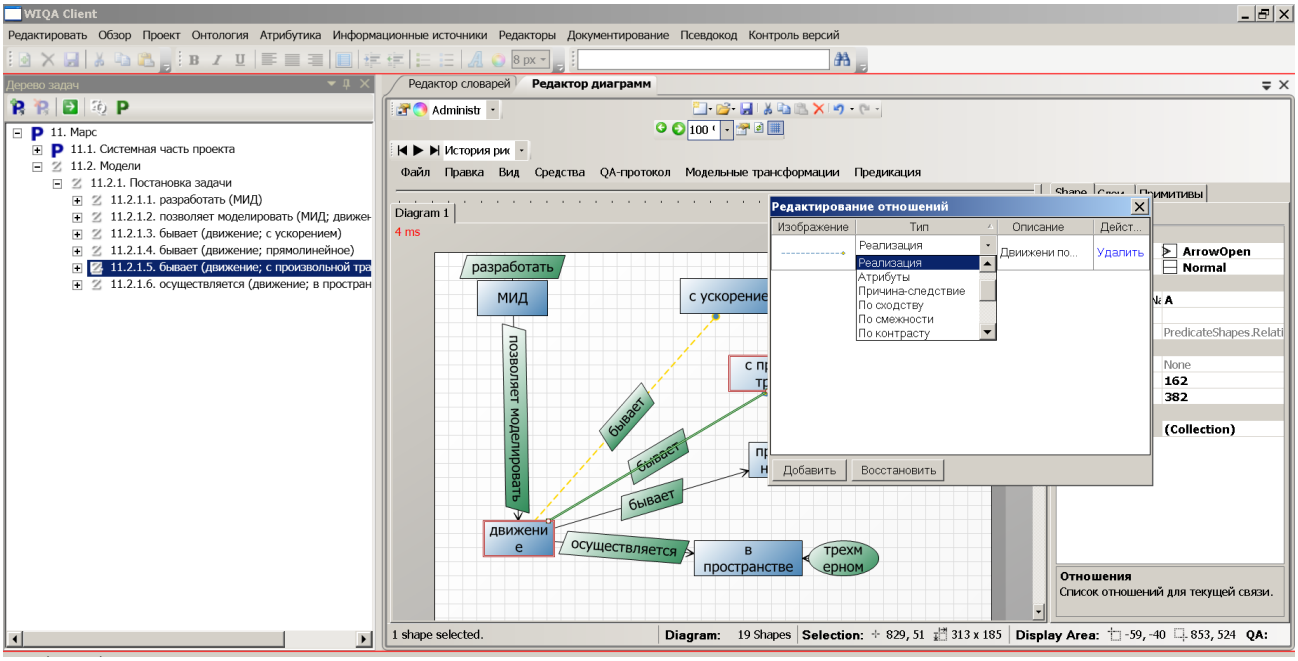

Рисунок 3. 23 - Изменение отношений между понятиями на семантической граф схеме

После формирование словаря понятий становится возможным непосредственно получать определения неизвестных терминов через интерактивную консоль интерпретатора ПРОЛОГ.

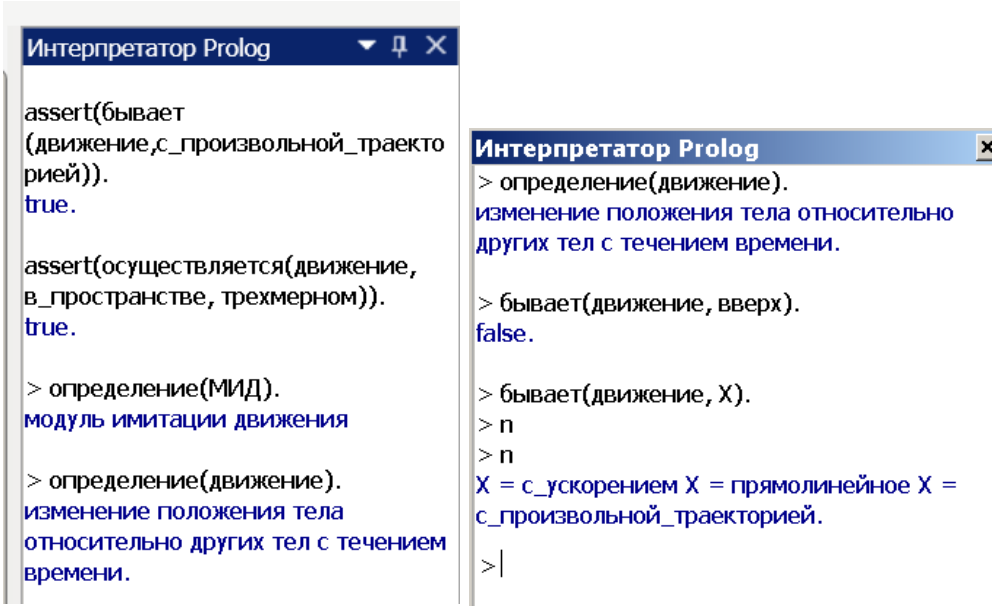

Рисунок 3. 24 - Интерактивная консоль интерпретатора ПРОЛОГ

Методы, выводящие определения и отношения реализованы как расширения декларативного программирования и не исключают использование полноценных конструкций языка ПРОЛОГ как показано на рисунке выше (где задается вопрос интерпретатору относительно видов движения). После завершения каждого этапа необходимо фиксировать текущее состояние вопросно-ответного дерева в адаптированной среде контроля версий. Для этого необходимо создать новый репозиторий и зафиксировать изменения через соответствующий пункт меню, для возможности, в случае необходимости, переключения на предыдущую версию.

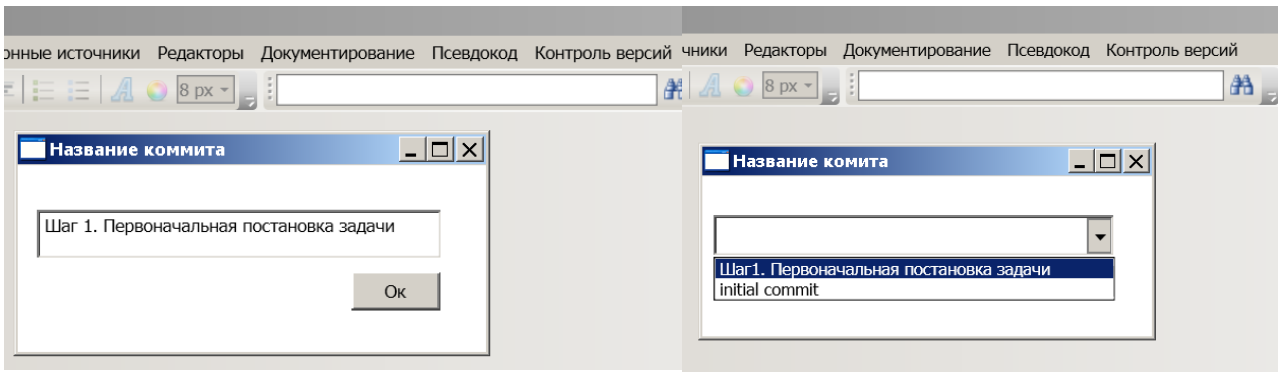

Рисунок 3. 25 - Фиксация изменений в адаптированной системе контроля версий

Шаг 2. Уточнение постановки задачи

Так как первоначальная постановка задачи является очень абстрактной для ее уточнения необходимо провести вопросно-ответный анализ и зафиксировать его результаты в QA-протоколе и декларативном представлении.

Q.1. Каким образом происходит задание произвольной траектории движения объектов?

A.1. Любая сложная траектория представляется последовательностью участков, реализующих различные виды движений, таких как равноускоренное движение, повороты, набор и снижение высоты, пикирование объекта и т.д. Входом для модуля расчета служит набор параметров, выходом является координаты X,Y,Z в определенный момент времени t.

Q.2. Какие виды движений необходимо реализовать в рамках данного прототипа?

A.2. Необходимо реализовать процедуру расчета равномерного прямолинейного движения и процедуру расчета равноускоренного (равнозамедленного) движения.

Q.2.1. По каким формулам осуществляется расчет равномерного прямолинейного движения?

A.2.1. Входными данными являются:

Xн, Yн, Zн - координаты начальной точки траектории;

K – курс объекта;

V – скорость объекта;

h – высота объекта;

t - время движения на участке прямолинейного и равномерного движения.

Математическое описание:

Расчет траектории осуществляется по системе формул:

$X(t) = Xн + V \cdot \cos(K) \cdot t;$

$Y(t) = Yн + V \cdot \sin(K) \cdot t;$

$Z(t) = Zн$

где X(t), Y(t), Z(t) – координаты объекта на момент времени t,

Q.1.2. По каким формулам осуществляется расчет равномерного равноускоренного движения?

А.1.2. Входными данными являются:

Xн, Yн, Zн - координаты начальной точки участка равноускоренного (равнозамедленного) прямолинейного движения;

К – курс объекта;

Vн – начальная скорость объекта на участке равноускоренного (равнозамедленного) прямолинейного движения;

Vк – конечная скорость объекта на участке равноускоренного (равнозамедленного) прямолинейного движения;

a – ускорение объекта на участке равноускоренного (равнозамедленного) прямолинейного движения;

t -время движения на участке траектории.

Математическое описание.

Расчет траектории осуществляется по системе формул:

$X(t) = Xн + Vн \cdot \cos(К) \cdot t + a \cdot \cos(К) \cdot t^2 /2;$

$Y(t) = Yн + Vн \cdot \sin(К) \cdot t + a \cdot \sin(К) \cdot t^2 /2;$

$Z(t) = Zн$

где X(t), Y(t), Z(t) – координаты объекта на момент времени t.

Текущая скорость объекта вычисляется по формуле;

$V(t) = Vн + a \cdot t$

Время движения на участке разгона (торможения) вычисляется по формуле:

$\Delta t = (Vк - Vн)/a$

Процедура равноускоренного (равнозамедленного) прямолинейного движения вырабатывает местоположение объекта и его элементы движения в текущий момент времени t.

Зафиксируем соответствующие результаты в вопросно-ответном протоколе и в декларативном представлении постановки задачи. Расширять декларативное

представление удобно в графическом режиме, путем добавления соответствующих вершин и связей на диаграмму (используя палитру PredicateShapes), а затем осуществить её перевод в вопросно-ответный протокол, как показано на рисунке ниже. Надо отметить, что для удобства навигации при выделении единицы в QA-протоколе будет выделен соответствующий подграф в графической проекции декларативного представления. В случае если диаграмма получается слишком большой возможно ее разбиение на несколько маленьких с сохранением каждой в виде QA-единицы.

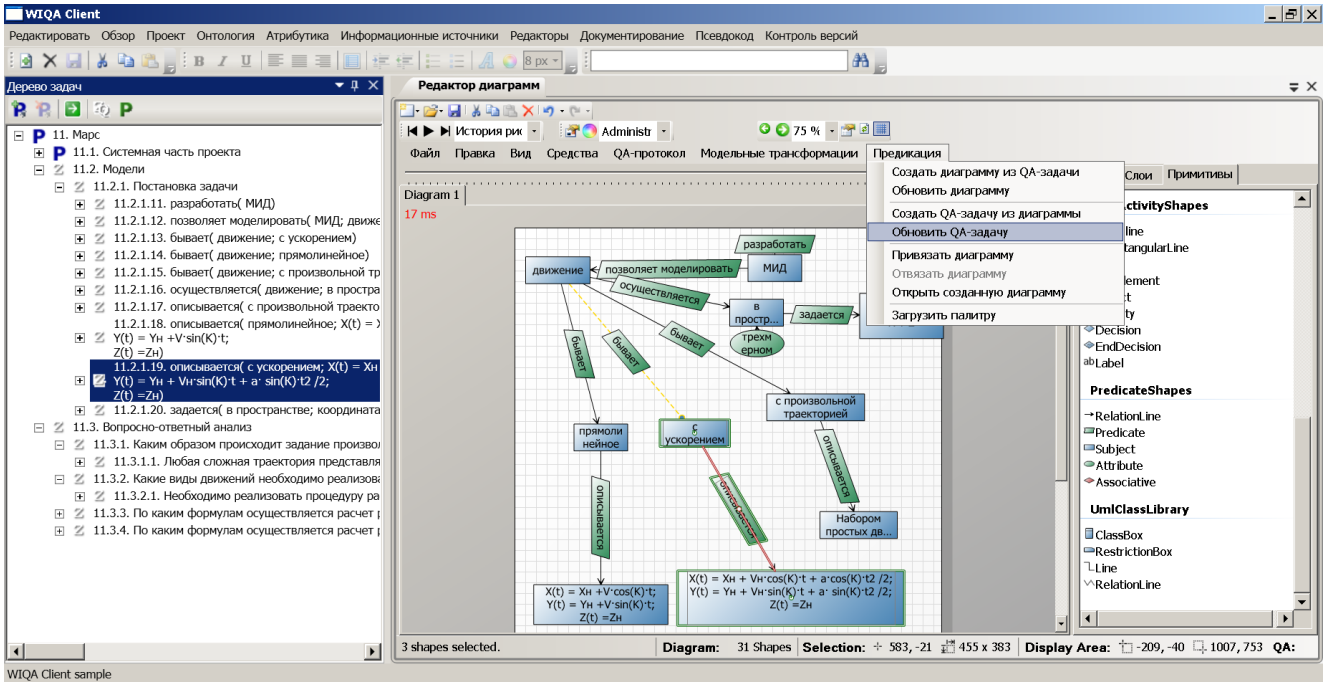

Рисунок 3. 26 - Фиксация QA-анализа в протоколе и в графической проекции декларативного представления с последующим переводом в вопросно-ответное дерево

Зафиксируем сделанные изменения в системе контроля версий, аналогично тому, как было сделано на предыдущем шаге.

Шаг 3. Разработка концептуально-алгоритмических моделей

К настоящему моменту, за счет произведенного анализа и уточнения постановки задачи, возможен переход к разработке прототипа, в виде создания концептуально-алгоритмических моделей (диаграммы активностей, диаграммы классов и диаграммы вариантов использования). Так как мы рассматриваем

только создание модуля имитации движений, с которым пользователь не будет взаимодействовать напрямую, то возможно ограничится только созданием диаграмм активностей и классов, причем, как и декларативное представлении, они могут создаваться либо через графическую или через программную (зафиксированную в QA-протоколе) проекции. Базовый алгоритм который будет использоваться при создание процедур движения представлен на рисунке ниже. Следует отметить что пункт «Инициализация параметров» может быть дополнительно описан, так как он должен включать также и проверку параметров на корректность, а также загрузку параметров из внешнего хранилища (СУБД)

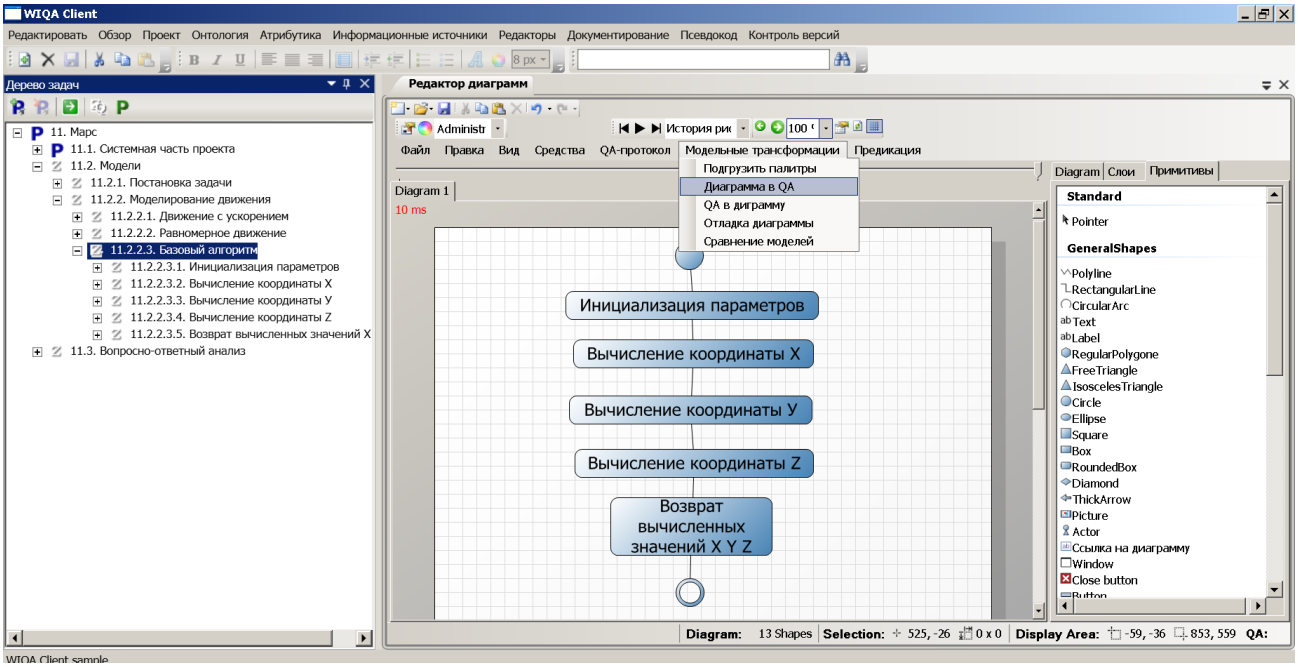

Рисунок 3. 27 - Базовый алгоритм в виде диаграммы активностей для процедуры вычисления координат

Сделаем конкретные реализации алгоритмов для рассмотренных ранее формул движения. Для удобства тестирование зададим основные параметры, а время будет получать через операцию ввода (INPUT).

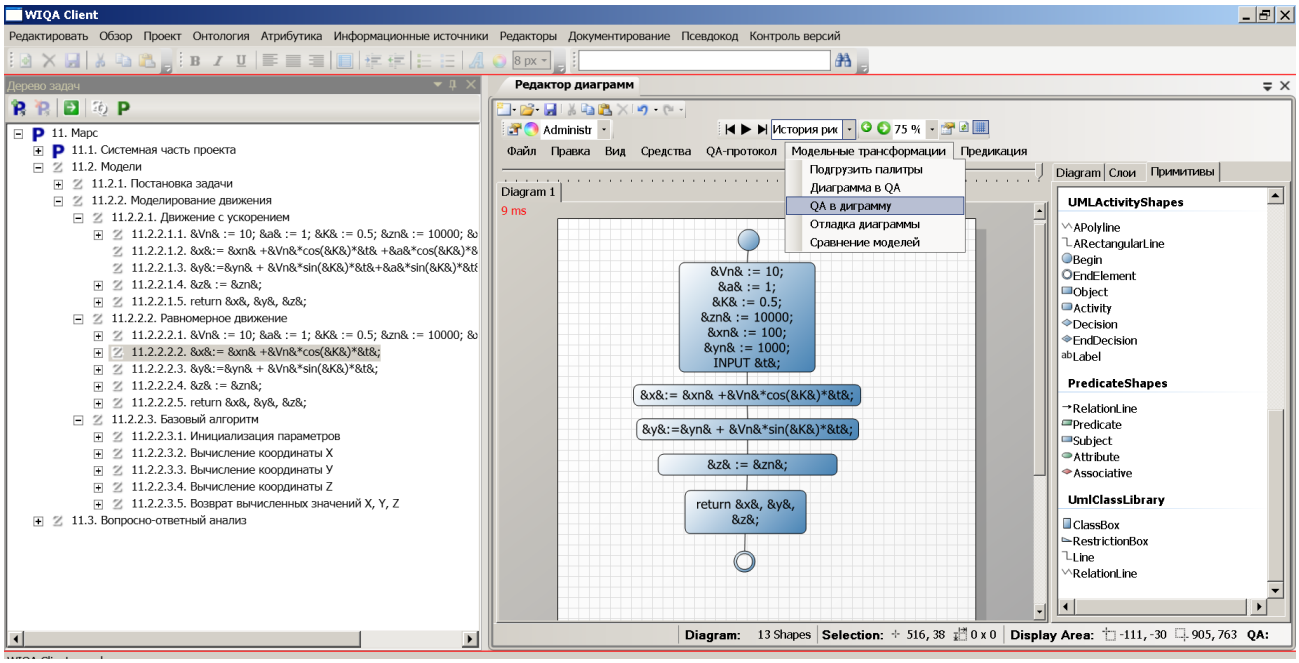

Рисунок 3. 28 - Пример реализации алгоритма на языке псевдокод с последующим построение графической проекции.

Запустим псевдокодовый интерпретатор и начнем отлаживать алгоритм путем пошагового исполнения.

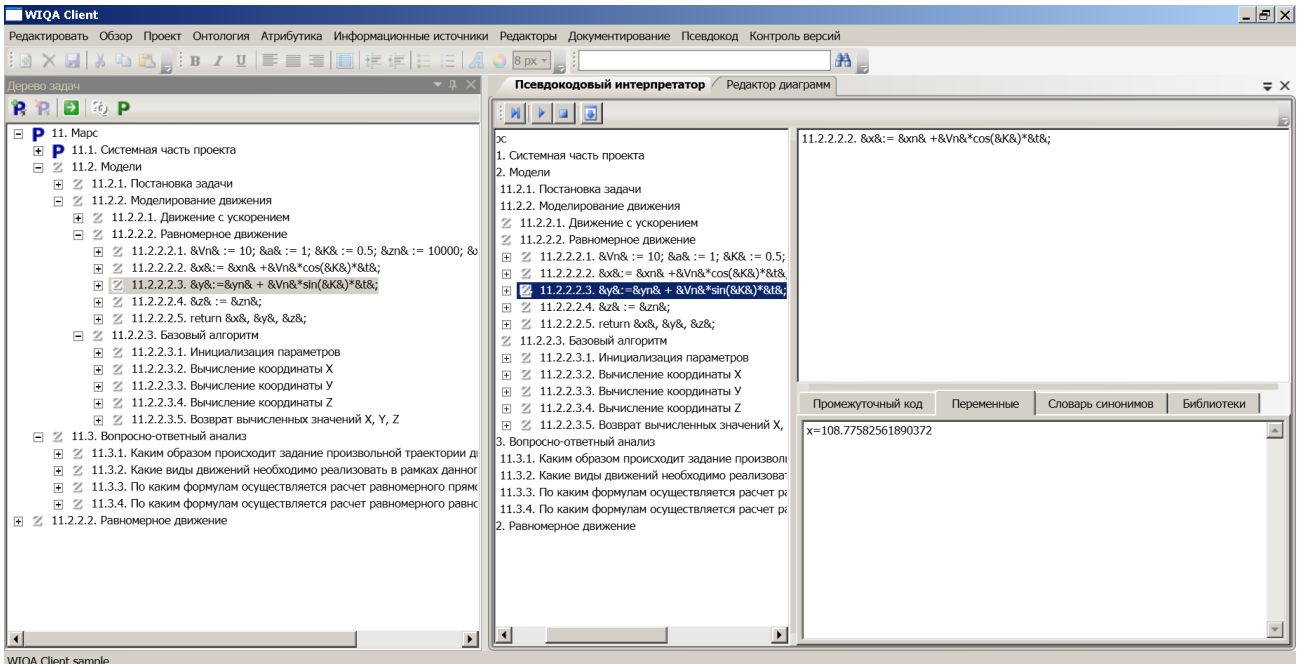

Рисунок 3. 29 - Пошаговое исполнение алгоритма в псевдокодовом интерпретаторе.

Сейчас метод возвращает координаты в виде массива, но удобней это делать в виде класса (структуры), которая во многих ЯВУ называется «Point3d» содержит три координаты и дополнительные функции для работы с ними

(приближенное сравнение, масштабирование, вычисление расстояния). Представим это структуру в виде простой диаграммы классов как показано на рисунке:

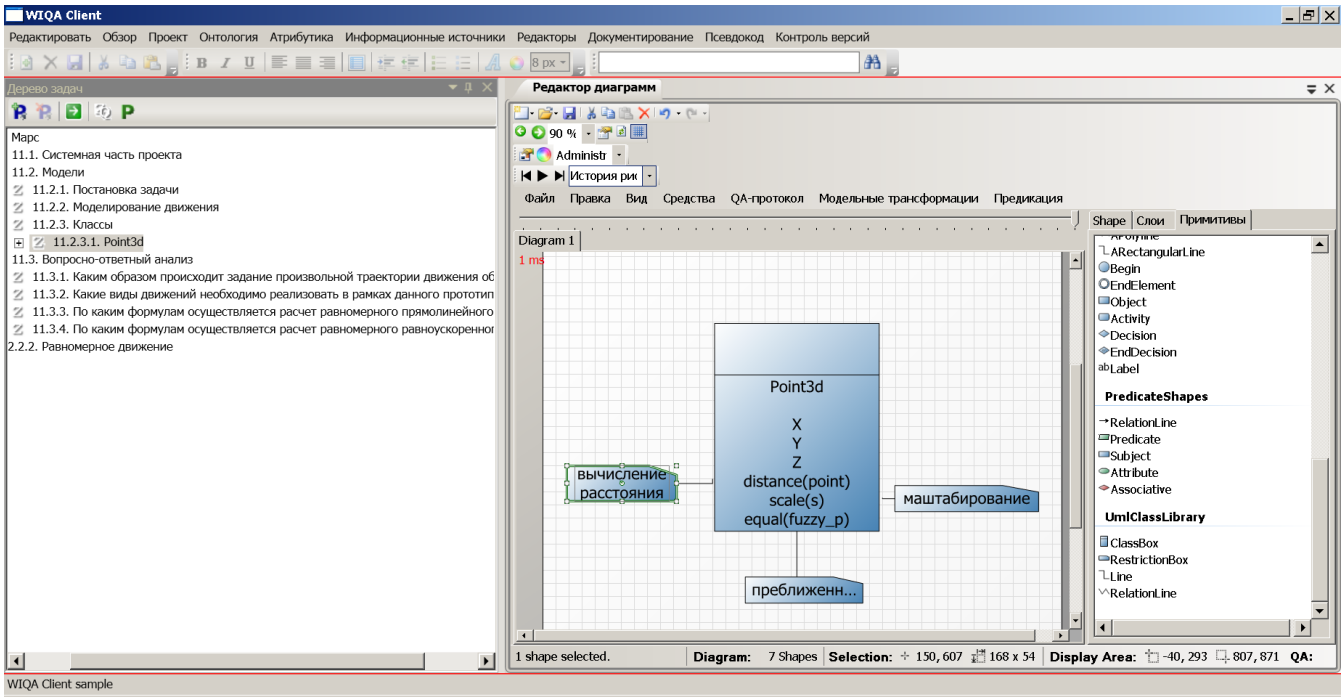

Рисунок 3. 30 - Диаграмма классов для структуры Point3d.

После отладки алгоритма зафиксируем изменения в репозитории:

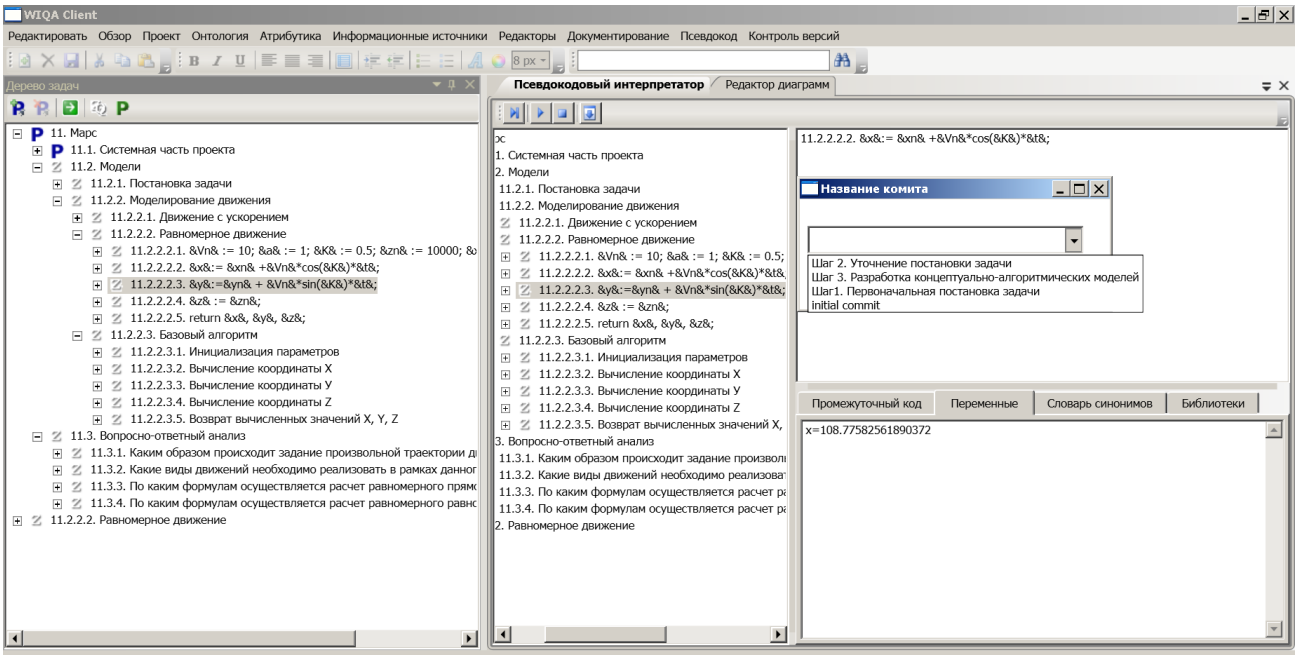

Рисунок 3. 31 - Фиксирование результатов 3 шага в системе контроля версий.

Шаг 4. Динамическая визуализация

Ранее были подробно рассмотрены работа с декларативным и концептаульно-алгоритмическим представлением, интерпретатором псевдокода и пролог, наполнение словаря понятий и возможность обращение к нему из интерактивной консоли не рассмотренным осталось формирование изобразительного представление, которое ориентировано на динамическую визуализацию и позволяет создавать произвольные диаграммы на основе доступных палитр. Как отмечалось в 3 главе в изобразительное представление можно перевести любые диаграммы путем выбора соответствующего пункта меню:

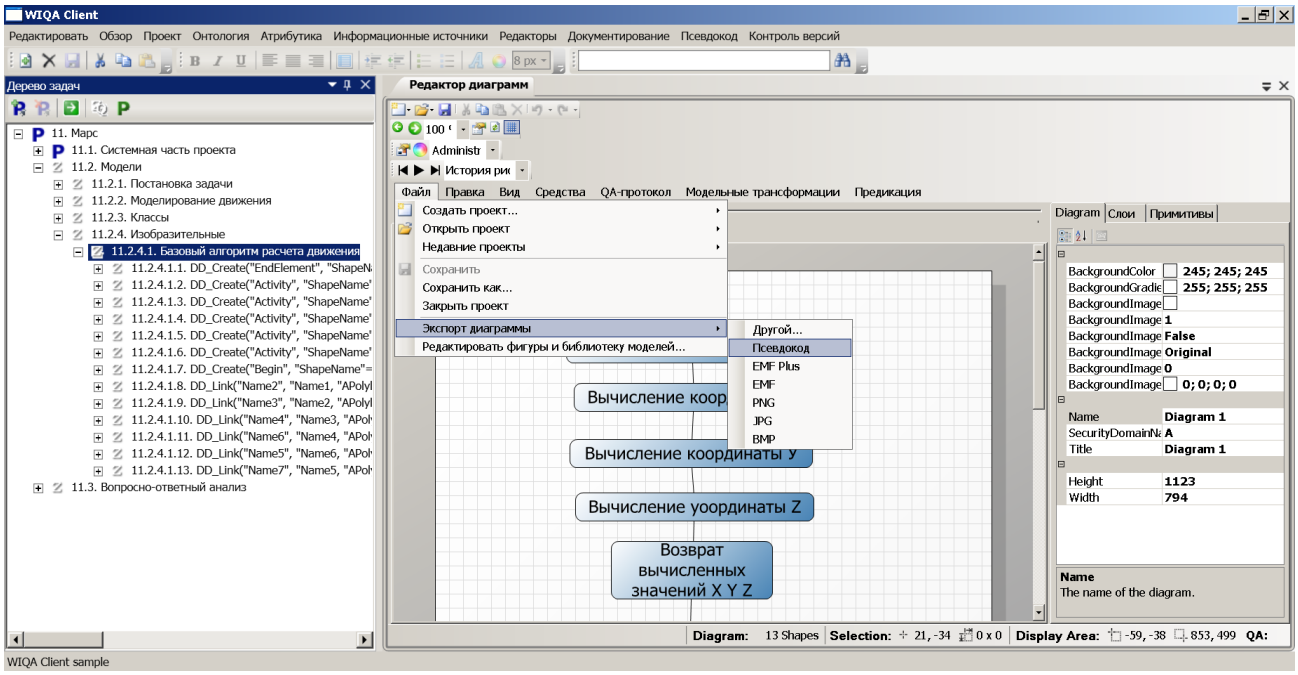

Рисунок 3. 32 - Перевод диаграммы активностей в программную проекцию изобразительного представления.

Так как программная проекция изобразительного представления является программой рисования на языке LWIQA (набором методов создания, удаления, изменения примитива и т.д.) на основе нее можно осуществлять различные эксперименты связанные с динамической визуализации и имитацией движения. Для рассмотрения этих возможностей применим функцию прямолинейного равноускоренного движения, которая была разработана ранее для визуализации процесса движения примитива. Так как визуализация происходит в двухмерном

пространстве графического редактора, то для простоты зафиксируем координату Z и в качестве единиц измерения выберем пиксель (скорость будет задаваться в пиксель/с, а ускорение в пиксель/$с^2$ соответственно). Выберем следующие начальные параметры:

$X_0 = 100$

$Y_0 = 100$

a = 10

$V_0 = 10$

K = 0,52 (примерно 30 градусов)

Количество точек 8, с дискретизацией 1 секунда

Представить движение можно двумя способами – через обновление координат у созданной вершины и через создание вершины с новыми координатами. Функция обновления и создания имеют вид:

```
DD_Update("Point",  "X"=&x&, "Y"=&y&)
DD_Create("Begin", "ShapeName"="Name"+&i&, "Diameter"="40", "Text"="", "CharacterStyle"="Normal", "ParagraphStyle"="Title", "X"=&x&, "Y"=&y&, "Angle"="0", "FillStyle"="Blue", "LineStyle"="Normal")
```

Мы выбрали 2 способ, который будет рисовать каждый новый элемент заново. Выполнение алгоритма представлено на рисунке ниже:

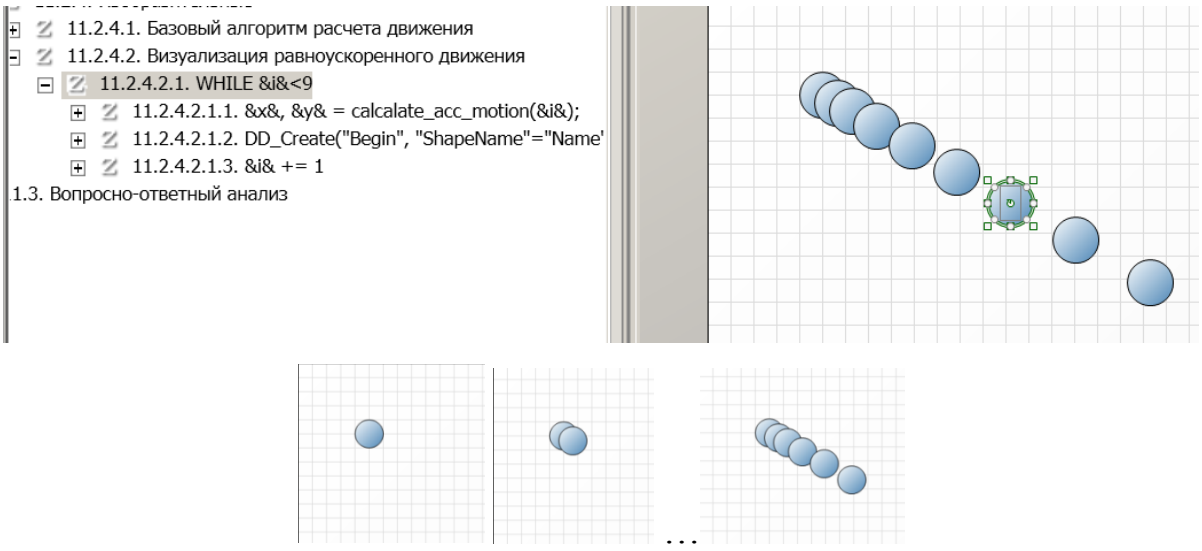

Рисунок 3. 33 - Пример динамической визуализации равноускоренного движения.

Использование динамической визуализации приносит множество преимуществ. В данном случае она позволила наглядно увидеть работу прототипа (алгоритма прямолинейного равноускоренного движения), тем самым показав его корректность. Если бы формула была не верна, то при динамической визуализации объект мог отклониться от прямолинейного пути (или не ускорятся) что сразу было видно при визуализации.

Таким образом, задача создания прототипа модуля движения была успешно решена, получен и отлажен прототип, началось наполнение базы понятий, которое будет происходить далее в процессе решение других модулей имитации движения.

Пример решенной задачи в рамках внедрения в ООО «ФБ-Групп»

Процесс решения проектной задачи уместно рассмотреть по шагам, начиная от этапа, когда задача представляется набором ключевых слов и постановка задачи еще не сформулирована и заканчивая этапами прототипирования пользовательского интерфейса и алгоритмов работы.

Шаг 1. Формулировка постановки задачи

Рассмотрим процесс решения этапа с самого начала, когда постановка еще не написана и задача представляется набором ключевых слов, предложений. Для формирования общего представления о структуре будущего программного комплекса удобно начать с формирования изобразительного представления, показанного на рисунке:

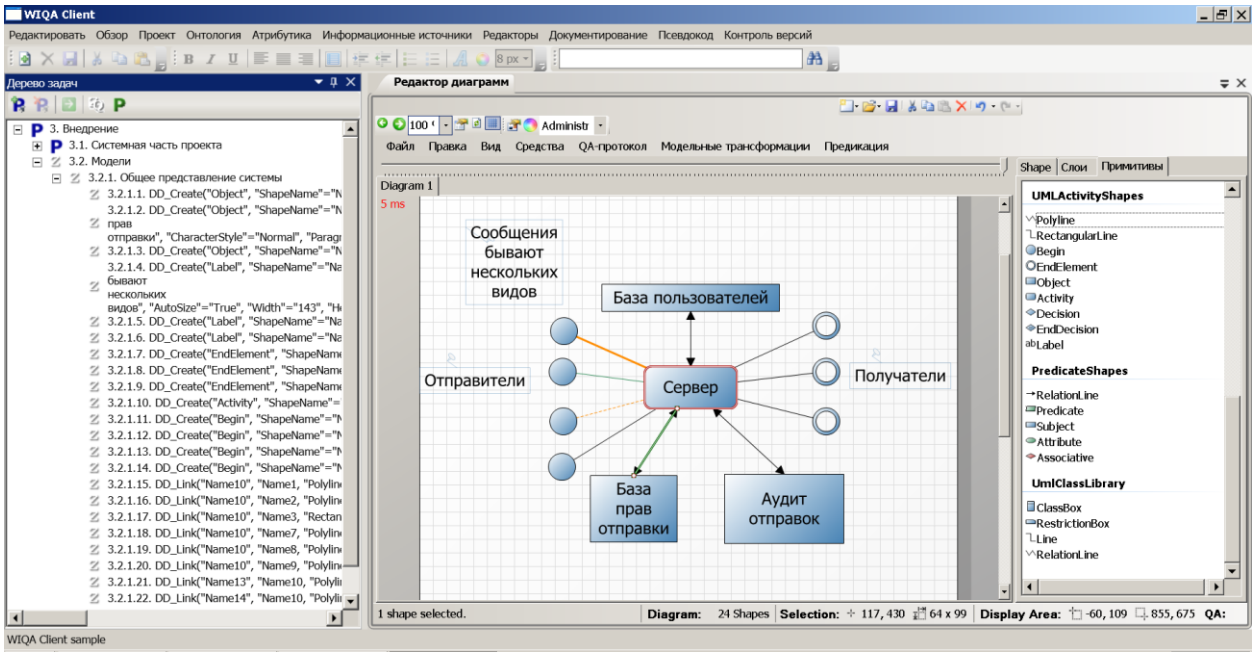

Рисунок 3. 34 - Построение изобразительной модели

Обобщенная постановка задачи может быть описано следующим образом. Необходимо разработать клиент-серверное приложение, позволяющее пересылать различные виды структурированных сообщений. При отправке необходимо учитывать права доступа к системе обмена сообщениями, а также права на возможность отправки абонентом «А» сообщения абоненту «Б». На стороне сервера выполняется протоколирование (аудит) информации об активности пользователей. Такую постановку можно представить в следующей предикативной форме:

<Предикат> (<Субъект> ; <Объект>) <Ассоциативная составляющая>

Для предиката, субъекта и объекта через запятую можно указать свойства – слова (словосочетание) отвечающие на вопрос «Какое?», «Какой?» и т.д.:

<имя субъекта>, <свойство1, свойство2, …>.

Представление задачи в виде текстовой проекции декларативного представления имеет вид:

```
разработать (приложение, клиент-серверное)
позволяет отправить (приложение; сообщение)
имеет (сообщение; структура)
бывает (сообщение; нескольких видов)
контролирует (приложение; права доступа)
проверяет (приложение; права на возможность отправки)
```

выполняет (приложение; аудит)

Для удобства восприятия текстовое (предикативное) представление может быть переведено в графическую проекцию:

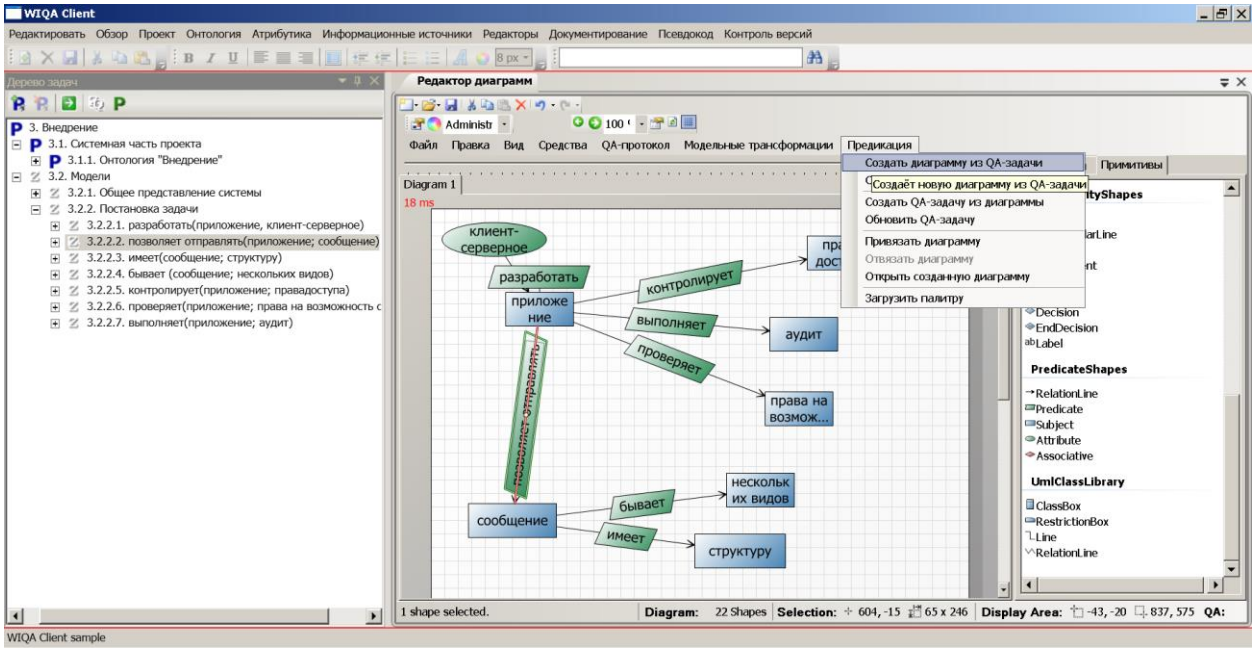

Рисунок 3. 35 - Формирование графической проекции декларативного представления

Концептуально-алгоритмическую модель на первом этапе рекомендуется строить в виде описания входа обработки и выхода в виде диаграммы активностей, которая представлена на рисунке в 2 проекциях (текстовое описание было автоматически сгенерировано из графической модели)

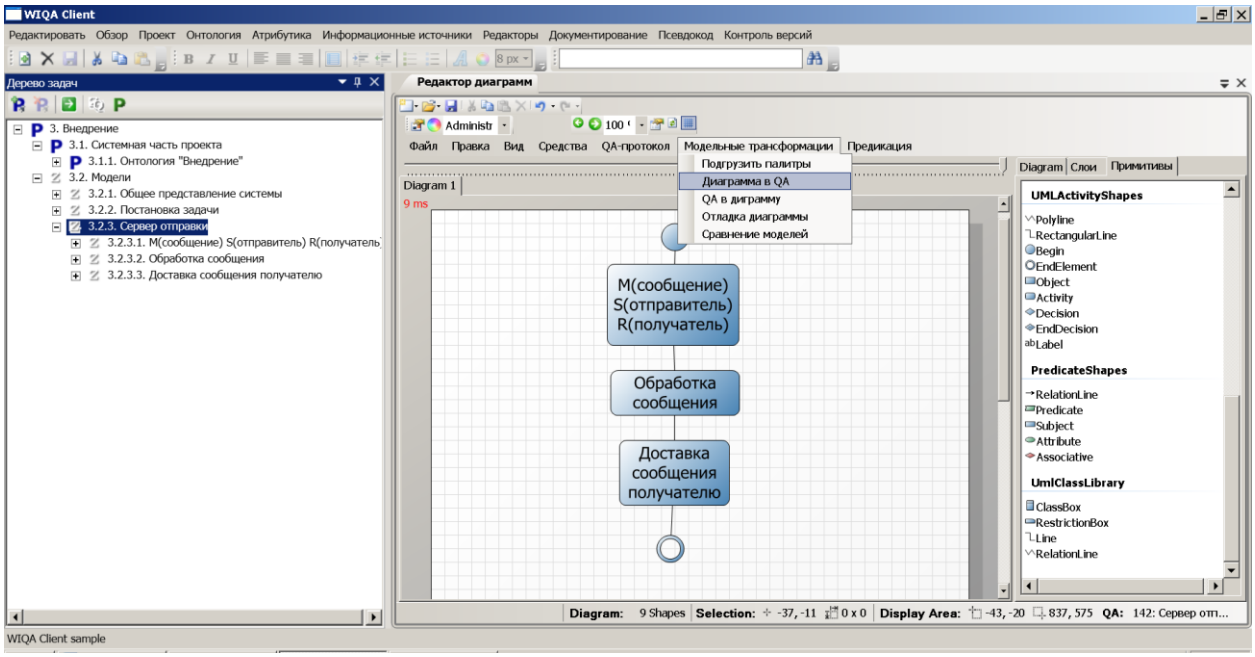

Рисунок 3. 36 - Первая версия диаграммы активностей, состоящая из блоков входа, обработки и выхода

Использование вопросно-ответного протокола как основу для фиксации стадий решения дает дополнительные возможности, такие как формирования вопросов, т.е. проведение вопросно-ответного анализа. На первом шаге были заданы следующие вопросы:

Какие бывают типы сообщений? Какую структуру имеет сообщение, как она зависит от типа? Как представляются права возможности отправки и в каком виде они задаются?

На этом этапе начинает формироваться словарь онтологий в виде определения новых терминов предметной области. На этом шаге было добавлено определение термина «клиент-сервер».

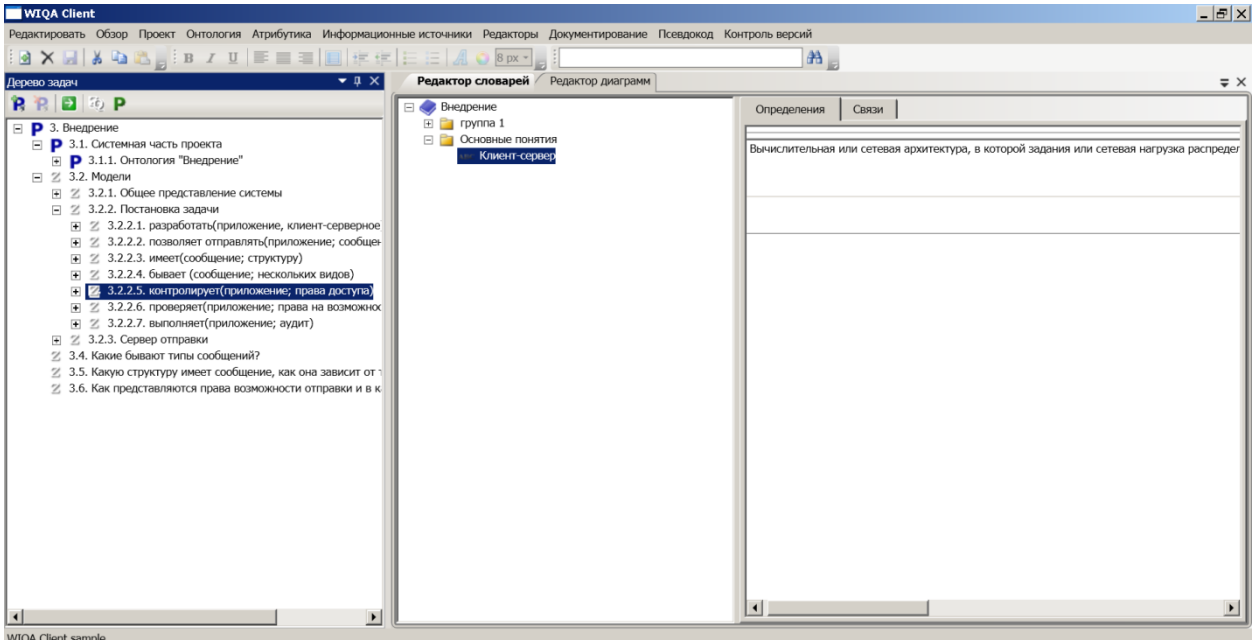

Рисунок 3. 37 - Формирование словаря онтологий

После завершение шага все наработки необходимо зафиксировать в среде контроля версий. Для этого необходимо создать новый репозиторий (рекомендуется создавать репозиторий для каждой задачи) и зафиксировать изменения.

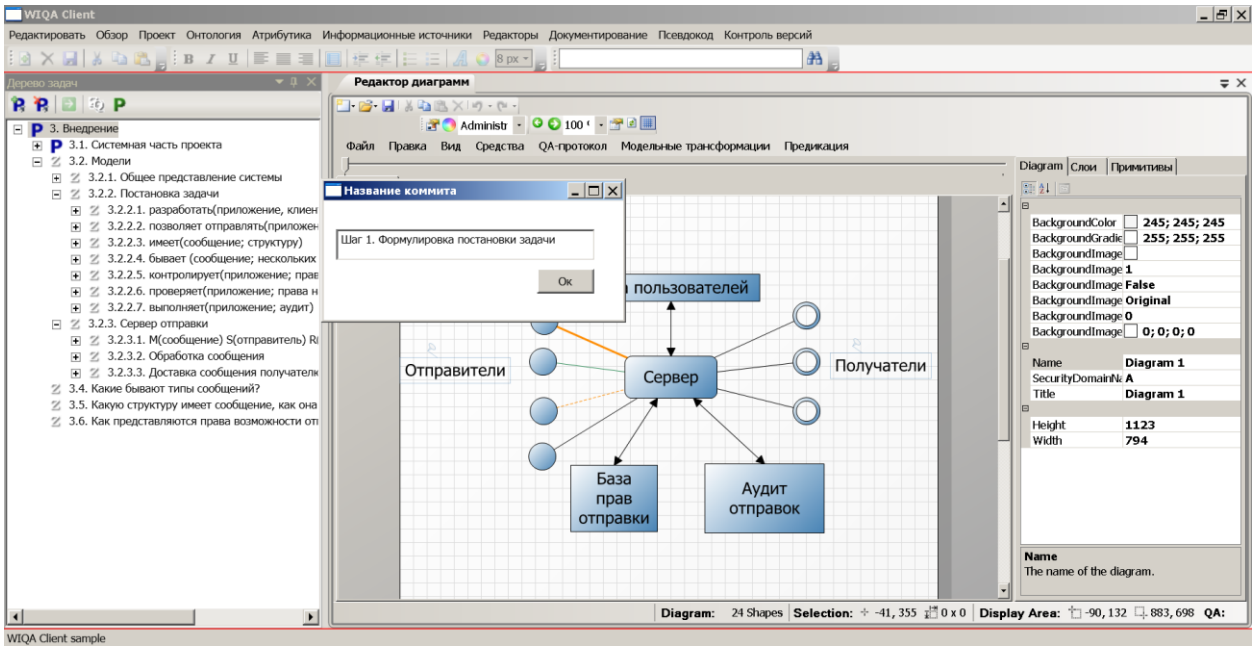

Рисунок 3. 38 - Фиксация изменений в средстве контроля версий

Шаг 2. Разработка концептуально-алгоритмического представления

Шаг логично начать с ответов на вопросы, которые были сформулированы на предыдущем шаге. Ответом на вопросы будут:

Q 1. Какие бывают типы сообщений?
A 1. Сообщение бывают 2 типов, в зависимости от его содержимого: текстовые сообщения и сообщения содержащие медиа-контент. В зависимости от типа сообщения они по-разному обрабатываются в подсистеме аудита и имеют различную структуру. Необходимо предусмотреть возможность расширения поддерживаемых типов.

Q 2. Какую структуру имеет сообщение, как она зависит от типа?
A 2. Структура сообщения зависит от типа, но имеет общие для всех типов поля: отправитель, получатель, важность сообщения (приоритет), контент и тип.

Q 3. Как представляются права возможности отправки и в каком виде они задаются?
A 3. Права отправки основаны на организационной структуре, которая имеет древовидную структуру. Сотрудник может отправлять сообщения либо своему непосредственному руководителю, либо сотрудникам того же отдела (имеют того же руководителя), либо своим подчиненным и сотрудникам более низшего уровня. Данную схему удобно представить в виде изобразительного рисунка, который представлен ниже.

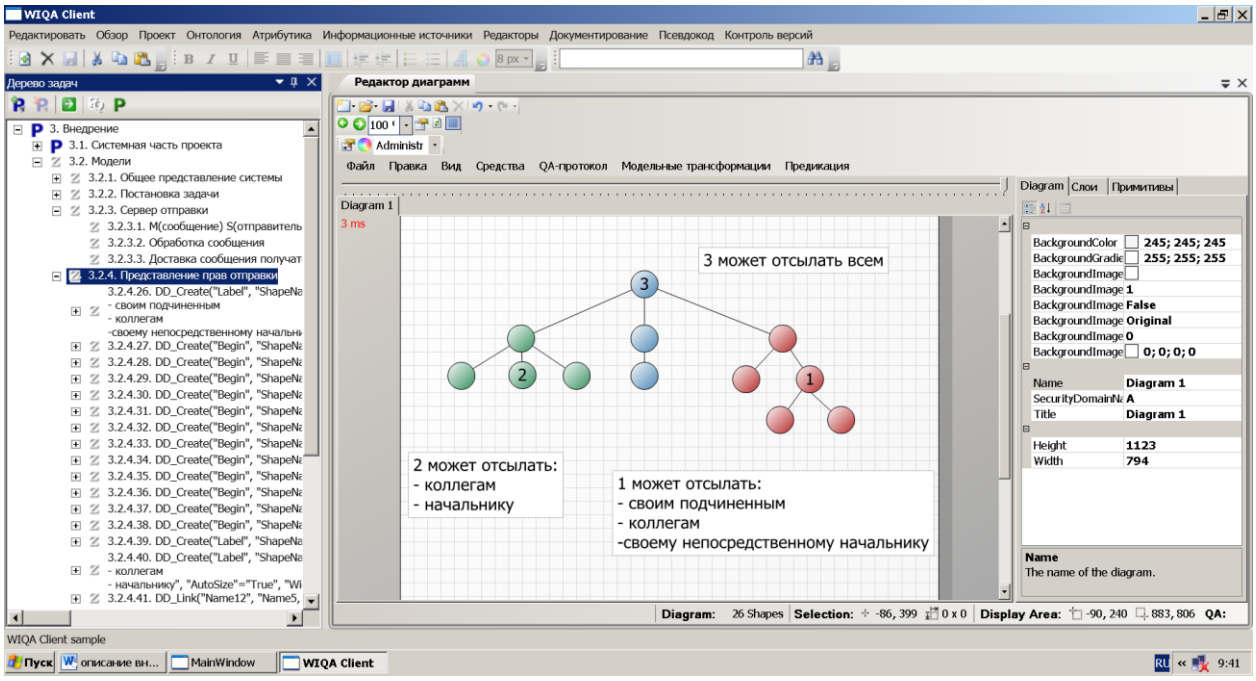

Рисунок 3. 39 - Представление оргструктуры в виде дерева

После перевода в текстовую проекцию изобразительного представления можно запустить процесс рисования и увидеть процесс в динамике, как показано на рисунке ниже.

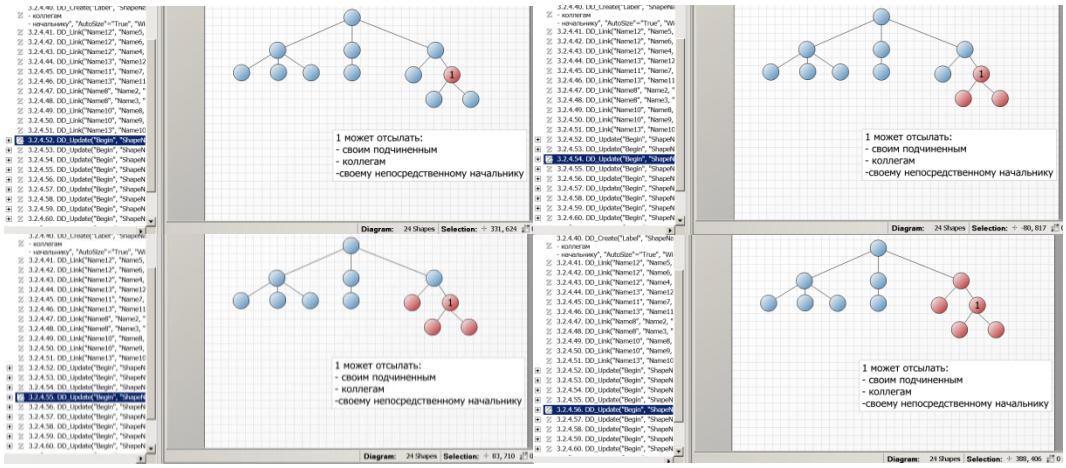

Рисунок 3. 40 - Процесс динамической визуализации

Уточнения задачи можно начинать с расширение графической проекции декларативной модели, путем фиксирования ответов на представленные вопросы и уточнения постановки задачи. Уточнение удобно производить с графической проекцией, а затем переводить полученный рисунок в текстовую проекцию декларативного представления.

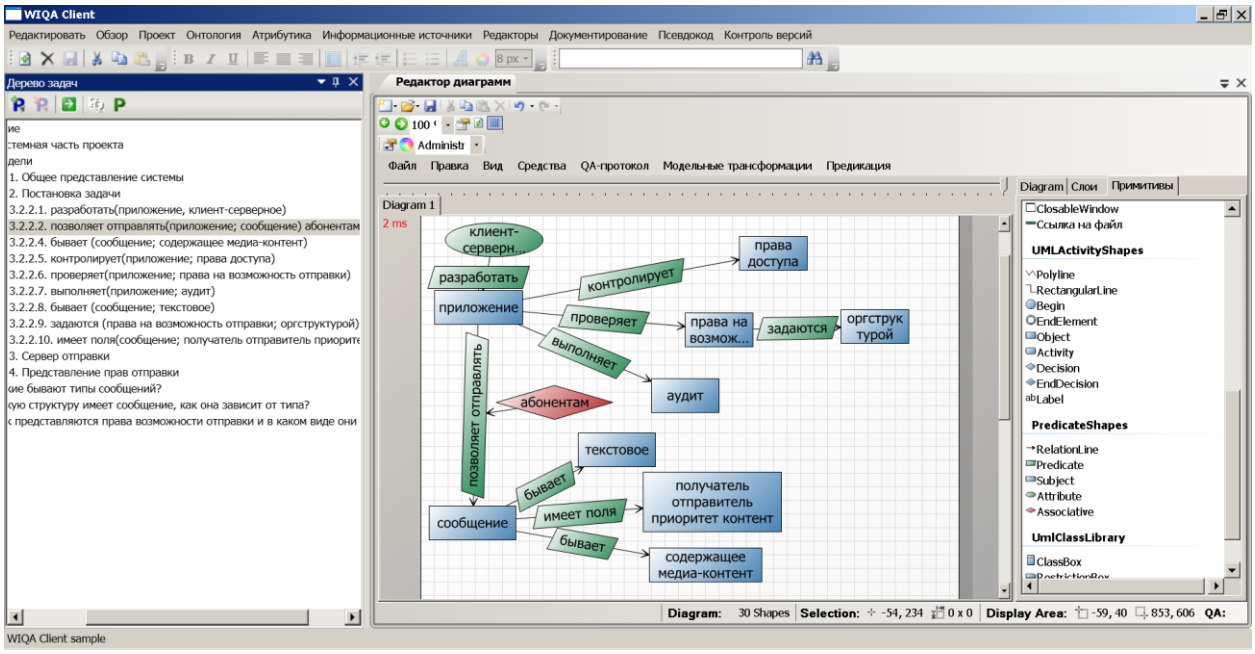

Рисунок 3. 41 - Уточнение декларативного представления

Постановка задачи на этом этапе имеет вид: «Разработать клиент-серверное приложение позволяющее отправлять сообщения абонентам. Приложение контролирует права доступа, проверяет права на возможность отправки, которые задаются на основе оргструктуры и выполняет аудит отправленных сообщений. Сообщения имеют поля отправитель, получатель, приоритет, контент и тип». Уточненная диаграмма активностей и диаграмма классов имеют вид:

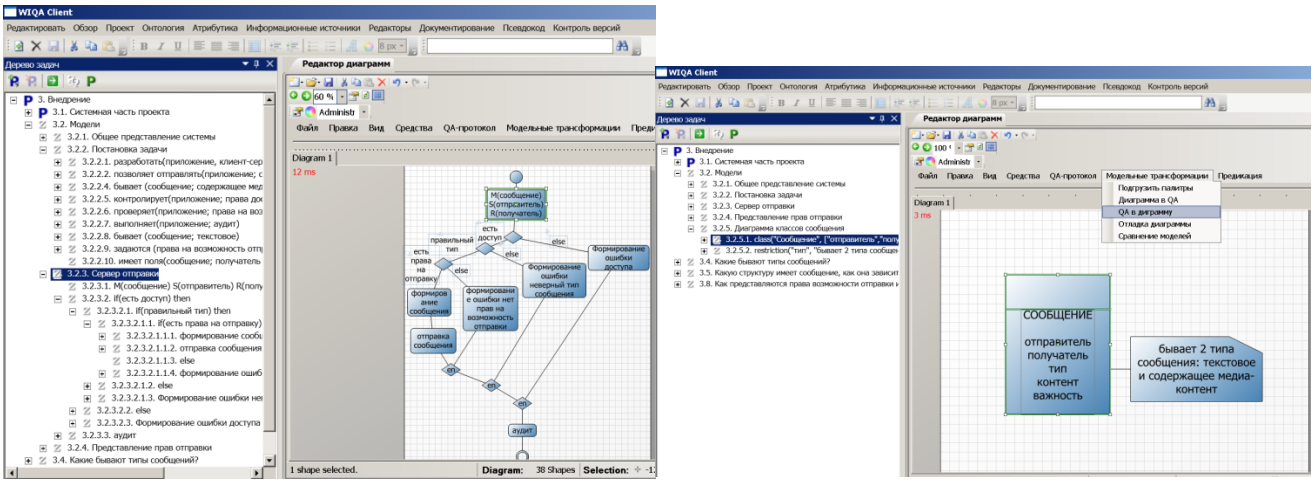

Рисунок 3. 42 - Уточнение концептуально-алгоритмического представления и создание диаграммы классов

Первоначально диаграмма содержала ошибки (неверно было соединена вершина завершения условия), поэтому на этапе перевода в текстовую проекцию концептуально-алгоритмического решения произошла ошибка, связанная с

отсутствием данного шаблона (LHS) в базе правил. Ошибка представлена на рисунке, цветом были выделены связи, возможно содержащие ошибку.

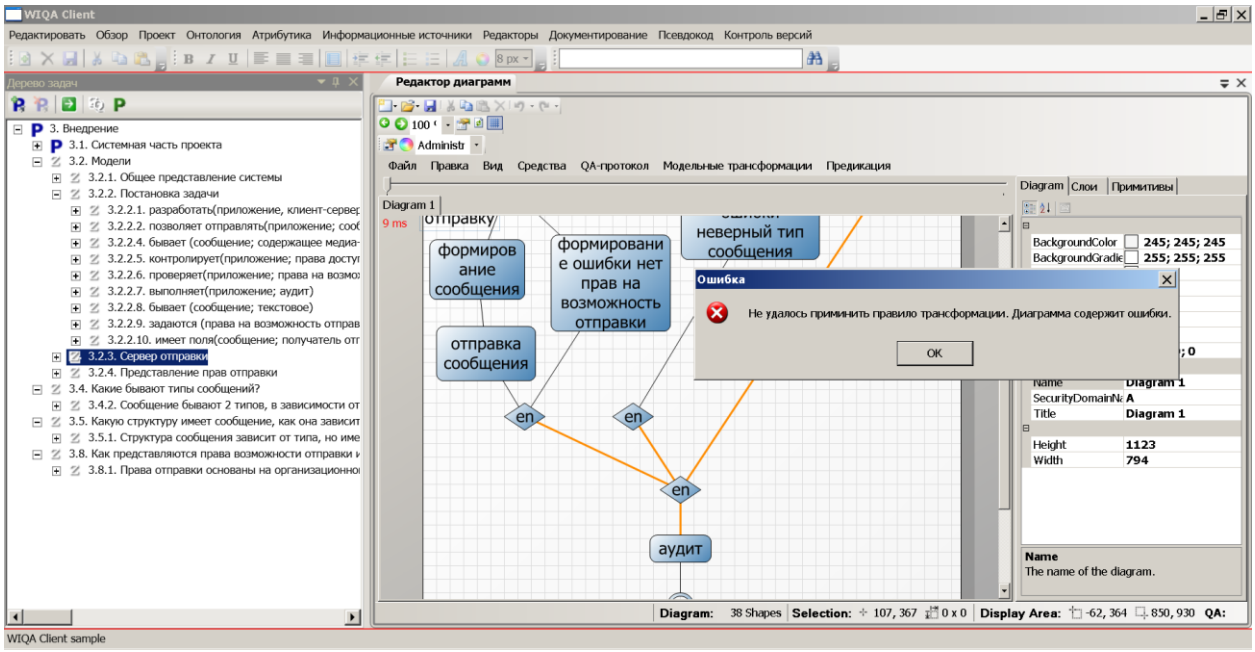

Рисунок 3. 43 - Вывод сообщения об ошибки при выполнении трансформации в вопросно-ответное представление

В процессе выполнения данного шага детализации возникли следующие вопросы, которые будут представлены в вопросно-ответном протоколе:

Q 1. Какой формат имеет аудит лог, создаваемый при отправке сообщений?

Q 2. Каким образом будут представлены клиентские приложения, реализующие данную функциональность?

Q 3. Какой протокол планируется использовать для взаимодействия между клиентом и сервером?

Q 4. В каком формате будет храниться оргструктура и права доступа?

После сохранения вопросов в вопросно-ответном протоколе фиксируем состояние задачи с помощью интегрированного средства контроля версий.

Шаг 3. Детализация

Выполнение шага логично начать с ответа на вопросы, представленные ранее:

А 1. Аудит лог имеет структуру текстового файла в формате CSV. И хранит следующий набор полей: дата, отправитель, получатель, приоритет, тип.

А 2. Клиентское приложение будет разработано на языке C# и будет иметь следующий набор элементов: текстовое поле для набора сообщение с возможностью загрузки медиа-контента, список выбора, содержащий возможные уровни приоритета, список выбора получателей, кнопка отправки.

А 3. В качестве протокола обмена сообщений был выбран протокол https включающий механизмы шифрования передаваемых сообщений.

А 4. Оргструктура будет представляться в СУБД в 2 таблицах. Одна будет хранить список сотрудников организации и их идентификаторы, другая будет хранить правила подчинения в виде идентификатора руководителя и идентификатора подчиненного. В СУБД будут определены ряд процедур, возвращающих следующие списки: список подчиненных данного работника, список коллег, список начальников (в случае если сотрудник принадлежит 2 отделам).

После фиксации ответов был уточнен алгоритм работы сервера; были использованы псевдокодовые функции INPUT(&переменная&) и PRINT(&переменная&), которые вызывают окно ввода значений и показ значения переменной. После этого текстовое представление было переведено в графическую проекцию концептуально-алгоритмического представления. Пошаговое выполнение данного кода с визуализации переходов на диаграмме представлено на рисунке:

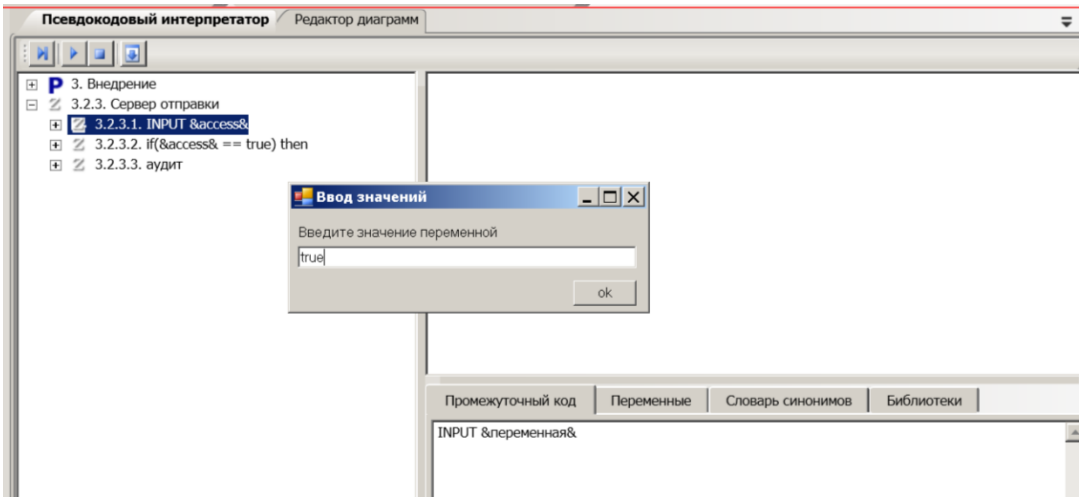

Рисунок 3. 44 - Отладка концептуальной программы (полученной из концептуально-алгоритмической диаграммы) в псевдо кодовом интерпретаторе

При выполнении функции INPUT в псевдокодовом интерпретаторе происходит вывод формы с возможностью задания значений. В зависимости от введенных значение активируются различные ветки условий, что отражается на диаграмме:

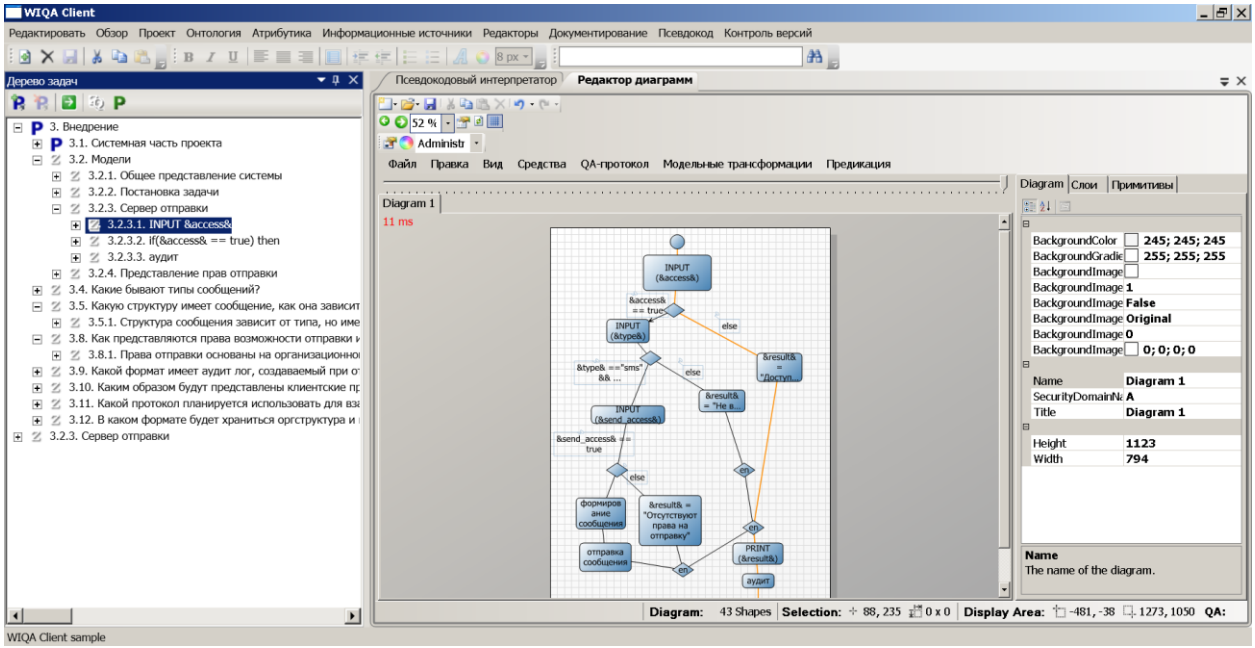

Рисунок 3. 45 - Представление пути на диаграмме

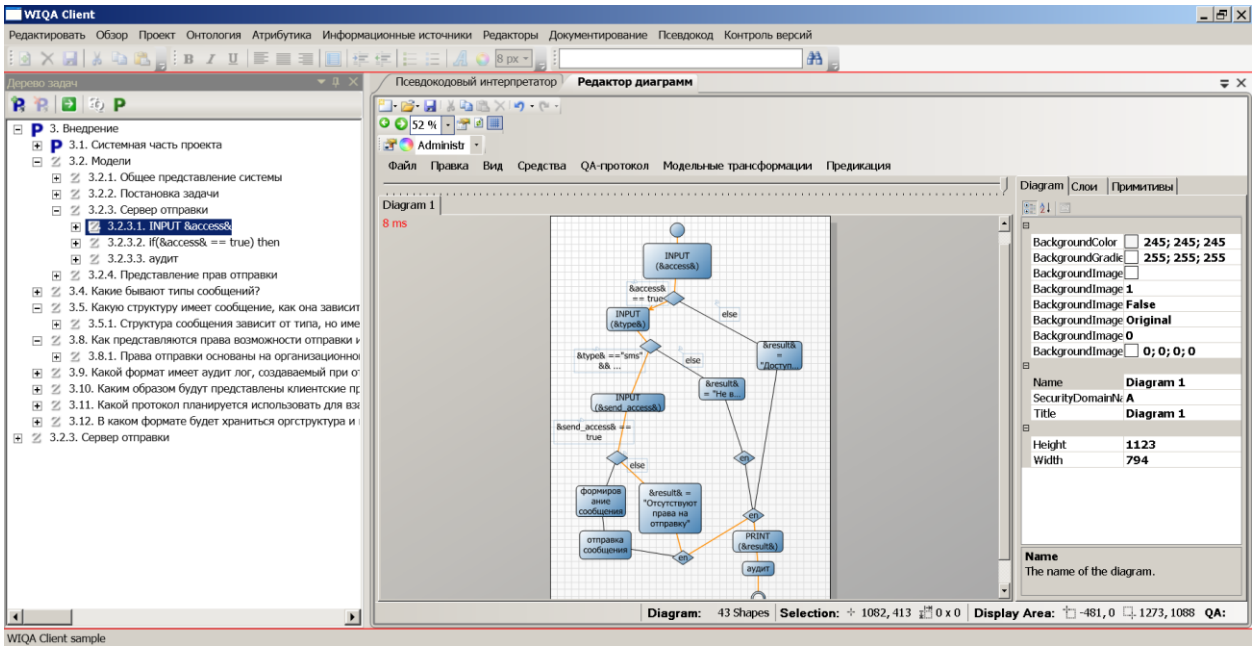

Рисунок 3. 46 - Изменение пути на диаграмме при изменении условий

Такая пошаговая детализация дает большие возможности при отладке концептуальных алгоритмов, так как позволяет производить экспериментирования и постепенно уточнять код путем ввода новых псевдокодовых функций. Первичное предположение о том, что сначала необходимо проверять тип сообщение на соответствие доступным форматам оказалось неправильным, так как проверка прав доступа является должна идти в самом начале.

В редакторе существует палитра для проектирование пользовательского интерфейса, таким образом, ответ на вопрос А 2. Может быть представлен в изобразительной моделью (и сгенерированной программой рисования).

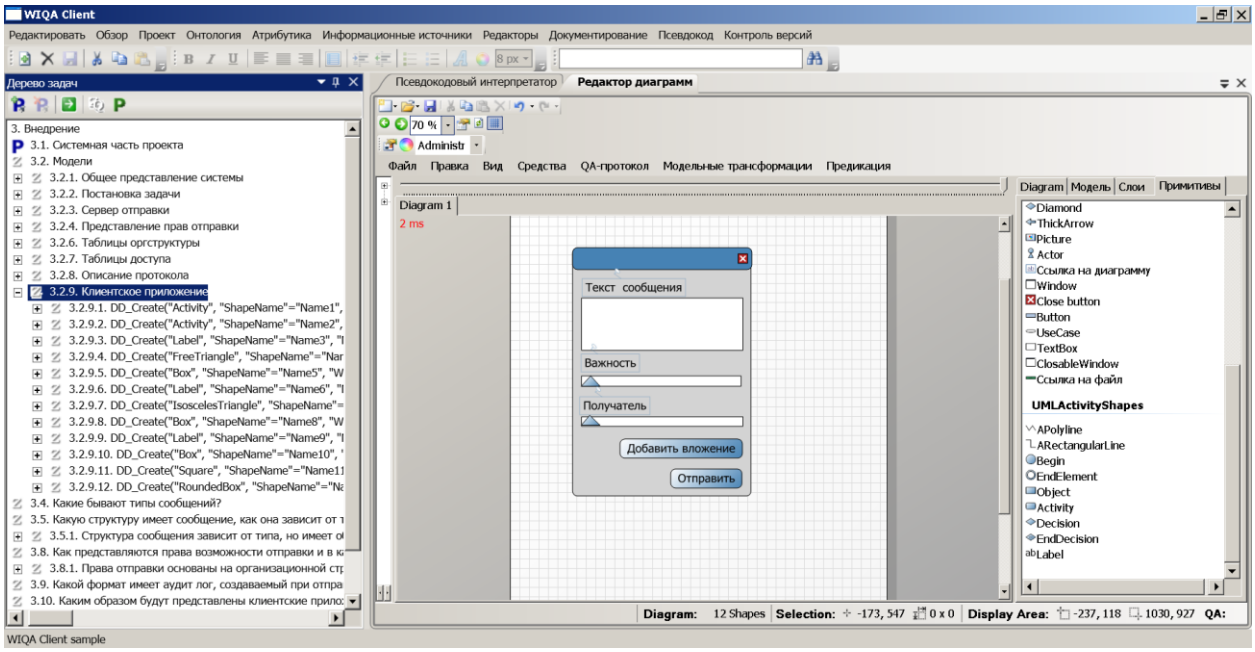

Рисунок 3. 47 - Прототипирование пользовательского интерфейса

Предложенные методы и средства были опробованы на практике в виде решения задачи обмена сообщение на базе клиент-серверной архитектуры. Рассмотрение задачи началось с этапа, когда постановка еще не была до конца сформулирована и путем процесса пошагового уточнения и согласований декларативной модели с ее предикатной формы, было получено решение на данном уровне детализации и получен и отлажен прототип описывающий логику работы сервера обработки сообщений. Дальнейшая детализация будет производить «понижение» уровня моделей и, в конечном счете, к реализации данной схемы, причем первоначальный прототип может быть реализован уже на псевдокоде. Таким образом, предложенные методы и разработанные средства поддерживают все этапы решения проектных задач, начиная от зарождения и заканчивая этапами прототипирования и проектирования пользовательского интерфейса.

Пример решенной проектной задачи в рамках внедрения в Нижегородское отделение компании Intel

Предложенные модели и методы были внедрены в практику решении проектных задач в отделе численных методов, который занимается разработки новых версий библиотек Intel® Math Kernel Library и Intel® Data Analytics

Acceleration Library. Далее будет рассмотрена только задача исследования и полученные результаты без подробного описание, которое приводилось для двух предыдущих внедрений. Задача внедрения заключалось в проверке гипотезы о том, что использование SIMD на многоядерных архитектурах позволит существенно ускорить быстродействие генераторов случайных чисел, без представления. Необходимо было реализовать новый подход к генерации случайных чисел и проверить его работоспособность на нескольких генераторах (mcg59, mcg31, mrg32k3a) и для разных распределений (uniform, Gaussian icdf, Gaussian box muller).

Версия ГСЧ, которая была реализована на тот момент, состояла из 3 основных шагов:

1. Заполнения массива N случайными числами с равномерным распределением
2. Трансформация значений – применения определенных распределений сгенерированного массива (нормальное распределение, Гауса и т.д.)
3. Нормализация – применение параметров распределения (min/max, sigma …)

Таким образом, требовалось 3 прохода по памяти. Предполагалось провести оптимизацию путем разбиение массива на блоки равные длине доступного для данной архитектуры SIMD регистра (размерностью 128, 256 и 512 байт) и осуществление 3 предыдущих шагов с регистром с последующей записью значений в память. Использование таких регистров позволяет работать с несколькими значениями параллельно. На рисунке показан код, демонстрирующий отличие в API двух програмных реализациях

```
vdRngUniform(VSL_RNG_METHOD_UNIFORM_STD, stream, ARR_LEN, arr, -1.0, 1.0);
vdErfInv(ARR_LEN, arr, arr);
for(i = 0; i < ARR_LEN; i++) {
    arr[i] = arr[i] * sigma * sqrt(2.0) + e;
}
```

```
for(i = 0; i < ARR_LEN; i++) {
    t = _mm256_mcg59_uniform_pd(st, a1, b1);
    t = _mm256_erfinv_pd(t);
    t = _mm256_add_pd(_mm256_mul_pd(t, sig_sqrt2), a);
    _mm256_storeu_pd((double*)&arr1[i],  t);
}
```

Рисунок 3. 48 - Процесс согласование задачи с декларативным представлением

В результате были реализованы прототипы трех ГЧС (Mcg59,Mcg31m1, mrg32k3a) и 3 распределений (Uniform, Gaussian ICDF, Gaussian Box-Muller (только для mcg31m1) и осуществлено тестирование быстродействий предложенных реализаций.

В однопоточном режиме новый подход показал эффективность на векторах большой размерностью:

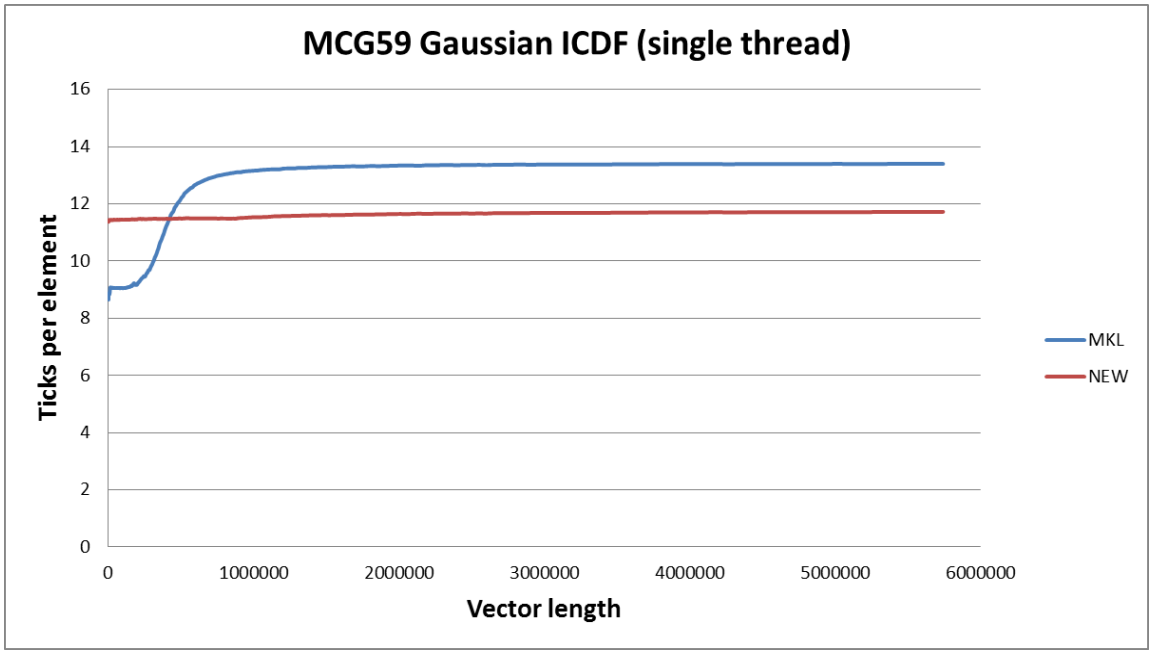

Рисунок 3. 49 - Сравнение быстродействий нового подхода с реализацией Intel MKL

Для многоядерных архитектур новый подход показал лучшие результаты:

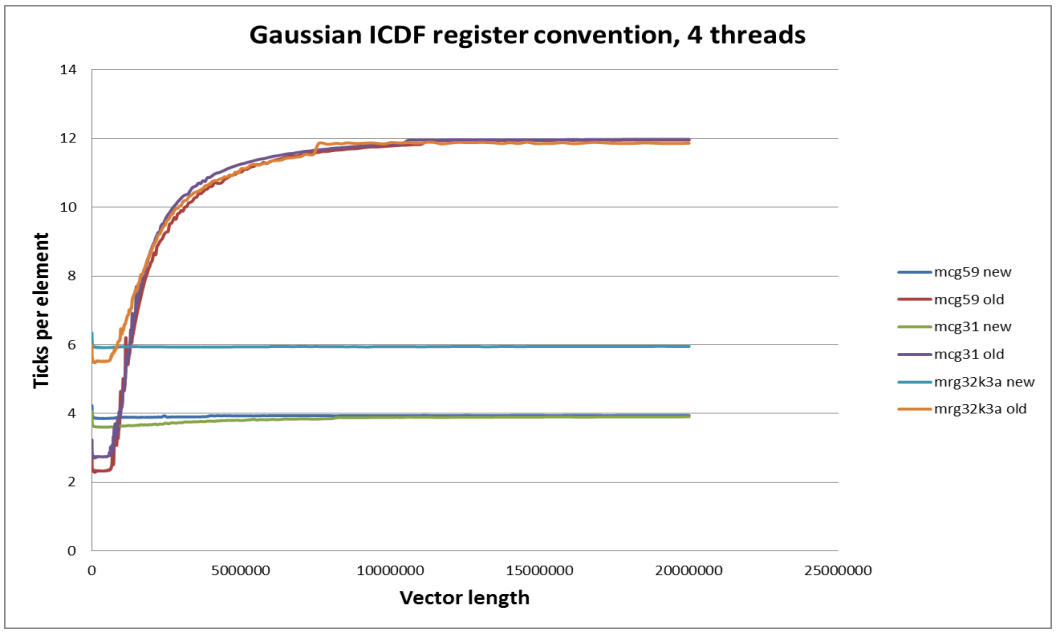

Рисунок 3. 50 - Сравнение быстродействий нового подхода с реализацией Intel MKL для многоядерных архитектур

По результатам анализа была построена таблица, представленная в таблице 3.1., показывающая выигрыш предложенных методов по сравнению с реализацией в Intel MKL.

Таблица 3.1 - Сравнение быстродействий методов для различных длин векторов

Длина вектора	**MKL**	**Новая реализация (register)**	**Новая реализация (array)**	**Выигрыш (register)**	**Выигрыш(array)**
20672	4.256	5.993	5.292	0.710	0.804
895744	5.219	5.939	5.242	0.879	0.996
1069888	5.963	5.941	5.248	1.004	1.136
1788384	8.193	5.941	5.239	1.379	1.564
20000000	11.912	5.952	5.284	2.001	2.255

По результатам анализа был сделан вывод что новый подход состоятелен и при генерации случайных чисел в 4 потока новый подход превосходит текущую реализацию примерно в 2 раза (для векторов больше 20 млн). Планируется внедрение предложенного подхода в новую версию компилятора ICC.

3.4. Выводы

1. Разработан метод итеративного согласования понятийного и образного содержания текстовых единиц с использованием их преобразования в прологоподобную форму (методика исполнения декларативного представления в интерпретаторе ПРОЛОГ, методика согласования понятийного и образного содержания, методика коррекции логических и фактических ошибок, методика взаимодействия со словарем онтологий).
2. Разработан метод понятийно-образной поддержки процесса пошаговой детализации в прецедентно-ориентированном решении проектных задач, позволяет включать автоматизированное концептуальное экспериментирования и моделирование в процесс решения (методика формирования изобразительного представления, методика создания концептуально-алгоритмического представления, методика, методика перевода в исполняемый псевдокод, методика согласования представлений, методика работы со средством контроля версий).
3. Разработанные модели методы и средства применены н практики для решения проектных задач в рамках диссертационного внедрения (задача разработки модуля имитации движения объектов для ИРЛС (ОАО НПО «Марс»), задача разработки клиент-серверного приложения пересылки структурированных сообщений (ООО «ФБ-Групп»), задача оптимизации генератора случайных чисел (Нижегородский филиал компании Intel))

Глава 4. Особенности реализации специализированного графического редактора.

В данной главе перейдем к вопросу о реализации экспериментально-практической части диссертационной работы. Центральное место в системе поддержки (рисунке 4.1) занимает специализированный редактор, поэтому логично начать описание реализации именно с него.

Рисунок 4. 1 - Общая структура разработанных компонентов

В качестве основы для разработки специализированного редактора был взят компонент с открытым исходным кодом NShape. Перед выбором основы для специализированного редактора было произведено сравнение нескольких графических компонентов (собственно разработанный графический редактор на WPF, Devexpress diagram editor, Diagram.NET, Open Diagram) и выбран NShape, по следующим причинам:

1. Открытость исходного кода.
2. Ориентация на крупные промышленные приложения.
3. Большой набор базовых функций по редактированию, представлению, манипуляцией с историей изменений.
4. Возможность работаты с большим числом элементов на диаграмме (приемлемое быстродействие при количестве элементов в несколько тысяч).

5. Поддержка шаблонов. Изменение шаблона позволяет быстро менять стили диаграмм.
6. Поддержка различных репозиториев (хранилищ) диаграмм.
7. Поддержка прав доступа к диаграммам.
8. Удобная интеграция в приложение.
9. Поддержка биндингов (изменение диаграммы при изменении модели данных).
10. Легкая расширяемость за счет поддержки механизма плагинов.

Так как специализированный редактор построен на базе свободного программного обеспечения NShape, то перед встраиванием компонентов в WIQA была проделана большая работа по изучению документации и общей архитектуры редактора. Фреймворк разбит на несколько больших модулей, описание которых приведено в таблице 4.1.

Таблица 4.1 - Описание основных пространств имен NShape

Название	Описание
Dataweb.NShape	Основное пространство имен, включающее классы, для работы с диаграммами и фигурами.
Dataweb.NShape.Advanced	Пространство имен включает классы для реализации дополнительных палитр, работы с формами фигур.
Dataweb.NShape.Commands	Классы действий над фигурами. Каждая команда может быть отменена.
Dataweb.NShape.Controllers	Пространство классов, отвечающих за представление.
Dataweb.NShape.Layouters	Интерфейсы и классы для автоматической расстановки фигур.
Dataweb.NShape.WinFormsUI	Классы для реализации пользовательских интерфейсов Windows Forms.
Dataweb.Utilities	Дополнительная функциональность, не вошедшая

	в предыдущие пространства классов.

Представление полной диаграммы классов затруднительно, так как редактор содержит большое количество классов и интерфейсов. Ниже (в таблице 4.2) будут рассмотрены только некоторые классы и интерфейсы, которые непосредственно использовались (дорабатывались) в процессе реализации специализированного редактора, а на рисунке 4.2 показаны базовые классы примитивов использующиеся для реализации палитр декларативного и концептуально-алгоритмического представлений.

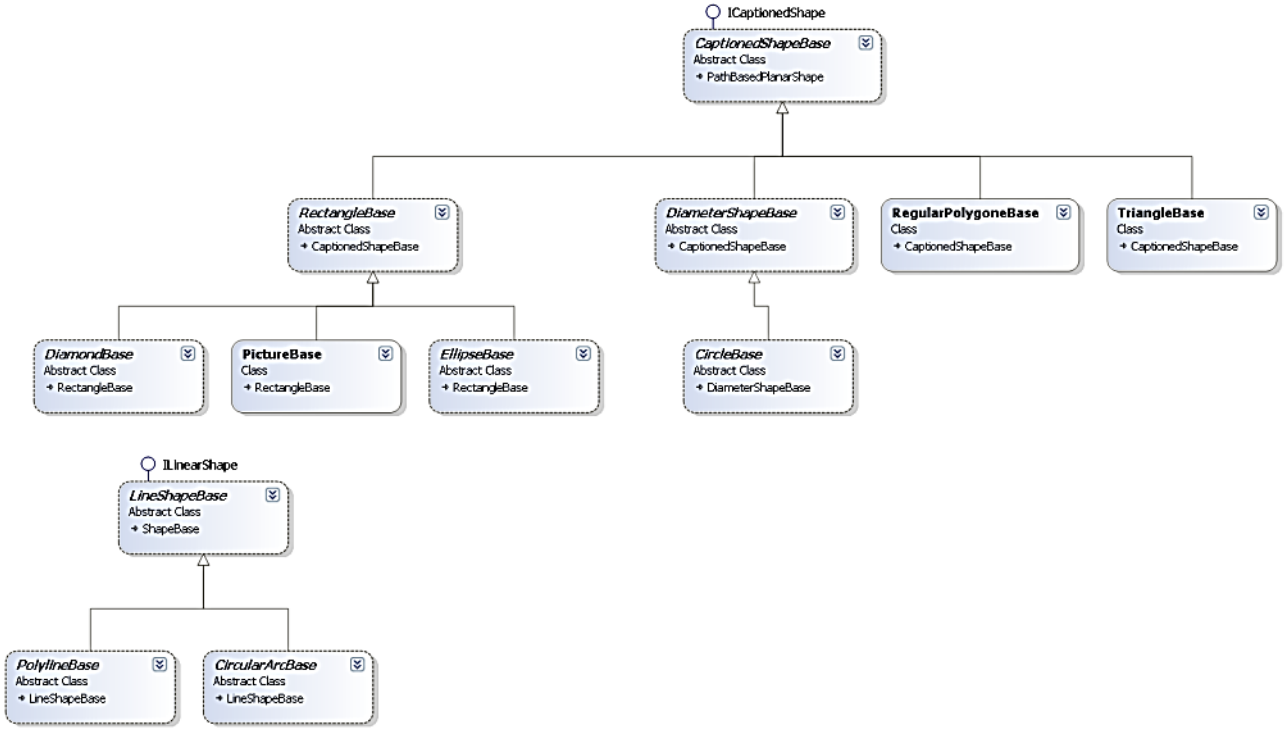

Рисунок 4. 2 - Диаграмма классов для базовых классов фигур палитр

Таблица 4.2 - Основные классы компонента NShape

Название	Описание
NShapeLibraryInitializer	Основной класс библиотек палитр, реализующий инициализацию библиотеки (метод Initialize). Использовался для реализации палитр UML (диаграмм классов, активностей и вариантов использования) и декларативного представления (палитра предикатов).

CaptionedShapeBase, CircleBase, CircularArcBase, DiamondBase, DiameterShapeBase, EllipseBase, ImageBasedShape, LabelBase, LineShapeBase, RectangleBase, и т.д.	Базовые классы для реализации палитр примитивов. Основные фигуры для палитры UML реализовывались на базе данных классов, путем изменения различных атрибутов и переопределения методов Attribute, CreateInstance, Clone, InitializeToDefault, CalculatePath и некоторых других. Отличие между базовыми классами заключается в способе задания фигур. Например, фигуры, унаследованные от CircleBase, задаются координатами центра (left, top) и двумя диаметрами (для описания эллипсов), тогда как для RectangleBase задаются ширина и высота.
Shape, IShapeCollection	Базовый класс для фигуры (в дальнейшем необходимо приведение к нужному типу фигуры) и интерфейс для работы с коллекцией фигур (diagram.shapes).
ControlPointId	Архитектура редактора реализована таким образом, что связи между фигурами представляют собой обычные фигуры (унаследованные от LineShapeBase, PolylineBase и т.д.) и содержатся в массиве diagram.shapes. Так как каждая фигура содержит определенный набор точек сопряжения, то типичное задание связи выглядит следующим образом: arrow.Connect(ControlPointId.FirstVertex, referringShape, ControlPointId.Reference);
Dataweb.NShape.Advanced. History	Класс управления истории изменений; используется при реализации динамической визуализации. Класс содержит методы отмены и повторения сделанных изменений.

ILayouter	Интерфейс для реализации автоматической расстановки фигур на диаграмме. Так как при переводе из программных проекций (текстовых) возникает задача расстановки примитивов на диаграмме, то имеет смысл рассмотреть некоторые из них.
ExpansionLayouter	Расстановка расширения. Сдвигает или раздвигает фигуры, не изменяя их относительное положение между собой. Для расстановки задаются коэффициенты относительного вертикального и горизонтального расширения или сжатия. Данная расстановка применима только к уже расставленным фигурам для того, чтобы уменьшить или увеличить расстояние между ними.
FlowLayouter	Потоковая расстановка. Группирует связанные фигуры в слои, так чтобы большинство связей (стрелок) в слое указывало в одном направлении. Для наилучшего распределения фигур по слоям используется метод градиентного спуска. Сам алгоритм расстановки состоит из трёх этапов: 1. Оптимизация уровней: Перемещение одиночных (не связанных) фигур вверх или вниз по одному слою, если он улучшает свои характеристики. 2. Сортировка: Расстановка фигур внутри одного слоя таким образом, чтобы связи не пересекались между собой или пересекались минимальное количество раз. 3. Позиционирование: Расстанавливает фигуры в

	пределах одного слоя так, чтобы связанные фигуры находились ближе друг к другу. Эти шаги повторяются до тех пор, пока не будет достигнуто нужного состояния или пока не пройдёт максимальное количество шагов. В алгоритм задаётся три основных параметра: LayerDistance - задаёт минимальное расстояние между слоями. RowDistance - задаёт минимальное расстояние между фигурами в пределах одного слоя. Direction - направление потока связей (стрелок).
GridLayouter	Расстановка по сетке. Позиционирует все фигуры на узлах прямоугольной сетки. Алгоритм предусматривает нахождение наиболее лучшего расположения фигур по сетке заданного размера. Для сетки задаются размер ячеек. Хорошо показала себя при расстановке диаграмм активностей и диаграмм классов UML.
RepulsionLayouter	Расстановка отталкивания и притяжения. Расстанавливает фигуры так, чтобы связанные фигуры находились ближе друг к другу, а несвязанные - дальше друг от друга. Работает по принципу моделирования физических сил отталкивания и сжатия пружины между соединёнными фигурами. Для алгоритма задаются некоторые основные физические параметры: Friction - сила трения. При увеличении уменьшает силу отталкивания и сжатия пружины.

	Mass - вес фигур. Repulsion - сила отталкивания. RepulsionRange - радиус действия силы отталкивания. SpringRate - сила сжатия пружины. Хорошо работает для декларативного представления и диаграммы вариантов использования.

UML диаграммы для описанных выше классов представлены на рисунке 4.3:

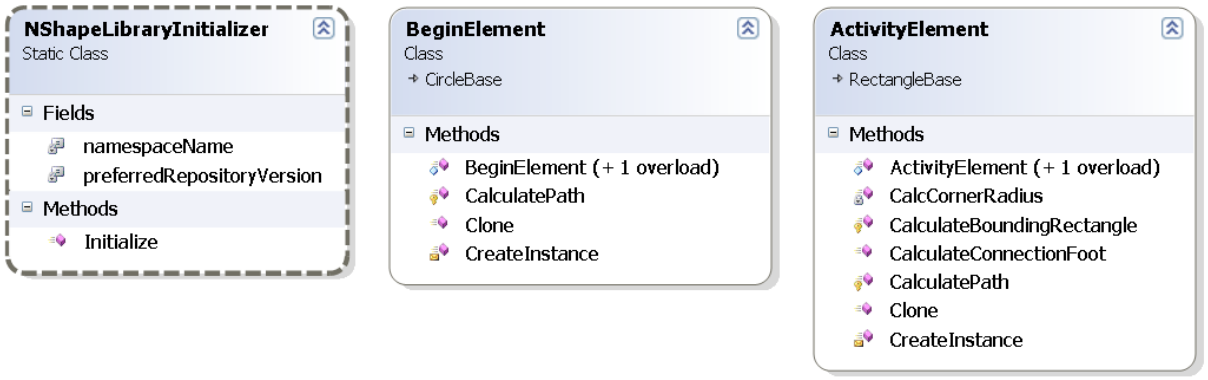

Рисунок 4. 3 - Диаграмма классов (без описания методов и атрибутов) для классов расстановки

Как отмечалось ранее (в третьей главе) наилучшие позиционирование удалось достичь при последовательном применении нескольких алгоритмов расстановки.

Виды динамической визуализации

В рамках специализированного графического редактора реализованы следующие виды динамической визуализации.

1. **Последовательная визуализация графической модели**. Как отмечалось ранее, текстовая проекция изобразительных моделей представляет собой программу рисования и генерируется автоматически (описание генерации будет представлено далее). В псевдокодовом интерпретаторе существует возможность пошагового исполнения (или исполнение с задержкой) такой программы, что

позволяет визуализировать диаграмму поэтапно. По умолчанию программа рисования создает сначала все вершины, а затем связи между ними. Исходный код программы (в виде QA-единиц) доступен для редактирования и может быть изменен для получения произвольной визуализации, как было показано в главе 3 в примере внедрений в ОАО НПО «Марс», где при отладке модуля имитации движения объектов требовалось наглядно показать траекторию движения.

2. **Визуализация история рисования**. Данный тип позволяет с определенной задержкой (либо в пошаговом режиме) представлять процесс рисования произвольной диаграммы и доступен с помощью следующей панели инструментов (рисунок 4.4):

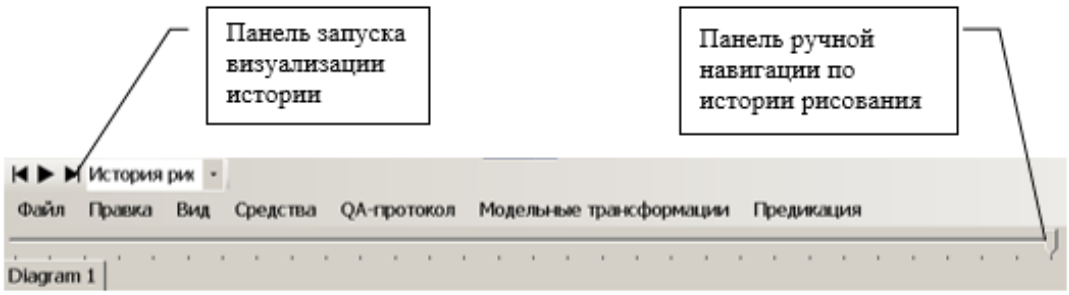

Рисунок 4. 4 - Панель инструментов, позволяющих работать с истории рисования

В текущей версии специализированного графического редактора история доступна в рамках одной сессии работы с редактором (после закрытия редактора история пропадает) либо при исполнении псевдокодовой программы рисования. Реализация указанного вида осуществляется за счет использования функций Redo, Undo, класса Dataweb.NShape.Advanced.History, которые выполняют соответственно отмену или повторение сделанных изменений. Дальнейшее улучшение данного вида визуализации заключается в сохранении истории в виде псевдокодовой программы, что обеспечит сохранение истории между запусками редактора.

3. **Использования различных вариантов расстановки**. Как отмечалась ранее (в таблице 4.2) в специализированном редакторе реализована функция расстановки диаграмм 4 видами или их комбинациями. Например, при расстановке декларативной модели необходимо использовать расстановку, учитывающую количество входных и исходящих связей у вершины, так как это количество является критерием «важности» вершины (указывает на то, что несколько предложений ссылаются на данный объект). Комбинация расстановок доступна за счет последовательного применения к созданной диаграмме различных вариантов.
4. **Реализация последовательной демонстрации графических моделей**. Для реализации возможности последовательной визуализации созданных графических моделей (сохраненных в формате специализированного графического редактора) в виде слайд шоу была создана псевдокодовая функция DD_OpenDiagram, которая принимает в качестве аргумента путь до файла диаграммы. При последовательном вызове этой функции происходит загрузка редактором указанных диаграмм и имитируется слайд шоу.

Идеи влияния на внимание разработчика (за счет динамической визуализации) основаны на естественном рефлексе концентрации восприятия на движущихся и изменяющихся объектах. При внимании в первую очередь активизируются лобные отделы головного мозга (в том числе передняя часть поясной извилины), которые тесно связаны с рабочей памятью [45, 26]. При нарушении работы в данной области мозга приводит к появлению навязчивых мыслей, нерешительности и невозможности находить решения [4]. Исследования наподобие [69] показывают, что внимание непосредственно влияет на процессы решения задач и в том числе на озарение (инсайт), поэтому, активация этих процессом является важным.

4.1. Реализация изобразительного представления

Поддержка изобразительного в псевдокодовом интерпретаторе была осуществлена ранее [85], но была не полной, так как отсутствовали методы экспорта произвольной диаграммы и дополнительные палитры, а также методы расстановки примитивов на диаграмме. Поэтому в задачу входило разобраться в текущей реализации и разработать недостающий функционал (таблица 4.3) и обеспечить полноценную возможность динамической визуализации за счет пошагового исполнения программы рисования в псевдокодовом интерпретаторе.

В псевдокодовом интерпретаторе реализованы следующие функции:

Таблица 4.3 - Описание основных псевдокодовых функций использующихся в изобразительном представлении

Название	Описания	Параметры
DD_Create	создание нового графического примитива на основе существующего шаблона (палитры)	({TemplateName}, “ShapeName”={ShapeName}, [{PropertyName=PropertyValue}]), где TemplateName – имя фигуры из палитры ShapeName – текстовое наполнение фигуры PropertyName/PropertyValue – название и значение определенного свойства.
DD_Update	обновление созданного графического примитива	({ShapeName}, [{PropertyName=PropertyValue}])
DD_Delete	удаление примитива	({ShapeName})
DD_Link	связывание двух	(ShapeNameSrc, ShapeNameDst,

	графических примитивов	LinkTemplate), где: ShapeNameSrc – наименование фигуры - источника ShapeNameDst – наименование фигуры - назначения LinkTemplate – имя шаблона, используемого для обозначения связи (шаблон стрелки)
DD_LinkDiagram	связывание двух диаграмм	({ShapeName}, {QAIDDst}), где: ShapeName – наименование фигуры, при клике на которую должен быть осуществлен переход на другую диаграмму. QAIDDst – идентификатор вопросно-ответной единицы, соответствующей диаграммы, на которую должен быть осуществлен переход.
DD_AutoLayout	Производит расстановку фигур на диаграмме согласно выбранным параметрам расстановки	({Type}, {Params}). Доступны следующие настройки: Тип: FLOW, парметры: DIRECTION, LAYERDISTANCE, ROWDISTANCE; Тип: GRID, параметры: X, У; Тип: EXPANSION, параметры HORIZONTAL, VERTICAL; Тип: REPULSION, параметры SPRINGRATE, REPULSION, REPULSIONRANGE, FRICTION,

		MASS. При отсутствие параметров будет выполнена расстановка по умолчанию (Repulsion).
DD_Clear	Удаляет все фигуры на диаграмме	нет
DD_OpenDiagram	Запускает редактор и загружает указанный в параметре файл диаграммы	({File}) – путь к файлу диаграммы, который необходимо открыть при загрузке редактора. Используется при реализации 4 вида динамической визуализации

При переводе графических проекций в диаграмму рисования были задействованы только методы создания и связывания фигур, как показано на рисунке 4.5 (показаны только методы, которые были реализованы).

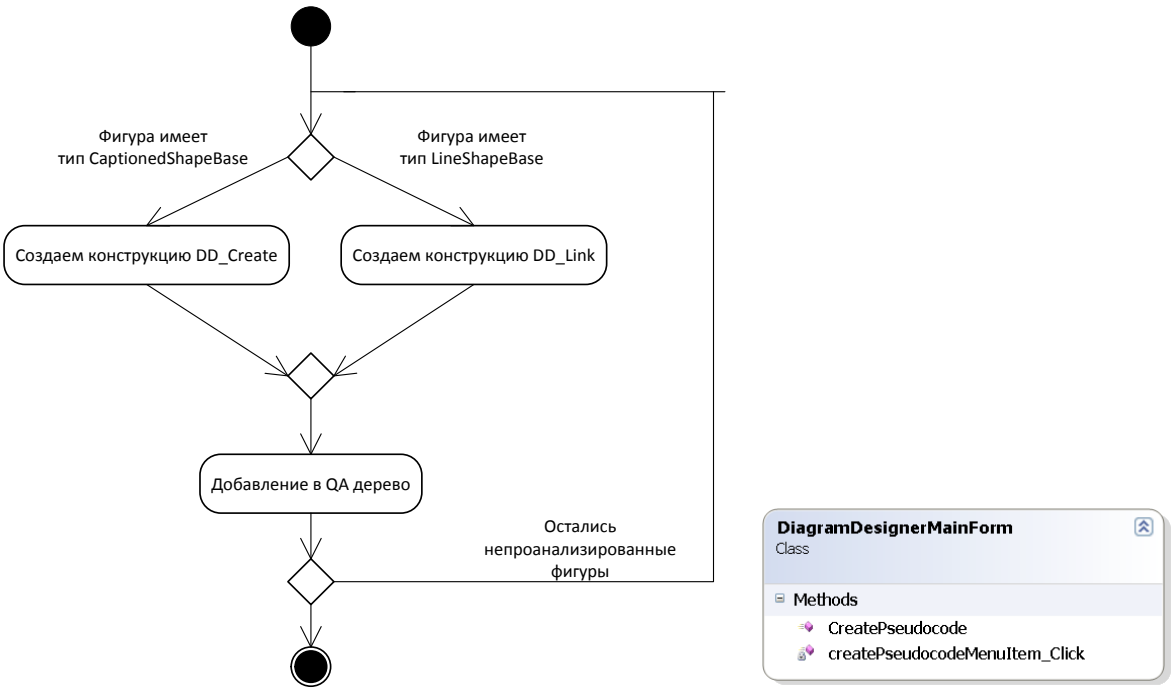

Рисунок 4. 5 - Диаграмма активностей и классов для перевода в псевдокодовую программу рисования

4.2. Реализация декларативного представления

Модуль поддержки декларативного программирования (трансформация прологоподобных форм в модели и обратно)

Палитра графических примитивов для редактора диаграмм представляет собой отдельную динамически загружаемую библиотеку, которая содержит себе описание нужных для работы геометрических фигур. Подключение произвольной библиотеки палитр происходит при помощи вызова функции AddLibraryByName (string assemblyName, bool unloadOnClose = true), где assemblyName – это имя библиотеки. Во время загрузки палитры происходит поиск статического класса NShapeLibraryInitializer, при помощи которого в методе Initialize регистрируются классы графических примитивов. После регистрации они становятся доступными в палитре редактора. Диаграмма классов созданной библиотеки фигур "Dataweb.NShape.PredicateShapes" представлена на рисунке 4.6.

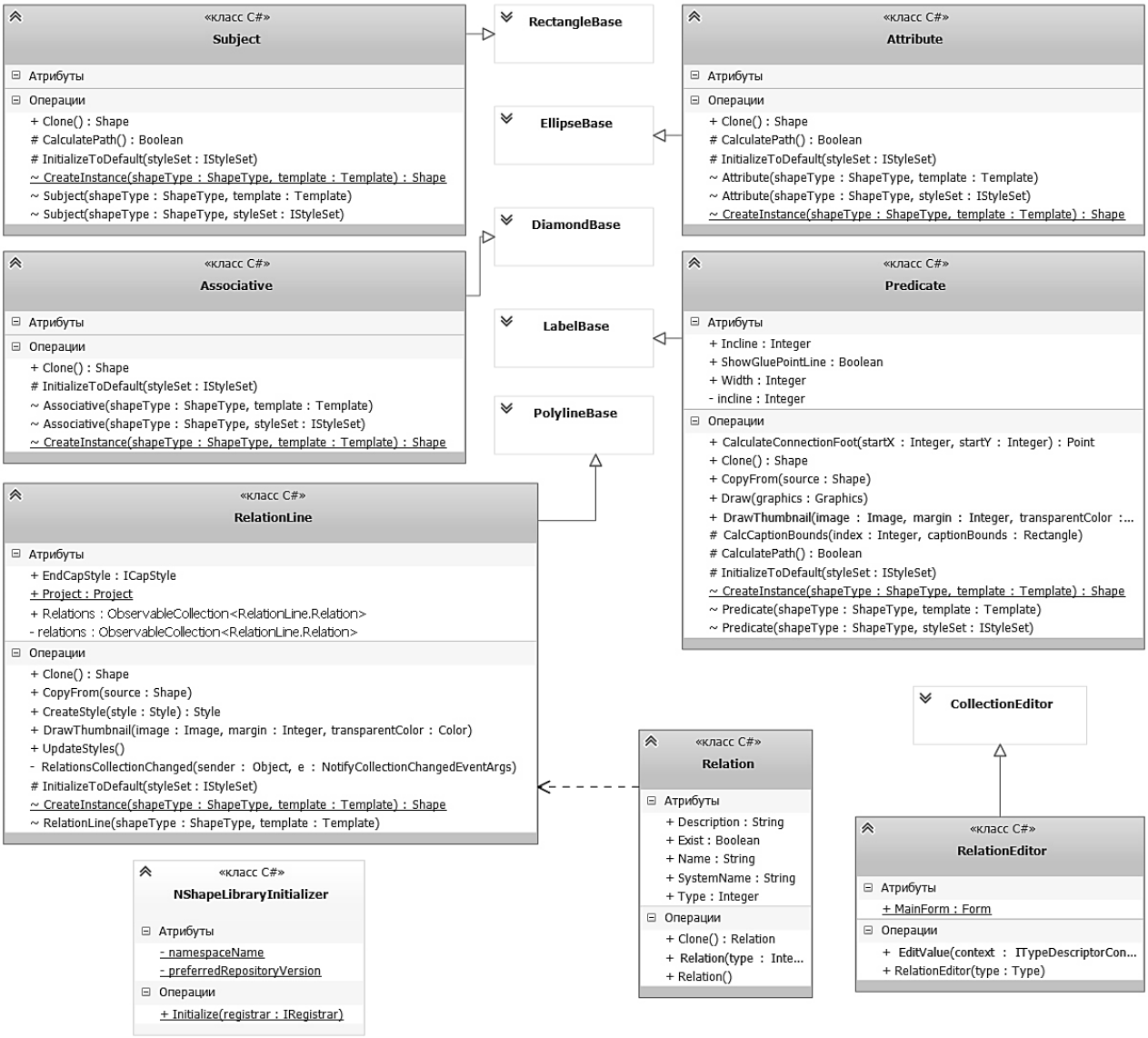

Рисунок 4. 6 - Диаграмма классов библиотеки фигур для декларативного представления

В качестве namespaseName выступает название палитры, которое будет отображаться, preferredRepositoryVersion отвечает за вывод версии библиотеки. Эта переменная не имеет особо значимого применения, поэтому может иметь любое значение. NewShape отвечает за имя класса фигуры.

В библиотеке Dataweb.NShape.PredicateShapes регистрируются следующие типы фигур:

- Predicate – отвечает за представление предиката и наследуется от класса LabelBase.
- Subject – отвечает за представление субъекта и наследуется от стандартного класса RectangleBase, описывающий прямоугольник.

- Associative – отвечает за представление ассоциативной составляющей и наследуется от базового класса DiamondBase, описывающий ромб.
- Attribute – отвечает за представление свойства и наследуется от класса EllipseBase, описывающий эллипс.
- RelationLine – отвечает за представление связи, наследуется от класса PolylineBase, описывающий линию.

Базовые классы, от которых наследуются графические примитивы, находятся в стандартной библиотеке Dataweb.NShape.dll редактора диаграмм. Для примитивов переопределяются следующие функции базовых классов:

- CreateInstance – создаёт и возвращает экземпляр класса.
- Clone – возвращает полную копию объекта.
- InitializeToDefault – инициализирует поля и свойства начальными значениями. Задаётся начальный размер и при помощи передаваемого в функцию аргумента styleSet задаются начальные стили отображения.
- CalculatePath – регистрирует точки, по которым рисуется фигура.
- CopyFrom – производит копирование всех полей класса в другой экземпляр.
- DrawThumbnail – производит рисование миниатюрной копии изображения примитива для использования его в окне выбора фигуры.
- CalculateConnectionFoot – регистрирует активные точки, к которым можно присоединять линии.
- CalcCaptionBounds – указывает расположение надписи у фигуры.

В классе Predicate для регулирования наклона параллелограмма включено свойство Incline. Это свойство используется в некоторых функциях для определения положения точек рисования. А изменение свойства ShowGluePointLine позволяет показывать или скрывать линию соединения надписи с каким-либо примитивом.

В классе RelationLine для поддержки изменения стилей отображения связей, в зависимости от хранящихся в ней отношений, определены следующие поля и методы:

- Relations – список отношений, хранящихся в связи.
- Project – статическое поле, отвечающее за доступ к текущим настройкам редактора диаграмм. Через него будут изменяться и добавляться новые стили для отображения связей.
- EndCapStyle – стиль отображения стрелки на конце линии.
- CreateStyle – если нужный стиль для связи ещё не был создан, то создаёт его и добавляет в список стилей редактора, используя переменную Project.
- UpdateStyle – обновляет изображения связей в зависимости от типов и количества отношений.
- RelationCollectionChange – вызывается при изменении списка отношений и обновляет отображение связей.

Для хранения типов отношений в классе RelationLine был создан класс Relation со следующими полями:

- Description – описание отношения.
- Exists – отвечает за значение того, существует ли в данный момент это отношение между понятиями. В зависимости от этого значения изменяет тип линии при отображении связи.
- Name – полное имя отношения.
- SystemName – системное имя, взятое из онтологического словаря.
- Type – тип или идентификатор отношения.

Для того, чтобы иметь возможность изменять список отношений при помощи интерфейса был создан класс RelationEditor, который наследуется от класса CollectionEditor и позволяет менять способ изменения коллекции. При помощи функции EditValue вызывается окно интерфейса, которое храниться в статической переменной MainForm. Определения для свойства Relations класса

RelationLine атрибута Editor("Dataweb.NShape.PredicateShapes.RelationEditor, Dataweb.NShape.PredicateShapes", typeof(UITypeEditor)) позволяет вызывать интерфейс из окна свойств редактора диаграмм (рисунок 4.7).

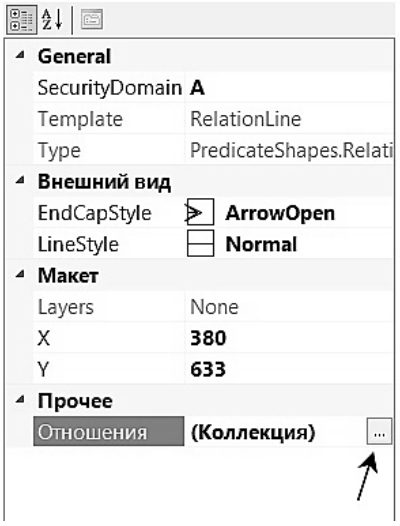

Рисунок 4. 7 - Задания отношения в окне свойств

Средства изменения отношений между понятиями

Для того, чтобы изменять отношения между понятиями в графическом представлении был создан специальный интерфейс. Диаграмма классов интерфейса представлена на рисунке 4.8.

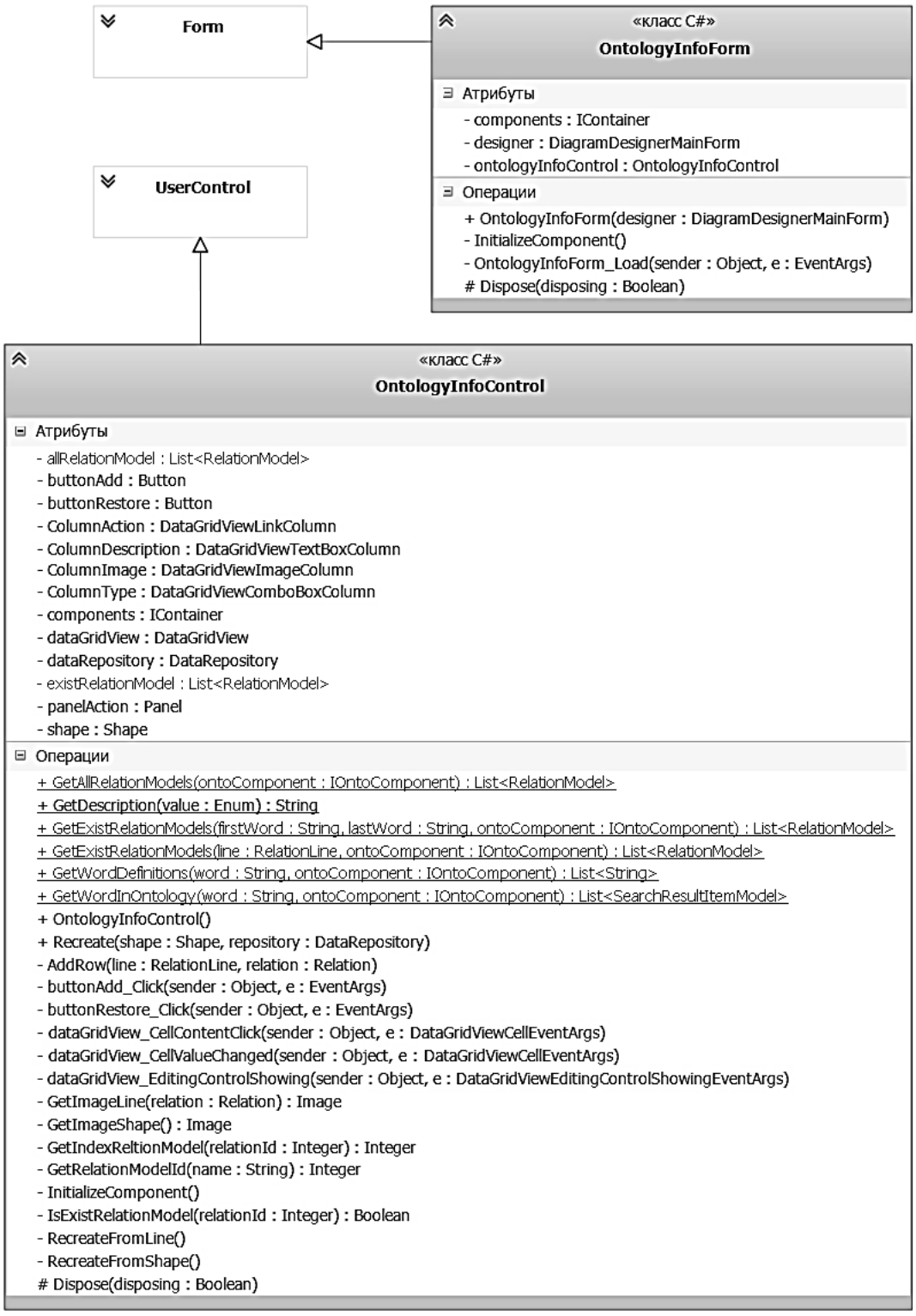

Рисунок 4. 8 - Диаграмма классов интерфейса для изменения отношений

Класс OntologyInfoForm представляет собой окно редактирования отношений для связей с типом RelationLine. Экземпляр этого окна записывается в статическую переменную MainForm класса RelationEditor, для того чтобы вызывать его из окна свойств редактора диаграмм. Для доступа к редактору и

обновлению его содержимого во время инициализации в экземпляр класса передаётся ссылка на редактор. Интерфейс окна состоит из пользовательского контроллера OntologyInfoControl. Внешний вид интерфейса представлен на рисунке 4.9.

Рисунок 4. 9 - Интерфейс для изменения отношений

Интерфейс OntologyInfoControl имеет в себе следующие основные методы:

- GetAllRelationModels – получает список всех отношений, которые хранятся в онтологическом словаре.
- GetExistRelationModels – получает список существующих связей между двумя понятиями.
- Recreate – заполняет таблицу данных в элементе управления для переданной фигуры.
- GetImageLine – получает изображение линии на основе переданного отношения (для обозначения каждого из отношений используется свой тип линии)
- AddRow – добавляет новую строку для редактирования отношения.
- GetRelationModelId – получает идентификатор отношения по его названию.
- IsExistRelationModel – проверяет, существует ли переданное отношение между связанными фигурами.
- GetWordInOntology – ищет переданное слово в онтологическом словаре и возвращает найденные для него модели.

Во время вызова интерфейса вызывается функция Recreate, которая в качестве аргумента принимает фигуру, для которой интерфейс вызывается. На основе отношений, которые уже существуют у этой фигуры, отображаются строки, в которых приведено описание этих отношений и их формальное изображение.

Элемент управления основан на классе DataGrid, что позволило создать редактируемые ячейки с привязкой к коллекции отношений. Строка данных состоит из ячейки с изображением, ячейки с выпадающим списком, ячейки с текстовым полем и ячейки с кнопкой. В ячейке с названием «Тип» содержится выпадающий список со всеми отношениями, которые сейчас существуют в онтологии. При его изменении меняется изображение в ячейке с названием «Изображение».

Программная проекция декларативного представления

Программная проекция декларативного представления записывается в QA-протоколе посредством включения в него предложений описания. Чтобы создать по этому описанию связанные геометрические фигуры требуется разделить его предложения на составляющие. Для этого был создан ряд классов для хранения составляющих предложения и алгоритмов работы с ним. Диаграмма классов, описывающая структуры для работы с программной проекцией декларативного представления, показана на рисунке 4.10.

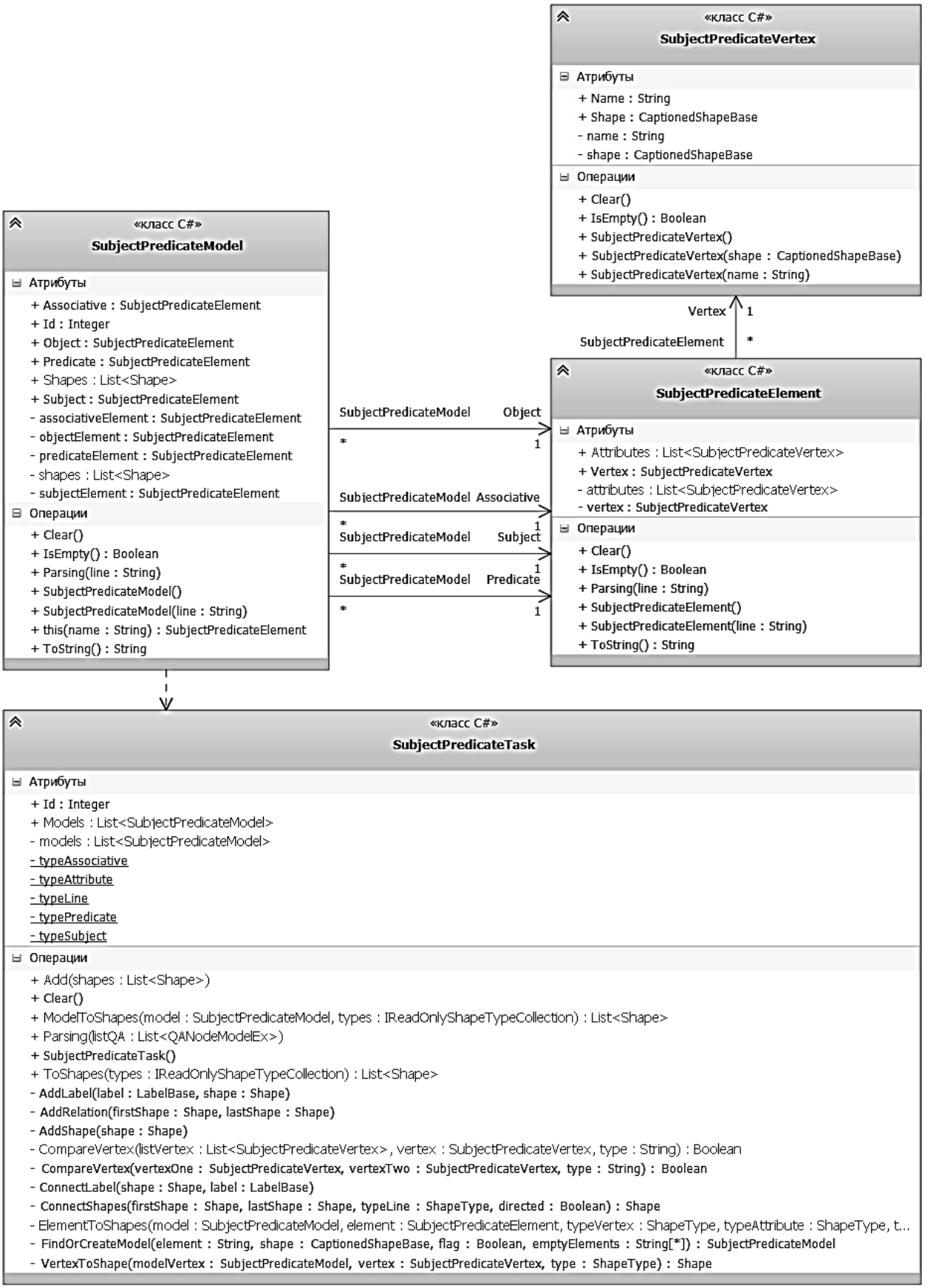

Рисунок 4. 10 - Диаграмма классов для работы с программной проекцией декларативного представления

Главный класс SubjectPredicateTask отвечает за хранение всех предложений описываемой задачи и за преобразование их в графическую проекцию. В классе определены следующие основные методы и поля:

- Models – список разобранных предложений декларативного представления.
- Id – идентификатор QA-задачи.
- ModelToShapes – переводит указанную в параметры программную проекцию в графическую форму. В качестве второго параметра указывается список всех примитивов редактора.
- Parsing – разбирает все предложения, указанные в надписях, переданных QA-улов, тем самым заполняет класс внутренним представлением.
- ToShapes – переводит понятийное описание в графические примитивы.
- Add – добавляет в класс список геометрических фигур и преобразовывает их в понятийное представление.

Класс SubjectPredicateModel отвечает за описания одного предложения декларативного описания. Основные его функции – это разбирать предложение на составляющие, осуществлять хранение его внутреннего представления и создание по нему геометрических фигур. Класс имеет следующие основные поля и методы:

- Id – идентификатор QA-узла.
- Associative – ассоциативная составляющая.
- Object – объект.
- Subject – субъект.
- Predicate – предикат.
- Shapes – список созданных фигур для этого QA-единицы.
- Parsing – метод разбора предикатной формы.
- ToString – переводит внутреннее представление понятийного описания в строку текста.
- Clear – полностью очищает весь класс от данных.

- IsEmpty – показывает, пустая ли строка храниться в модели.

Класс SubjectPredicateElement отвечает за хранение отдельных элементов, таких как субъект, объект, ассоциативная составляющая или предикат. Класс имеет следующие основные поля и методы:

- Vertex – основная вершина элемента.
- Attributes – свойства элемента.
- Parsing – разбор переданной строки вида <имя элемента>, <свойства> на составляющие и заполнение значения класса.
- Clear – полная очистка класса от данных.
- ToString – перевод внутреннего представления элемента понятийного описания в текстовую строку.
- IsEmpty – проверка на пустоту значений класса.

Класс SubjectPredicateVertex отвечает за хранение отдельного слова понятийного описания. Имеет в себе следующие основные поля и методы:

- Name – описываемое слово.
- Shape – созданная геометрическая фигура.
- Clear –метод, который очищает класс от данных.
- IsEmpty – метод, который проверяет класс на пустоту.
- Конструктор SubjectPredicateVertex – принимает в качестве аргумента фигуру, из которой выделяется название, или слово, и заполняет класс данными.

Автоматизированное преобразование

Для того, чтобы поддерживать автоматические преобразования между программной и графической проекцией декларативного представления был создан класс PredicateDiagramWork, который отвечает за автоматические преобразования при изменении одного из связанных представлений. Диаграмма класса представлена на рисунке 4.11.

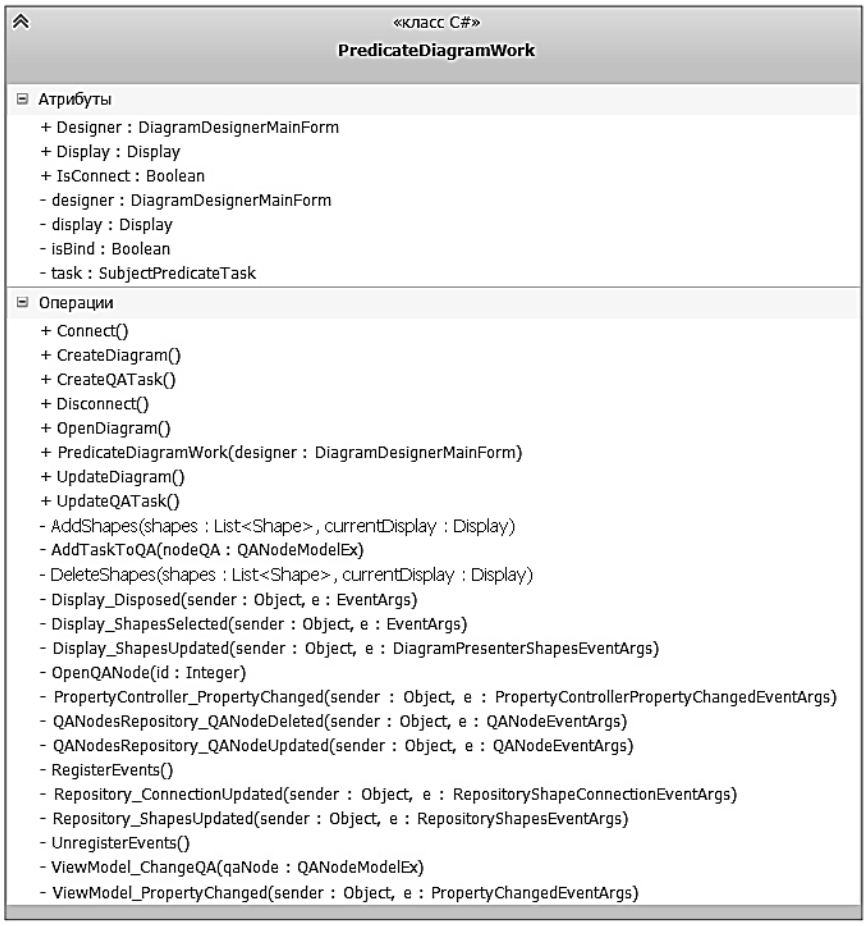

Рисунок 4. 11 - Класс PredicateDiagramWork

Класс содержит в себе следующие основные поля и методы:

- Designer – ссылка на редактор диаграмм.
- Display – ссылка на привязанную диаграмму.
- Connect – метод, который связывает диаграмму и QA-узел между собой.
- Disconnect – отвязывает диаграмму от QA-узла.
- CreateDiagram – создает фигуры и добавляет их на пустую диаграмму.
- CreateQATask – создает программное представление в виде QA-задач.
- OpenDiagram – выбирает и отображает привязанную диаграмму.
- UpdateDiagram – обновляет изображения привязанной диаграммы, добавляет новые фигуры и осуществляет их автоматическую расстановку.

- UpdateQATask – обновляет привязанную QA-задачу, добавляет или изменяет программную проекци. на основе геометрических фигур, хранящихся в привязанной диаграмме.
- AddShapes – добавляет новые фигуры на диаграмму.
- AddTaskToQA – добавляет QA-узлы в дерево задач.

Модуль перевода и загрузки прологоподобных форм в язык PROLOG

В качестве основы для среды исполнения предикатов была взята свободная реализация языка пролог – SWI-Prolog. Эта среда обладает богатым набором возможностей (большое количество открытых библиотек, интерфейс к языку java, ODBC и т.д.), хорошей стабильностью работы (среда развивается с 1987г.) а также является достаточно популярной в университетской среде. Для взаимодействия с интерпретатором была написана прослойка, занимающаяся проксированием запросов из внешнего интерфейса в интерпретатор и обратно. Такая схема взаимодействия со средой отличается большой гибкостью и может работать практически с любым интерпретатором пролог. Общая схема взаимодействия среды с SWI-Prolog [61] представлена на рисунке 4.12.

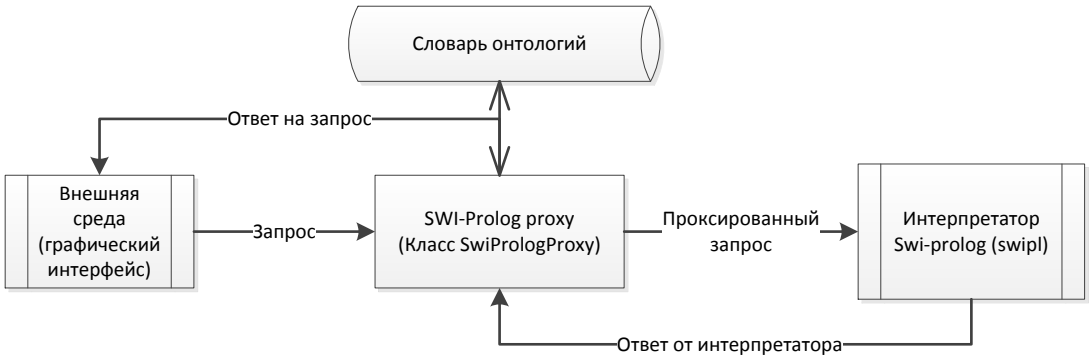

Рисунок 4. 12 - Общая схема взаимодействия с пролог интерпретатором

SwiPrologProxy является частью среды WIQA и призван решать следующие задачи:

1. Проксирование запросов из среды в интерпретатор. Синтаксисы предикатов, используемые в среде и в интерпретаторе, отличаются, поэтому приходится производить преобразование из одного формата в другой на лету. Это становится возможным, так как синтаксис,

используемый в WIQA и синтаксис SWI-Prolog имеют определенное соответствие и не зависят от состояния интерпретатора.

2. Обработка дополнительных команд. Некоторые команды, посланные в прокси не должны проксироваться в интерпретатор. К таким командам преимущественно относятся служебные команды (очистка экрана консоли, статистика и т.д.), а также команды взаимодействия со словарем онтологий.
3. Обработка ответов интерпретатора. Для выполнения команды интерпретатору необходимо некоторое время. После выполнения запроса в интерпретатор класс прокси ожидает прихода определенной последовательности символов, которые служат сигналом о том, что запрос был выполнен. К таким символам относятся символы перевода строки (“\r\n”). После получения такой последовательности прокси считывает результат и передает его дальше на обработку.
4. Обработка успешного результата. В случае, если результат выполнения успешен, прокси передает ответ в неизменном виде в графическую среду. В случае, если ответ представляет собой ошибку, происходит обработка ответа с целью приведения к оригинальному синтаксису (как было описано выше синтаксисы предикатов отличаются).

В связи с вышеизложенными требованиями реализация класса SwiPrologProxy не является достаточно простой. Общий размер кода класса порядка 500 строк. Диаграмма классов представлена на рисунке 4.13.

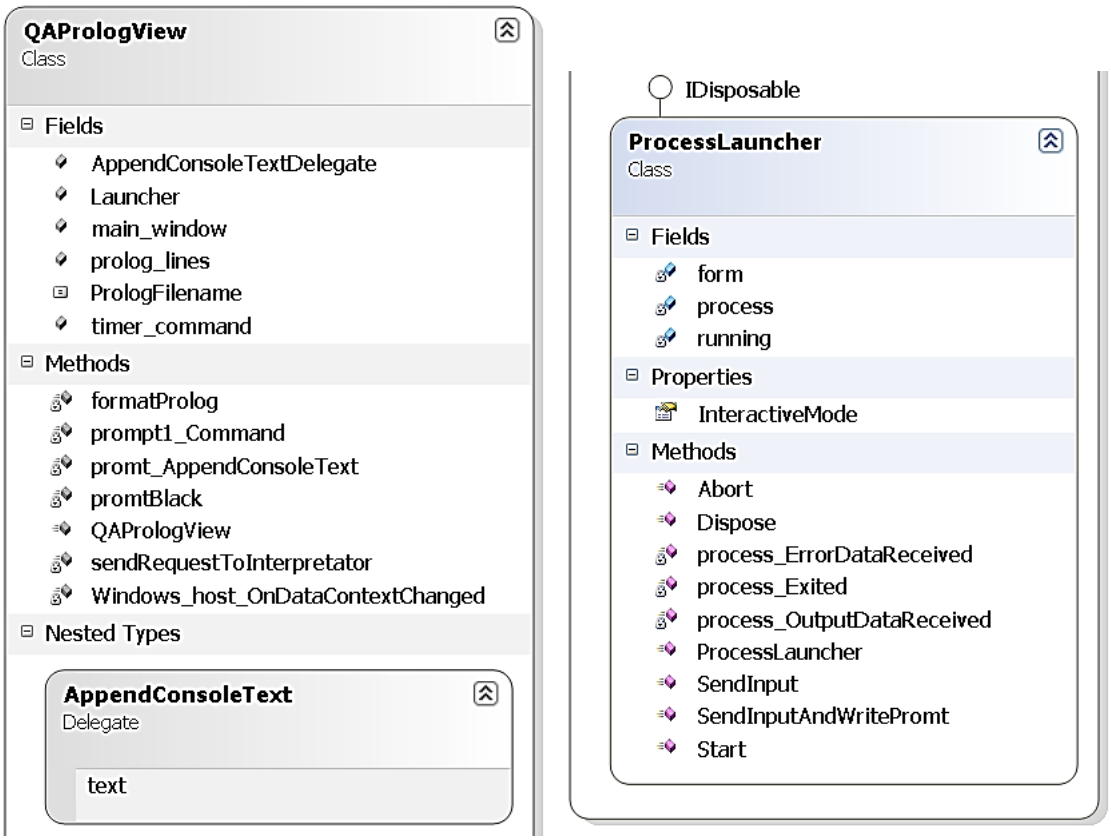

Рисунок 4. 13 - Диаграмма классов модуля перевода и исполнение пролог подобных форм в интерпретаторе PROLOG

4.3. Реализация концептуально-алгоритмического представления

Модуль перевода концептуально-алгоритмических форм в графические модели и обратно

При реализации концептуально-алгоритмического представления необходимо было разработать палитру фигур. Как было описано ранее, палитры реализованы в виде динамически подключаемых библиотек, а примитивы построены на основе базовых классов фигур. Для концептуально-алгоритмического представления было реализовано три вида палитр: палитра диаграмм активностей UML 2.x, палитра диаграмм вариантов использования (была заимствована из примеров палитры NShape general, поэтому описываться не будет) и диаграмм классов. Рассмотрим примитивы палитр более подробно:

1. Палитра диаграммы активностей (деятельности)
    a. ActivityElement – представляет собой прямоугольник со скругленными краями. Содержит текстовое поле, которое

используется для задания псевдокодовых операций или другой активности. Наследуется от класса RectangleBase.

b. BeginElement – используется для указания точки входа в диаграмму; наследуется от CircleBase.
c. Decision – используется для реализации условий, содержит текстовое поле использующееся для описания условия на языке псевдокода или в произвольной форме. Наследуется от DiamondBase.
d. EndDecision – блок завершения условия. Так как диаграмма активностей основана на механизмах сетей Петри [105], то для блока условий требуются блоки, в которых маркеры будут объединяться (это справедливо и для разделения на потоки). Класс является наследником от DiamondBase, но в отличие от Decision не имеет текстового поля.
e. EndElement – представляет собой примитив, указывающий на завершения обработки процесса; не содержит текстовых полей доступных для редактирования. Модель может иметь несколько выходов. Наследуется от CircleBase.
f. Label – служит для добавления примечаний. Не участвует в процессах трансформации (игнорируется). Наследуется от LabelBase.
g. ObjectElement – служит для описания объектов. На диаграммах 2 версии появилась возможность описания траектории или потока объектов (показать изменение/передачу объектов между сущностями). Наследуется от класса RectangleBase.
h. Polyline – соединительная линия. Наследуется от PolylineBase.

2. Диаграммы классов.
    a. ClassBox – примитив для задания класса. Текстовая составляющая состоит из 3 блоков: название, описание полей, описание методов. Наследуется от класса RectangleBase.

b. RestrictionBox – служит для описания ограничений значений полей у класса (или ассоциативных ограничений). В UML принято описывать ограничения на языке OCL (Object Constraint Languages), но в нашем подходе для этих целей используется псевдокод, обладающей достаточной гибкостью. Наследуется от RectangleBase.

c. Line – необходима для задания отношений между классами. Наследуется от класса RectangularLineBase.

Средства поддержки создания, трансформирования и работы с диаграммами UML 2.0

При описании классов, задействованных в процессе трансформаций из текстового представления в графического, логично начать с описания внутреннего представлении модели как ориентированного графа, состоящего из массива вершин и связей (рисунке 4.14).

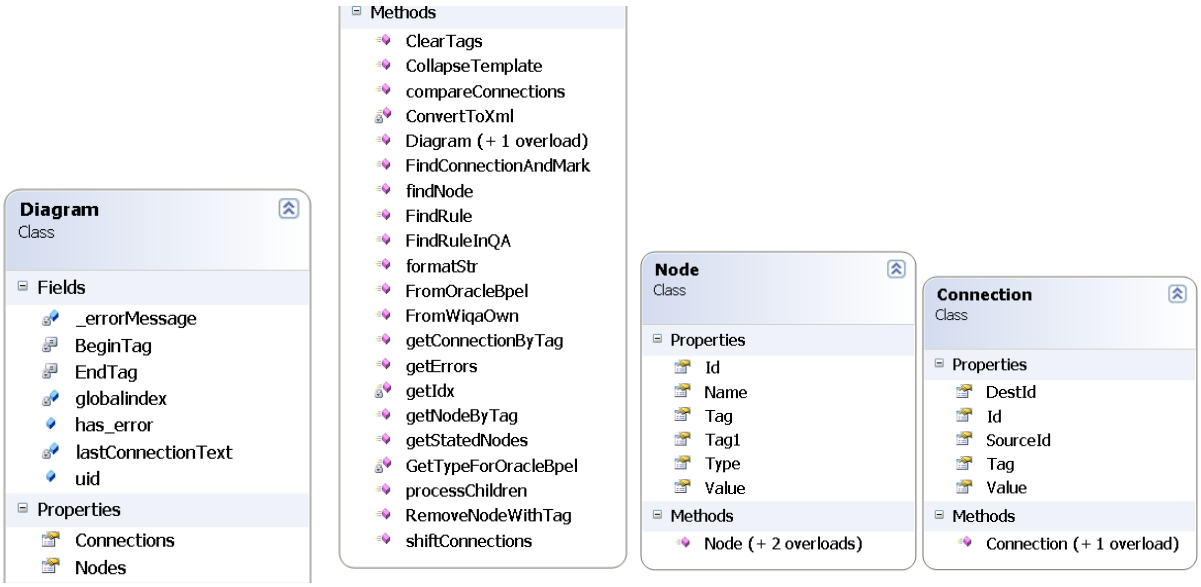

Рисунок 4. 14 - Диаграмма классов для внутреннего представления модели в виде ориентированного графа

Класс вершин (Node) содержит следующие поля и методы:

- Id – идентификатор вершины (uuid). Необходимо для связывания фигур.
- Name – название вершины

- Tag, Tag1 – служебные поля в которых хранятся метки при поиски шаблона для свертки
- Type – тип вершины (состоит из названия палитры и названия примитива)
- Value – текстовое наполнение вершины (псевдокод)

Класс связи (Connection) содержит следующие поля и методы:

- Id– идентификатор вершины (uuid)
- SourceId – идентификатор вершины из которой выходит связь
- DestId – идентификатор вершины в которую входит связь
- Tag – служебное поле для хранения метки в процессе сопоставление шаблона
- Value – подпись к связи

Класс диаграммы (Diagram) содержит следующие поля и методы (рассмотрим только основные):

- Nodes – список вершин
- Connections – список связей
- hasError – булево значение наличия или отсутствия ошибок на диаграмме
- FromWiqaOwn – метод создания диаграммы из файла формата Wiqa.Own
- FromOracleBpel – метод создания диаграммы из файла формата Oracle Bpel
- FindRule – метод поиска определенного правила на диаграмме
- FindRuleInQa – метод поиска правила в QA представлении
- CollapseTemplate – метод свертки найденного правила (помеченного тегами) согласно правилу трансформации
- RemoveNodeWithTag – удаление вершин, помеченных определенным тегом.

Далее логично рассмотреть классы, которые используются для представления результата перевода графической проекции в QA-представление и для хранения и загрузки правил трансформации (которые динамически загружаются из словаря правил rules.xml). Некоторые из этих классов представлены на рисунке 4.15.

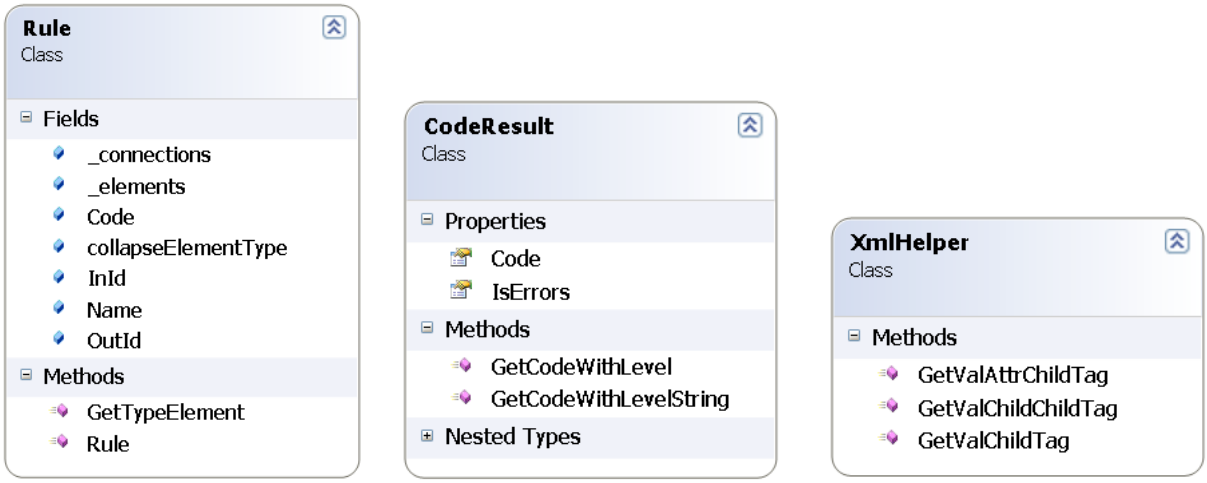

Рисунок 4. 15 - Диаграмма классов, используемых в средстве поддержки концептуально-алгоритмического представления

Класс для хранения правил (Rule) имеет следующие поля и методы:

- _connections – список связей
- _elements – список вершин
- Code – шаблон псевдокода, который должен быть сгенерирован при применении правила
- InId – идентификатор входной вершины (из массива _elements)
- OutId – идентификатор выходной вершины (из массива _elements)
- Name – название правила
- CollapseElementType – тип вершины, которая будет создана при применении правила

Класс для хранения результата перевода диаграммы в вопросно-ответное представление (CodeResult) имеет следующие поля и методы:

- Code – массив строк, содержавший программную проекцию КА представления

- IsError – булево значение указывающее на наличие или отсутствие ошибок
- GetCodeWithLevelString - метод, который возвращает массив объектов типа CodeLevel (содержит 2 поля текст и уровень)

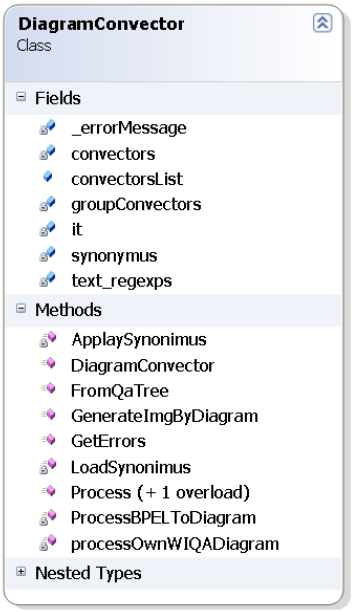

Рисунок 4. 16 - Диаграмма классов, используемых в средстве поддержки концептуально-алгоритмического представления

Класс перевода диаграммы в вопросно-ответное представление имеет следующие поля и методы (на рисунке 4.16):

- Convectors - массив правил трансформаций, объединенных в группу
- groupConvectors - словарь название группового правила/массива правил.
- Synonymus - массив синонимов для типов примитивов.
- Text_regexps – регулярное выражение для сгенерированного QA-представления
- applySynonimus – метод применения синонимов к указанной диаграмме
- FromQATree – метод генерации диаграммы из QA-дерева

- GenerateImgByDiagram – метод для отладки процесса перевода, позволяющий генерировать изображение (файл png) для диаграммы с помощью библиотеки Graphviz.
- GetErrors – метод, возвращающий список найденных ошибок

4.4. Оценка степени повышения эффективности проектировщика при использовании системы поддержки процесса решения задач

При оценке эффективности сложной системы, которой является представленная разработка, целесообразнее производить оценку эффективности отдельных ее компонентов. Оценка комплексного решения в данном случае является сложным, так как отсутствуют прямые аналоги разработанного подхода. В данном случае логично оценить эффективность средства поддержки декларативного и образно алгоритмического решения и на основе полученных результатов оценить предложенный метод в целом.

Оценка эффективности средства поддержки концептуально-алгоритмического представления

В промышленной индустрии разработки программного обеспечения (как и отечественной, так и в иностранной) существует ряд стандартов относительно видов, комплектов и требований к документации разрабатываемой системы. В РФ в настоящее время действуют ГОСТ 34 и 19 серий, а также РД 50-34.698-90 [75, 104, 92]. В западных странах существует ряд стандартов ISO/IEC 19505-1 и 19505-2 [20], которые набирают популярность в российской индустрии разработки ПО. В основе лежит использование визуальных моделей для описания поведения системы. Примеры диаграмм, описывающие алгоритмы представлены на рисунке 4.17.

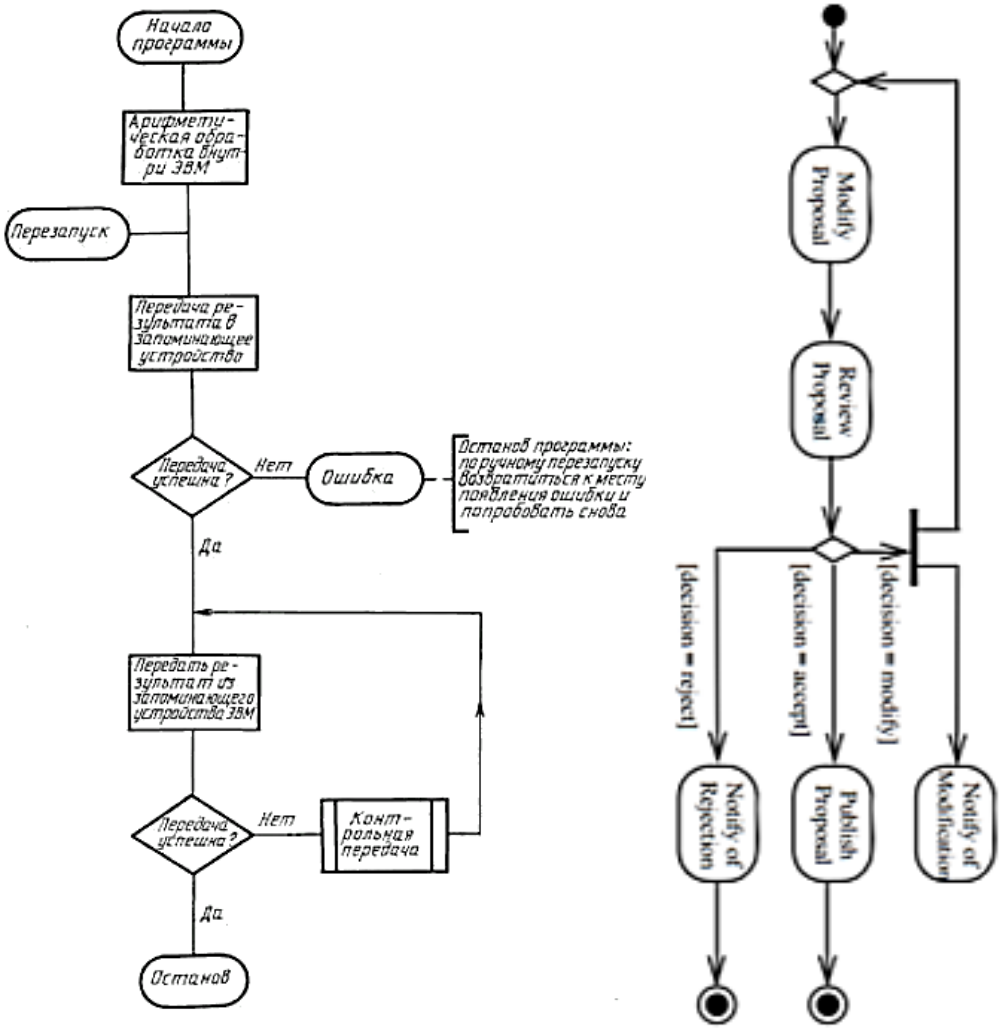

Рисунок 4. 17 - Пример алгоритма по ГОСТ 19.701-90 [103] и ISO/IEC 19505-2:2012

Очень часто в разработке АС по ГОСТ (или ISO) документацией и разработкой занимаются разные коллективы. В таком случае возникают проблемы, связанные с рассогласованностью кода и документации, а также с двойной работой по формализации алгоритма (программисты формализуют алгоритм в коде, аналитики в графическом редакторе). Разработанный подход, базирующийся на MDD нацелен на решение такой проблемы, так как модели, созданные на этапе проектирования могут быть непосредственно использованы в при разработке. Безусловно, в чистом виде использование MDD не дает таких преимуществ, так как модели получаются очень детальными и не понятны всем заинтересованным сторонам. Поэтому в качестве наполнения модельных вершин был использован псевдокодовый язык, снимающий эти ограничения.

Целью эксперимента является проверка следующей гипотезы:

H1. Применение средства концептуально-алгоритмической поддержки решения задач позволяет рационализировать использование человеческих ресурсов и избежать противоречия документации и системы (кода).

Необходимо описать допущения, на которых будет базироваться эксперимент:

- В качестве метода для оценки затраченного времени будет использоваться методология GOMS;
- Оценка времени потраченное на документирование и реализацию алгоритма будет ограничено только анализом времени потраченного на создание диаграммы активностей и написание псевдокода;
- Допускается, что псевдокод обладает достаточным уровнем абстракции для понимания всеми заинтересованными сторонами и может быть использован для наполнения графической модели.

В качестве входных параметров выступает задача интеграции SWI-Prolog в среду WIQA, а именно реализация компонента «Swi-Prolog proxy», рассматриваемого в параграфе «Модуль перевода и загрузки пролог подобных форм в язык PROLOG». Задача интеграции будет выполнена 2 различными способами:

1. Непосредственно путем реализации алгоритма на языке псевдокода, и документирования его в редакторе диаграмм по ISO/IEC 19505-2:2012.
2. Создание в редакторе диаграмм модели по ISO/IEC 19505-2:2012 и перевод модели в псевдокодовое представление.

В качестве выходных параметров будет выступать оценка затраченного времени согласно GOMS для каждого из подходов. В качестве плана проведения эксперимента выступает следующая методика:

1. Решить задачу интеграции среды SWI-Prolog в среду WIQA первым способом.

   a. Написать код на псевдокоде, решающий поставленную задачу.
   b. Создать диаграмму активностей для реализованного алгоритма.
2. Решить задачу интеграции среды SWI-Prolog в среду WIQA вторым способом.
   a. Создать диаграмму активностей, в качестве наполнения вершин использовать псевдокод.
   b. Перевести полученную диаграмму в псевдокодовое представление в автоматическом режиме.
3. Оценить время с помощью GOMS, затраченное в первом случае.
4. Оценить время с помощью GOMS, затраченное во втором случае.
5. Сравнить результаты, сделать выводы.

Правила расчета затраченного времени по модели GOMS

Модель GOMS основана на том принципе, что время, необходимое для выполнения любой задачи в системе «пользователь-компьютер», представляет собой сумму времен для различных жестов (нажатие клавиши, передвижение мыши и т.д.). Для оценки вводят следующие параметры:

- K – время, необходимое для нажатия клавиши. Примерная оценка 0,2с.
- P – время, необходимое для того чтобы указать на определенную позицию на экране монитора. Примерная оценка 1,1с.
- H – время, необходимое для перемещения руки с мыши на клавиатуру или обратно. Примерная оценка 0,4с.
- B – время, затраченное на нажатие кнопки мыши. Примерная оценка 0,1с.
- M – ментальная подготовка. Время подготовки к следующему шагу. Примерная оценка 1,2с.
- R – задержка компьютера на для ответа. Может варьироваться.

Вводятся также ряд правил, необходимых для модификации последовательности действий:

- Оператор M необходимо ставить перед всеми операторами K и P, которые обозначают выбор команды. Если K и P являются аргументами команды, то M не ставится.
- Если после M следует ожидаемое с точки зрения оператора действие, то M в данном случае не ставится.
- При наборе непрерывных последовательностей следует оставить только оператор M в начале (например, набора текста на клавиатуре).
- Если оператор M перекрывается оператором R, то оператор M можно опустить.

Алгоритм, рассматриваемый в эксперименте, отличается от программной реализации (несколько упрощен) и реализован на языке псевдокода (рисунке 4.18). На рисунке 4.19 представлен псевдокод, который получился в результате ручного кодирования.

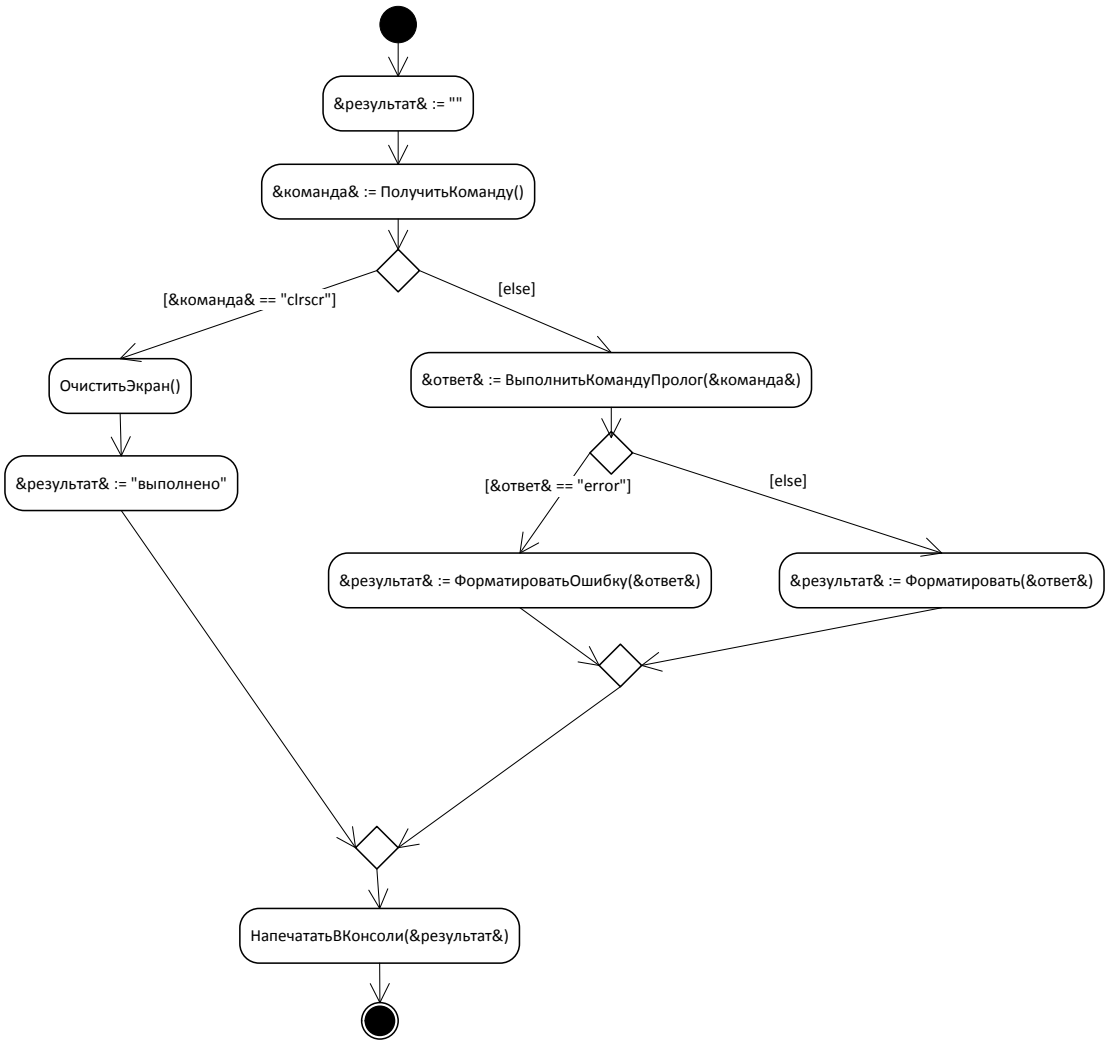

Рисунок 4. 18 - Диаграмма активностей функции исполнения

```
&результат& := "";
&команда& := ПолучитьКоманду();
IF &команда& == "clrscr" THEN
        ОчиститьЭкран();
        &результат& := "выполнено";
ELSE
        &ответ& := ВыполнитьКомандуПролог(&команда&);
        IF &ответ& == "error" THEN
                &результат& := ФорматироватьОшибку(&ответ&);
        ELSE
                &результат& := Форматировать(&ответ&);
        END
END
НапечататьВКонсоли(&результат&);
```

Рисунок 4. 19 - Псевдокод, полученный после ручного кодирования

Для первого процесса время будет складываться из следующих составляющих:

1. Время, затраченное на создание псевдокода в вопросно-ответном протоколе
    a. Всего было набрано 335 символов, состоящих из 14 когнитивных блоков (операторов). Для создания нового оператора сначала надо переместить мышь на кнопку добавление вершины, а затем на созданную вершину и нажать левую кнопку мыши для начала редактирования. После завершения редактирования необходимо проделать операцию заново (переключившись с клавиатуры на мышь). Суммарное время вычислялось по формуле

$$14M + 335K + 14 * (2P + 2B + 2H)$$

2. Время, затраченное на переключение контекста. Необходимо дождаться запуска редактора диаграмм, выбрав соответствующий пункт меню. Логично здесь рассмотреть суммарное время R, необходимое для загрузки редактора и не рассматривать отдельно время, затраченное на перемещение курсора и нажатие кнопок мыши для его вызова. Среднее время запуска редактора R=5c.

3. Время, затраченное на создании диаграммы активностей в специализированном редакторе. Для создания 1 вершины необходимо выбрать вершину из палитры, перевести курсор на нее, нажать левой кнопкой мыши, перевести курсор на поле редактирования и нажать левую кнопку мыши. При выборе связей ситуация аналогичная, за исключением отсутствия М (так как связь не имеет типов) и наличие дополнительного клика (для задания связи необходимо произвести щелчок по обеим вершинам). При редактировании текстов вершины необходимо набрать меньшее количество символов, так как часть операторов (например, условие) генерируется на основе используемых типов вершин.
    a. Созданная диаграмма имеет 14 вершин и 15 связей, было набрано порядка 290 символов. Суммарное время можно вычислить по формуле

$$14 * (\mathrm{M} + 2\mathrm{P} + 2\mathrm{B} + 2\mathrm{H}) + 15 * (2\mathrm{P} + 3\mathrm{B}) + 290\mathrm{K}$$

Для второго процесса время будет складываться из следующих составляющих:

1. Время, затраченное на создание диаграммы активностей в специализированном редакторе
    a. Это время эквивалентно времени из п. 3. в предыдущем процессе.
2. Время, необходимое для перевода созданной диаграммы в ее псевдокодовое представление (в вопросно-ответном протоколе)
    a. Является величиной, зависящей от количества вершин и связей на диаграмме. При тестировании автоматического перевода было выявлено, что среднее время для перевода диаграмм порядка 20 вершин (количество связей варьировалось, в зависимости от используемых операторов) составляет порядка 5 – 10 секунд. То есть для данного случая примем R=7с.

Поставив соответствующие значения для приведенных выше формул, получили следующие результаты

1. Первый процесс (написание кода в QA-протоколе и создание диаграммы в специализированном редакторе)
    a. Время, затраченное на создание псевдокода в вопросно-ответном протоколе примерно равно 130с.
    b. Время, затраченное на переключение контекста примерно равно 5с.
    c. Время, затраченное на создании диаграммы активностей в специализированном редакторе примерно равно 160с.
    d. Суммарное затраченное время примерно равно 295с.
2. Второй процесс (создание диаграммы в специализированном редакторе и автоматический перевод в QA-протокол)
    a. Время, затраченное на создание диаграммы активностей в специализированном редакторе - 160с.
    b. Время необходимое для перевода созданной диаграммы в ее псевдокодовое представление – 7с.
    c. Суммарное время 167с

На основе проведенных измерений можно сделать следующие выводы:

1. Применение средства образно-алгоритмической поддержки решения задач позволяет рационализировать использование человеческих ресурсов и избежать противоречия документации и системы, что подтверждает Н1. Выигрыш по времени составляет примерно 40 %
2. При отсутствии необходимости документирования (наличия моделей) текущий подход проигрывает классическому подходу решения задач через программирования на псевдокоде (проигрыш составляет порядка 20 %)

Оценка эффективности средства поддержки декларативного представления

Как отмечалось выше, декларативное представление необходимо для формирования некоторых знаний в предметной области разрабатываемого ПО, а также для выявления различного рода противоречий, неполноты знаний и т.д. С другой стороны, согласованное графическое представление позволяет вовлекать правое (отвечающие за восприятий образов). Представление о предметной области может формироваться не только через формирования списка предикатов (единиц знания), но путем построения различных визуальных моделей, которые рассматривались в главе 1.

Обоснование выбора задачи для проведения эксперимента

Для демонстрации применимости предложенных решений необходимо рассмотреть задачу, имеющую следующие характеристики

- Предметная область задачи должна содержать большое количество разнородных фактов и правил, которые связаны друг с другом. Это обусловлено тем, что с таким видом задач (содержащих большое количество разнородных фактов) приходится часто сталкиваться при разработке SIS систем. В качестве такой задачи могла бы выступить логическая задача, предложенная Эйнштейном [13], так как она обладает таким свойством.
- Задача должна быть связана с предметной областью САПР или со смежной.
- В задаче должна быть заложена некоторая перспектива развития или изменения требований (фактов предметной области).
- Предметная область задачи (совокупность фактов) может быть описана с помощью древовидной диаграммы.

Проанализировав большое количество задач, было принято решение выбрать ГОСТ Р ИСО/МЭК 9126-93 [81] в качестве основы для формулирования задачи по следующим причинам:

- Описывает довольно большое количество фактов (метрик и характеристик). Описывает отношения для 6 показателей-характеристик и 21 атрибутов, использующихся для оценки качества ПО.
- В “Приложении А” к ГОСТ Р ИСО/МЭК 9126-93 [81] отмечается, что перечень описанных атрибутов пока не позволяет их включить в стандарт, так как они могут быть изменены (перспектива развития), но рекомендованы для практического использования.
- Представленные характеристики и атрибуты могут быть представлены с помощью простой древовидной модели (как будет показано далее).

Задача для эксперимента может быть сформулирована следующим образом: «Представить отношения между характеристиками и атрибутами качества ПО, которые описаны в ГОСТ Р ИСО/МЭК 9126-93 [81] в виде удобным для восприятия (графическая диаграмма) с целью формирования целостного знания о предметной области».

Ниже будет проведен эксперимент, целью которого является доказательство следующих гипотез:

Н1. Применение средства декларативной поддержки решения задач позволяет рационализировать (сократить) использование человеческих ресурсов.

Н2. Применение средства декларативной поддержки решения задач ориентировано на развитие (расширение) знаний о предметной области, т.е. позволяет вводить новые правила и факты и извлекать новые знания на их основе.

Н3. Автоматическая генерация визуальной модели из декларативного представления работает корректно и выполняется за приемлемое время.

В качестве входных параметров для проведения эксперимента выступает задача представления знаний о характеристиках и метриках оценки качества программного обеспечения, представленных в ГОСТ Р ИСО/МЭК 9126-93 и ГОСТ 28195-89 [94, 81]. В качестве выходных параметров будут выступать

оценки затраченного времени (по методологии GOMS) для построения диаграммы, описывающую предметную область. В качестве плана эксперимента выступает следующая методика:

1. Представление отношений между характеристиками и атрибутами для оценки качества ПО, согласно ГОСТ Р ИСО/МЭК 9126-93 [81] с помощью древовидной диаграммы, описанной в главе 1. Данный вид диаграмм нашел широкое применение в различных областях знаний и позволяет наглядно представлять отношения между различными сущностями. В качестве редактора будет использован специализированный графический редактор, представленный в диссертационном исследовании.
    a. Оценить время, затраченное на построение диаграммы в специализированном редакторе согласно методике GOMS
2. Представление отношений между характеристиками и атрибутами для оценки качества ПО, согласно ГОСТ Р ИСО/МЭК 9126-93 [81] в разработанном средстве декларативной поддержки решения задач.
    a. Представить факты о предметной области в виде списка предикатов в QA-протоколе (текстовое представление) и оценить время, потраченное на их создание.
    b. Перевести полученную декларативную форму в согласованное с ней графическое представление (близкое по семантике к древовидной диаграмме) и оценить корректность произведенных преобразований и время работы.
3. Оценить потраченное на создание моделей время, сделать выводы относительно предложенных решений.

Алгоритм оценивания времени (и его численные характеристики) были представлены в этой главе ранее. Здесь будут рассмотрены только итоговые значения. Как было отмечено ранее, для создания 1 вершины в специализированном графическом редакторе необходимо выполнить операции:

$М + 2Р + 2В + 2Н$. Для создания связи между двумя вершинами необходимо выполнить: $2Р + 3В$. Таким образом, для создания древовидной диаграммы, состоящей из 28 вершин, 27 связей и 470 символов текста, приведенной на рисунке 4.20 необходимо затратить примерно 285 секунд:

$$28 * (М + 2Р + 2В + 2Н) + 27 * (2Р + 3В) + 470К = 285с$$

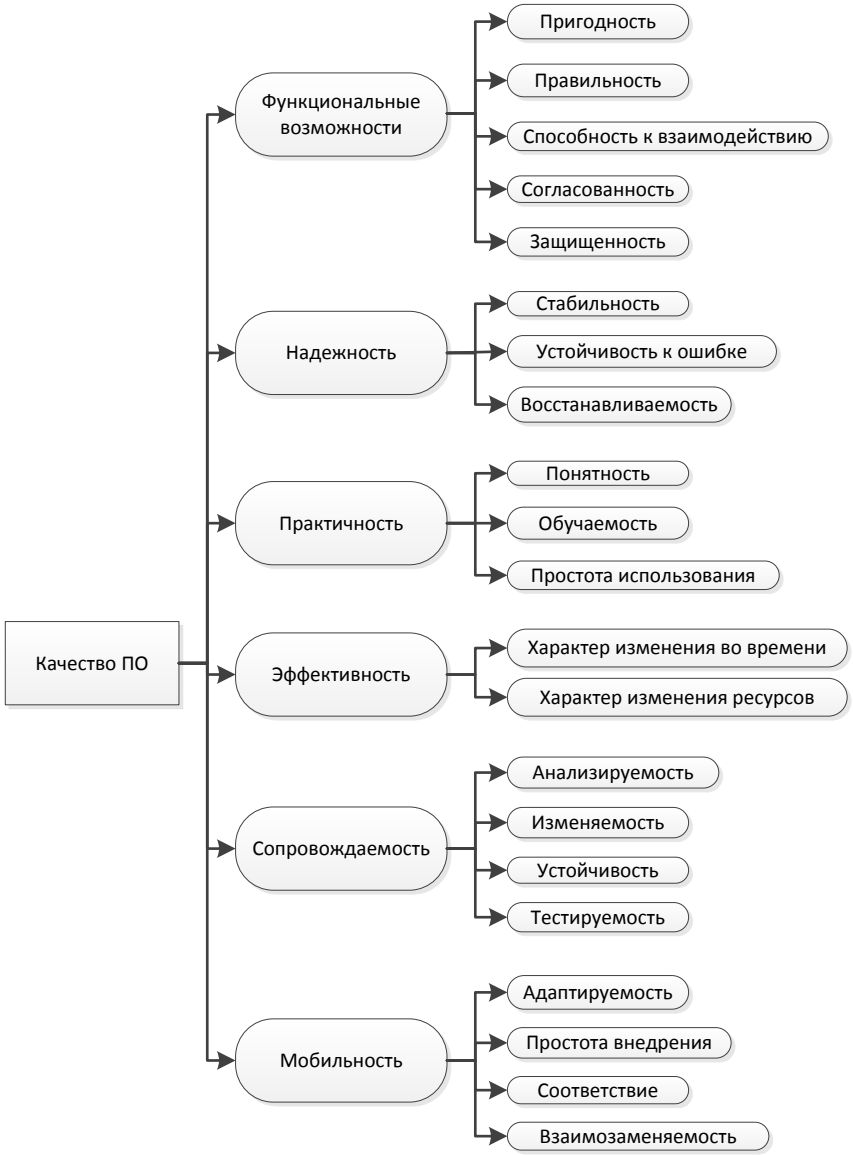

Рисунок 4. 20 - Древовидная диаграмма для представления отношений между характеристиками и атрибутами для оценки качества ПО

В предыдущем эксперименте уже оценивалось время, которое необходимо затратить на создание одной псевдокодовой единицы в вопросно-ответном протоколе. По приведенной формуле можно сделать вывод, что время,

затраченное на создание декларативного представление, примерно равняется 316 секундам.

$$27\text{М} + 988\text{К} + 27 * (2\text{Р} + 2\text{В} + 2\text{Н}) = 316\text{с}$$

На рисунке 4.21 представлено декларативное представление для задачи представления отношений между характеристиками и атрибутами качества ПО в двух представлениях (в виде семантической граф схемы и текстовой проекции):

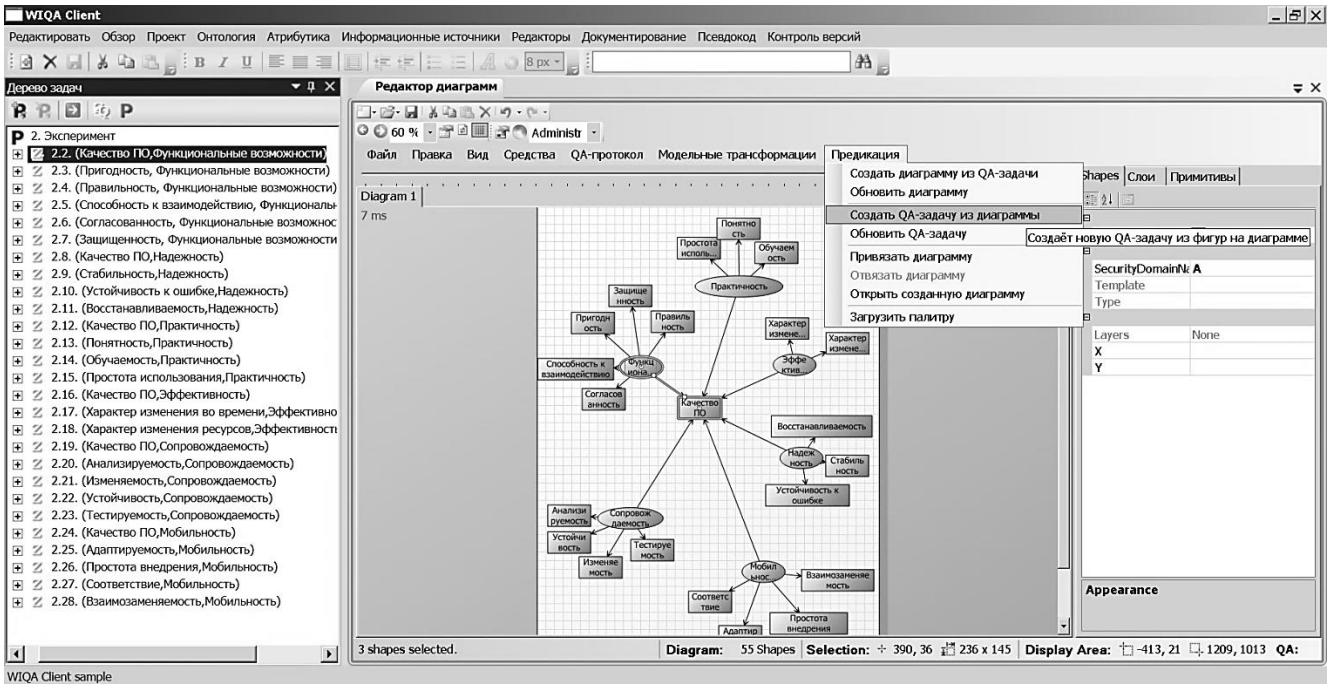

Рисунок 4. 21 - Декларативное представление для задачи представления отношений между характеристиками и атрибутами качества ПО

Согласно полученным данным время, затраченное на создание декларативного представления и автоматический перевод его в диаграмму, превышает время затраченное на создание той же диаграммы в специализированном редакторе. Рассмотрим причины получения именно такого результата и попытаемся выделить область, где второй способ является более эффективным по сравнению с первым. Сделаем следующую гипотезу

Н4. Причина проигрыша автоматической генерации семантической граф схемы ручному построению древовидной диаграммы, заключается в большой длине названий характеристик и может быть устранена при сокращении длины последних.

Если рассмотреть процесс построения более детально, то можно заметить, что в первом случае происходит сначала построение вершин, наполнение их текстом, а затем проставление связей между ними. В декларативных описаниях отсутствует определение вершины как такового. Там происходит описание связи, из-за чего приходится дублировать названия вершины, из которой выходит связь. Во многих системах генерации графа по текстовому описанию, таких как Graphviz, для устранения этого недостатка, вершине присваивается короткое имя, которое потом используется для представления связи. Рассмотрим общий вариант для выявления зависимости между длиной текста и временем, затраченным на создание диаграммы для обоих способов. Ниже будет рассмотрен упрощенный вариант диаграммы, состоящий из 2 уровней, причем первый уровень состоит из 1 вершины, из которой выходят связи ко всем остальным.

Формула для вычисления времени по 1 способу имеет вид:

$$M + 2P + 2B + 2H + KL_o + (N-1) * (M + 2P + 2B + 2H + KL_i) + (N-1) * (2P + 3B))$$

По 2 способу:

$$(N-1) * (M + 2P + 2B + 2H + KL_o + KL_i + 3K)$$

Где:

N – количество вершин в диаграмме

L_o – длина текста для вершины, из которой выходят связи

L_i – длина текста у вершин, в которые входит связь

Формулы получились путем замены количества вершин, связей и длины текста на переменные значения. Во 2 формуле присутствует учет дополнительных 3 символов, которые представляют собой скобки и символ запятую (формат записи декларативного представления требует наличия этих символов). Зафиксируем значение L_o и N и построим график отношения затраченного времени по первому способу к затраченному времени по второму (таблица 4.4).

Таблица 4.4 - Отношение затраченного времени для двух способов

L_i	L_o	N	Время 1 способа, t1	Время 2 способа, t2	Отношение времени первого способа ко второму

20	10	28	248,7	297	0,837374
19	10	28	248,5	291,6	0,852195
18	10	28	248,3	286,2	0,867575
17	10	28	248,1	280,8	0,883547
16	10	28	247,9	275,4	0,900145
15	10	28	247,7	270	0,917407
14	10	28	247,5	264,6	0,935374
13	10	28	247,3	259,2	0,95409
12	10	28	247,1	253,8	0,973601
11	10	28	246,9	248,4	0,993961
10	10	28	246,7	243	1,015226
9	10	28	246,5	237,6	1,037458
8	10	28	246,3	232,2	1,060724
7	10	28	246,1	226,8	1,085097
6	10	28	245,9	221,4	1,110659
5	10	28	245,7	216	1,1375
4	10	28	245,5	210,6	1,165717
3	10	28	245,3	205,2	1,195419
2	10	28	245,1	199,8	1,226727
1	10	28	244,9	194,4	1,259774

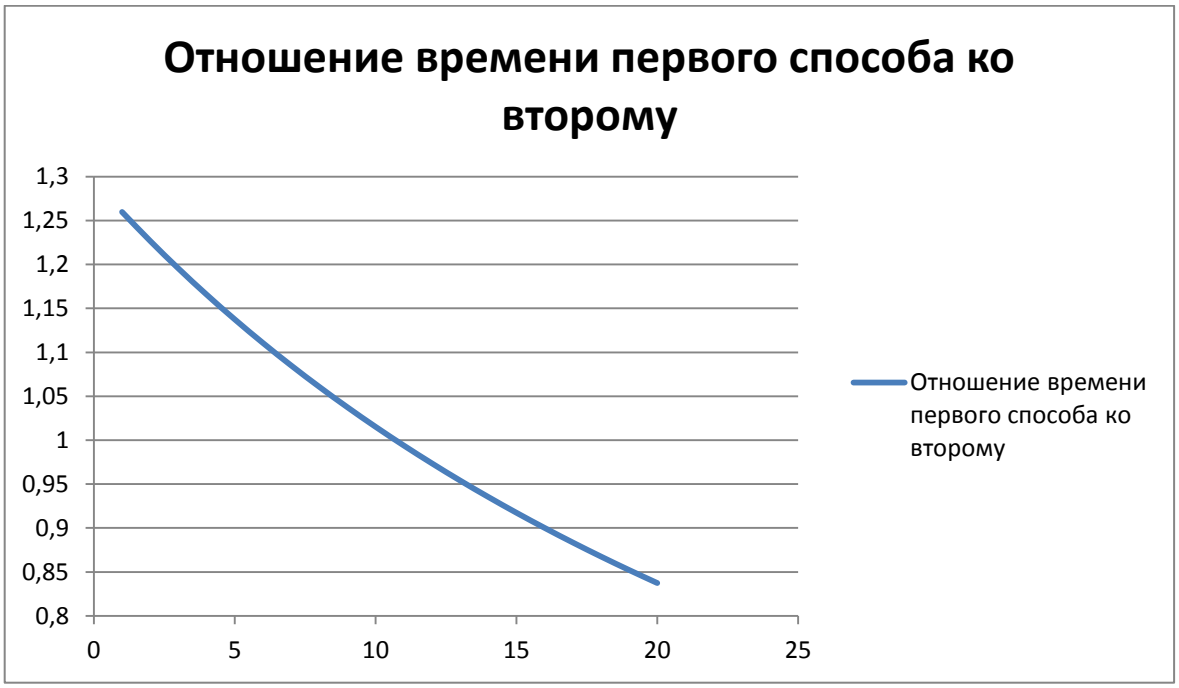

Рисунок 4. 22 - Зависимость эффективности метода (отношение затраченного времени) от длины текста (ось X) в вершины, из которой исходят связи

Из рисунка 4.22 при L_i = 11 символов оба способа показывают одинаковую эффективность (отношение примерно 1). При увеличении количества смыслов второй подход начинает проигрывать и при L_i = 20 отстает от первого подхода на

15 %. Если двигаться в обратную сторону, то второй способ дает выиграш по времени, который доходит до 26 % при $L_i = 1$. Таким образом гипотезы H1 и H4 являются доказанными, т.е. можно сделать вывод, что текущая реализация декларативного представления показывает свою эффективность при средней длине текста в вершинах менее 10 символов. Этот показатель может быть улучшен, путем отдельного описания вершин и присвоения каждой однобуквенного идентификатора (как делается в Graphviz). За время проведения эксперимента не возникло ни одной ошибки, следовательно гипотеза H3 о корректности работы разработанного средства подтверждена.

Рассмотрим поведение обоих методов при изменении фактов в предметной области. При значительном увеличении фактов первый метод будет показывать неэффективность, так как количество связей будет расти пропорционально количеству вершин. Если исходить из принципа, что человеческий мозг может удерживать одновременно около 7 объектов в памяти, то этот придел понимания будет, достигнут достаточно быстро. Надо отметить, что преимущество древовидных (иерархических) структур частично решает эту проблему, так введение иерархии, позволяет концентрироваться только на определенных узлах и не обозревать диаграмму целиком. Второй подход позволяет решить эту проблему, так как декларативное представление может быть автоматически переведено в пролог подобную форму и исполнено на интерпретаторе, т.е. представлять собой код. Таким образом, проверка гипотез (и выявления зависимостей между фактами) сводиться к “заданию вопросов” интерпретатору ПРОЛОГ, т.е. можно говорить, что предложенное средство ориентировано на развитие (увеличения) предметной области, что подтверждает гипотезу H2.

4.5. Выводы

1. В основе специализированного графического редактора лежит компонент NShape, использующийся для разработки enterprise решений, который был выбран из большого количества альтернатив из-за его расширяемости и быстродействия на больших моделях.

2. Реализованная поддержка изобразительного типа обеспечивает возможность визуальной поддержки мысленного воображения, и «усиливает» его эффекты в прецедентно-ориентированном решении проектных задач.

3. Реализация поддержки декларативного типа ориентирована на акты понимания в формировании постановки задачи и логическую проверку в интерпретаторе ПРОЛОГ с возможностью выходу на словарь онтологий.

4. Реализованный концептуально–алгоритмический тип позволяет представлять решение задачи в виде диаграмм активностей, диаграмм вариантов использования и диаграмм классов с возможность выход на интерпретатор псевдокода для проведения концептуальных экспериментов.

5. Разработанный компонент контроля версий, базирующийся на системе контроля версий GIT, учитывает специфику типизированного набора моделей и позволяет эффективным образом осуществлять фиксацию этапов решения и переключения между ними из среды WIQA.

6. Разработанные программные средства позволили применить на практике предложенные модели и оценить их эффективность на уровне программных решений в сравнении с моделью GOMS (Goals, Operators, Methods and Selection rules) для концептуально-алгоритмического и декларативного представлений с учетом введённых допущений и показали эффективность порядка 20%.

Заключение

Цель диссертационной работы – повышение эффективности деятельности проектировщика в системах автоматизированного проектирования за счет использования средств поддержки решения задач – достигнута.

Основными результатами работы являются:

1. Типизированный набор образно-семантических моделей и система их согласованных преобразований, отличающихся тем, что для каждого типа моделей используется композиция проекций, ориентированных на программную интерпретацию.
2. Метод понятийно-образной поддержки процесса пошаговой детализации в прецедентно-ориентированном решении проектных задач, использующий конструктивное и управляемое включение автоматизированного концептуального экспериментирования и моделирования.
3. Метод итеративного согласования понятийного и образного содержания текстовых единиц (предложений текста постановки задачи) с использованием их преобразования в прологоподобную форму, специфику которого определяет взаимодополняющее итеративное уточнение графического и текстового представления требований и спецификаций с использованием автоматического взаимодействия с онтологией.

Предложенные модели, методы и средства реализованы в виде комплекса инструментально-технологических средств, обслуживающих авторский подход к прецедентно-ориентированному решению профессиональных задач с использованием образно-семантической поддержки, способствующей конструктивному и управляемому включению в процессы решения мысленного воображения и повышающий эффективность человеко-компьютерного взаимодействия.

Список литературы

1. A periodic table of visualization methods [Электронный ресурс] - Режим доступа: http://www.visual-literacy.org/periodic_table/periodic_table.html (дата обращения: 23.10.2016)

2. A. Cockburn. Agile Software Development: The Cooperative Game / Alistair Cockburn // 2nd edition, October 2006, Addison-Wesley Professional

3. Activiti 5.14 User Guide [Электронный ресурс] - Режим доступа: http://www.activiti.org/userguide/index.html (дата обращения: 12.06.2015)

4. Aharoni E., Vincent G. M., Harenski C. L. et al. Neuroprediction of future rearrest (англ.) // Proceedings of the National Academy of Sciences. — 2013. — Vol. 110, no. 15. — P. 6223—6228. — ISSN 0027-8424. — DOI:10.1073/pnas.1219302110. — PMID 23536303.

5. Alfresco (ECM-система) [Электронный ресурс] - Режим доступа: https://www.alfresco.com/ (дата обращения: 22.08.2015)

6. Andrew Forward and Timothy C. Lethbridge. Problems and Opportunities for Model-Centric Versus Code-Centric Development: a Survey of Software Professionals. In Proceedings of the International Workshop on Models in Software Engineering ICSE' 08, pages 27– 32, New York, 2008

7. Batra D. Conceptual data modelling in theory and practice [Электронный ресурс] / D. Batra, G. Marakas // European Journal on Information Systems - Режим доступа: http://search.proquest.com/openview/700900bd037d540ed0deaffbd2e42a95/1.pdf?pq-origsite=gscholar (дата обращения: 9.01.2015)

8. Beydeda, S., Book, M., Gruhn, V. (Eds.). Model-Driven Software Development. Springer-Verlag, Heidelberg, 2005, 464 c., ISBN 978-3-540-25613-7

9. Brian Dobing and Jeffrey Parsons. How UML is used. Commun. ACM, 49: 109– 113, 2006

10. Domain Expert DSLs [Электронный ресурс] - Режим доступа: http://www.infoq.com/presentations/DSL-Magnus-Christerson-Henk-Kolk (дата обращения: 19.06.2016)

11. Drools - Business Rules Management System [Электронный ресурс] - Режим доступа: https://www.drools.org/ (дата обращения: 12.06.2015)

12. Drools Expert [Электронный ресурс] - Режим доступа: http://www.jboss.org/drools/drools-expert.html (дата обращения: 12.06.2015)

13. Einstein's Puzzle [Электронный ресурс] - Режим доступа: https://web.stanford.edu/~laurik/fsmbook/examples/Einstein'sPuzzle.html (дата обращения: 03.11.2015)

14. Fowler M. IntentionalSoftware [Электронный ресурс] / M. Fowler - Режим доступа: http://martinfowler.com/bliki/IntentionalSoftware.html (дата обращения: 19.06.2016)

15. France R., Rumpe B. Model-driven Development of Complex Software: A Research Roadmap. Proc. FOSE '07 2007 Future of Software Engineering. Washington, 2007, с. 37-54.

16. Friedrich Steimann and Thomas Kühne. Coding for the Code. Queue, 3: 44–51, 2005.

17. G. Booch, J. Rumbaugh, I. Jacobson. The Unified Modeling Language User Guide, Second Edition // AddisonWesley Professional, 2005. 496p

18. Haan Johan. 15 reasons why you should start using Model Driven Development [Электронный ресурс] - 2009 - Режим доступа: http://www.theenterprisearchitect.eu/archive/2009/11/25/15-reasons-why-you-should-start-using-model-driven-development (дата обращения: 7.01.2015).

19. Human Tasks in jBPM 5 [Электронный ресурс] - Режим доступа: https://docs.jboss.org/jbpm/v5.2/userguide/ch.Human_Tasks.html (дата обращения: 02.02.2016)

20. Information technology - Object Management Group Unified Modeling Language (OMG UML), Infrastructure. ISO/IEC 19505-1:2012(E) [Электронный ресурс] - Режим доступа: http://www.omg.org/spec/UML/ISO/19505-1/PDF (дата обращения: 28.10.2015)

21. Install eclipse jBPM [Электронный ресурс] - Режим доступа: http://people.redhat.com/kverlaen/install-eclipse-jbpm.swf_____(дата обращения: 02.02.2016)

22. Intentional Platform [Электронный ресурс] - Режим доступа: http://www.intentsoft.com/intentional-technology/intentional-platform (дата обращения: 19.06.2016)

23. Jacobson I. Methods Need Theory [Электронный ресурс] / Jacobson I, Meyer B – 2009 - Режим доступа: http://www.drdobbs.com/architecture-and-design/methods-need-theory/219100242 (дата обращения: 9.01.2015)

24. Jboss [Электронный ресурс] - Режим доступа: http://www.jboss.org/ overview/ (дата обращения: 22.08.2015)

25. jBPM [Электронный ресурс] - Режим доступа: http://www.jboss.org/jbpm (дата обращения: 11.02.2015)

26. Astle, D. E.; Scerif, G. (2011). Interactions between attention and visual short-term memory (VSTM): What can be learnt from individual and developmental differences?. Neuropsychologia. 49 (6): 1435–1445. doi:10.1016/j.neuropsychologia.2010.12.001. PMID 21185321.

27. Karl Duncker and Cognitive Science [Электронный ресурс] - Режим доступа: http://digitalcollections.library.cmu.edu/awweb/awarchive?type=file&item=47038 (дата обращения: 19.06.2015)

28. Lars Grunske, Leif Geiger, Albert Zündorf. Using Graph Transformation for Practical Model-Driven Software Engineering p. 91-118

29. Manoli Albert, Jordi Cabot, Cristina Gómez, and Vicente Pelechano. Automatic generation of basic behavior schemas from UML class diagrams. Software and System Modeling, 9(1). 2010. – С. 47– 67

30. Mens T. Taxonomy of Model Transformations. Journal Electronic Notes in Theoretical Computer Science (ENTCS) archive Volume 152, March, 2006 Pages 125-142

31. Meta [Электронный ресурс] - Режим доступа: http://www.intentsoft.com intentional-technology/meta/ (дата обращения: 19.06.2016)

32. Milner R. Communicating and Mobile Systems: The Pi Calculus / R. Milner - Cambridge University Press - 1 edition (June 13, 1999) - 174 c.

33. Milner R. The poliadic-calculus: a Tutorial / R. Milner // University of Edinburgh - 1991.

34. Moulton, S. T.; Kosslyn, St. M.: 2009. Imagining predictions: mental imagery as mental emulation Phil. Trans. R. Soc. B 2009, 364, pp. 1273-1280.

35. MyExperiment Blast Align and Tree [Электронный ресурс] - Режим доступа: http://www.myexperiment.org/workflows/3369.html (дата обращения: 02.06.2015)

36. Navarro-Prieto P.; Cañas, J. J.: 2001. Are visual programming languages better? The role of imagery in program comprehension, International Journal of Human-Computer Studies, Volume 54, Issue 6, pp. 799-829

37. Oracle Fusion Middleware Developer’s Guide for Oracle SOA Suite 11g Release 1 (11.1.1) E10224-01 [Электронный ресурс] - Режим доступа: http://download.oracle.com/otndocs/products/soa/e10224.pdf (дата обращения: 17.05.2016)

38. Patterns: Service-Oriented Architecture and Web Services [Электронный ресурс] - Режим доступа: http://www.redbooks.ibm.com/redbooks/pdfs/sg246303.pdf (дата обращения: 12.04.2014)

39. Paul Hudak. Domain Specific Languages [Электронный ресурс] - Режим доступа: http://cs448h.stanford.edu/DSEL-Little.pdf (дата обращения: 12.06.2015)

40. Peter F. MacNeilage, Lesley J. Evolutionary Origins of Your Right and Left Brain // Scientific American Magazine - June 24, 2009 [Электронный ресурс] - Режим доступа: https://www.researchgate.net/file.PostFileLoader.html?id=5328c41dd11b8bab218b463a&assetKey=AS%3A272156582187019%401441898594918 (дата обращения: 18.12.2016)

41. Petre, M.: 2010. Mental imagery and software visualization in high-performance software development teams. In J. Vis. Lang. Comput., 21 (3), pp. 171-183.

42. Petre, M.; Blackwell, A. F.: 1999. Mental imagery in program design and visual programming. Int. J. Hum.-Comput. Stud. 51 (1), pp. 7-30.

43. Ponirakis A. L. Improving Software Efficiency: Automated Program Transformation for Reclaiming Execution Efficiency [Электронный ресурс] / A. L. Ponirakis - Режим доступа: http://www.navysbir.com/n13_1/N131-061.htm (дата обращения: 20.12.2015)

44. Ponirakis A. L. Improving Software Efficiency: Automated Program Transformation for Reclaiming Execution Efficiency [Электронный ресурс] / A. L. Ponirakis // Navysbir. - 2012. - Режим доступа: http://www.navysbir.com/n13_1/N131-061.htm. (дата обращения: 14.05.2014)

45. Posner, M. I.; Petersen, S. E. (1990). "The attention system of the human brain". Annual Review of Neuroscience. 13: 25–42. doi:10.1146/annurev.ne.13.030190.000325. PMID 2183676.

46. Public Intentional Demo [Электронный ресурс] - Режим доступа: http://www.intentsoft.com/dsl-devcon/ (дата обращения: 19.06.2016)

47. Puhlmann, F., Weske, M. Using the Pi-Calculus for Formalizing Workflow Patterns. In van der Aalst, W., Benatallah, B., Casati, F., eds.: Proceedings of the 3rd International Conference on Business Process Management, volume 3649 of LNCS, Berlin, Springer-Verlag (2005) с. 153–168

48. Pulvermüller F. How neurons make meaning: brain mechanisms for embodied and abstract-symbolic semantics [Электронный ресурс] / F. Pulvermüller — Режим доступа: http://www.cell.com/trends/cognitive-sciences/fulltext/S1364-6613(13)00122-8 (дата обращения: 18.12.2016)

49. Pylyshyn, Z.: 2002. Mental Imagery: In Search of a Theory. Behavioral and Brain Sciences, 25 (2), pp. 157-237.

50. Rabbi F., MacCaull W. Model Driven Workflow Development with T □. Proc. CAiSE 2012 International Workshops. Gdańsk, Poland, 2012, c. 25-26.

51. Reisberg, D.: 2013. Mental Images, The Oxford Handbook of Cognitive Psychology, D. Reisberg (ed.), http://www.oxfordhandbooks.com/view/10.1093/oxfordhb/ 97801953767/46.001.0001/oxfordhb-9780195376746-e-25

52. Revisiting the Left-Brain/Right-Brain Dialogue [Электронный ресурс] - Режим доступа:https://class.uark.edu/_resources/downloads/self/leftbrainrightbrain.pdf (дата обращения: 18.12.2016)

53. S. W. Ambler, J. Nalbone, M. Vizdos. The Enterprise Unified Process: Extending the Rational Unified Process // Prentice Hall PTR, February 2005

54. Sangiorgi, D. A Theory of Bisimulation for the Pi-Calculus. In: CONCUR ’93: Proceedings of the 4th International Conference on Concurrency Theory, Berlin, Springer-Verlag (1993) c. 127–142

55. Sheth A. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure [Электронный ресурс] / A. Sheth, D. Georgakopoulos, M. Ornick // Scientific Workflow Management and the Kepler System - Режим доступа: http://users.sdsc.edu/~ludaesch/Paper/kepler-swf.pdf_(дата обращения: 20.12.2015)

56. Simonyi C. Representing Software using a Domain Workbench (is software same as programming?) [Электронный ресурс] / C. Simonyi - 2008 - Режим доступа: http://web.stanford.edu/class/ee380/Abstracts/080206-slides.pdf (дата обращения: 19.06.2016)

57. Sosnin P. Role «Intellectual Processor» in Conceptual Designing Of Software Intensive Systems, ICCSA’2013 – the 11-th International conference on Computational Science and Applications, Part III, LNCS 7973 Springer, Heidelberg , 2013, c. 1–16.

58. Standish Group 2015 Chaos Report - Q&A with Jennifer Lynch [Электронный ресурс] - Режим доступа: https://www.infoq.com/articles/standish-chaos-2015 (дата обращения: 27.11.2016)

59. Stephen A. Introduction to BPMN [Электронный ресурс] / A. Stephen - Режим доступа: http://www.omg.org/bpmn/Documents/Introduction_to_BPMN.pdf (дата обращения: 07.04.2014).

60. Stolz M.C. Verification of workflow control-flow patterns with the SPIN model checker [Электронный ресурс] - Режим доступа: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.174.6971&rep=rep1&type=pdf (дата обращения: 14.04.2014)

61. SWI-Prolog 5.10. Reference Manual [Электронный ресурс] - Режим доступа: http://www.swi-prolog.org/download/stable/doc/SWI-Prolog-5.10.0.pdf (дата обращения: 03.07.2015)

62. T. R. G. Green, T. R. G.; Navarro, R.: 1995. Programming Plans, Imagery, and Visual Programming, Human—Computer Interaction, Springer US, pp. 139-144.

63. Thomas, N. J.T.: 2014. Mental Imagery, The Stanford Encyclopedia of Philosophy (Summer 2016 Edition), Edward N. Zalta (ed.), http://plato.stanford.edu/archives/sum2016/entries/mental-imagery/.

64. Tufte E. R. The Visual Display of Quantitative Information. – Graphics Press, 2001.

65. UML in Practice. [Электронный ресурс] - Режим доступа: http://neverworkintheory.org/2013/06/13/uml-in-practice-2.html (дата обращения: 03.07.2016)

66. VisiRule [Электронный ресурс] - Режим доступа: http://www.lpa.co.uk/vsr.htm (дата обращения: 23.10.2016)

67. W.M.P. van der Aalst. Verification of Workflow Nets. Proc. ICATPN '97 Proceedings of the 18th International Conference on Application and Theory of Petri Nets. London, 1997, pp. 407 - 426.

68. W.M.P. van der Aalst. Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques. Proc. Business Process Management, Models, Techniques, and Empirical Studies. London, 2000, pp. 161 - 183.

69. Wegbreit E., Suzuki S., Grabowecky M., Kounios J., Beeman M. Visual Attention Modulates Insight Versus Analytic Solving of Verbal Problems. http://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1127&context=jps 2011г.

70. What is an Influence Diagram? [Электронный ресурс] - Режим доступа: http://www.lumina.com/technology/influence-diagrams (дата обращения: 19.06.2015)

71. Why use workflows? [Электронный ресурс] - Режим доступа: http://www.taverna.org.uk/introduction/why-use-workflows/ (дата обращения: 02.06.2015)

72. Wieringa R. J., W.M.P. van der Aalst, Engels G., Brinksma H., etc. Semantics and Verification of UML Activity Diagrams or Workflow Modelling. Wierden, 2002, 230 p.

73. Workflow Patterns [Электронный ресурс] - Режим доступа: http://www.workflowpatterns.com/ (дата обращения: 20.03.2014).

74. Zhao Z. A Survey on Semantic Scientific Workflow / Z. Zhao, A Paschke [Электронный ресурс] - Режим доступа: http://semantic-web-journal.org/sites/default/files/swj275.pdf (дата обращения: 02.06.2015)

75. Автоматизированные системы. Требования к содержанию документов [Электронный ресурс] - Режим доступа: http://www.rugost.com/index.php?option =com_content&view=article&id=98:50-34698-90&catid=22&Itemid=53 (дата обращения: 28.03.2016)

76. Ассоциации [Электронный ресурс] - Режим доступа: http://psyznaiyka.net/view-pamat.html?id=associaciy (дата обращения: 09.10.2016)

77. Введение в Пи-Исчисление [Электронный ресурс]. — Режим доступа: http://randnet.files.wordpress.com/2011/04/pi-calculusrus.pdf (дата обращения: 20.12.2015).

78. Доброхотова Т. А., Брагина Н. Н. Функциональная асимметрия и психопатология очаговых поражений мозга, 1977

79. Дункер К. О процессе решения задач [Электронный ресурс] / К. Дункер, И. Кречевский - Режим доступа: http://vikent.ru/enc/1766/ (дата обращения: 19.06.2015)

80. Информатика, моделирование, автоматизация проектирования: сборник научных трудов / под ред. Н. Н. Войта. - Ульяновск : УлГТУ, 2010. - 531 с.

81. Информационная технология. Оценка программной продукции. Характеристики качества и руководства по их применению : ГОСТ Р ИСО/МЭК 9126-93 – Введ. 01.07.1994 – М., 1994

82. Как работает Oracle BPEL Process Manager? [Электронный ресурс] - Режим доступа: http://www.oracle.com/technetwork/ru/middleware/bpel/bpel-pm-exec-1532285-ru.html (дата обращения: 02.07.2016)

83. -Каленкова А. А. Автоматическая верификация и оптимизация потоков работ: Дис. канд. физ.-мат. наук. М., 2011. 164 с.

84. -Коробков К. Н. Исследование и разработка системы управления бизнес-процессами: Дис. канд. тех. наук. М., 2008. 195 с.

85. Лапшов, Юрий Александрович. Средства программно-картотечного управления потоками работ в коллективном проектировании автоматизированных систем : диссертация ... кандидата технических наук : 05.13.12 / Лапшов Юрий Александрович; [Место защиты: Ульян. гос. техн. ун-т]. - Ульяновск, 2015. - 246 с.

86. Лурия А.Р. Мозг человека и психические процессы. Том II. Нейропсихологический анализ сознательной деятельности. Москва, 1970

87. Лурия А.Р. Об историческом развитии познавательных процессов. М.: Наука, 1974.

88. Маторин С. И. Формализация моделей процессов на основе пи-исчисления [Электронный ресурс] / С. И. Маторин, М. В. Михелев - Режим

доступа: http://cyberleninka.ru/article/n/formalizatsiya-modeley-protsessov-na-osnove-pi-ischisleniya (дата обращения: 16.05.2015)

89. Михеев А. Война стандартов в мире workflow [Электронный ресурс] / А.Михеев, М. Орлов — Режим доступа: http://www.runawfe.org/rus/images/c/ce/Stat_ya2.pdf (дата обращения: 20.12.2015)

90. Михеев А., Орлов М. Перспективы WorkFlow-систем [Электронный ресурс] - Режим доступа: http://kak.znate.ru/pars_docs/refs/70/69920/69920.pdf (дата обращения: 20.03.2014).

91. О процессе решения задач [Электронный ресурс] - Режим доступа: http://www.metodolog.ru/00886/00886.html (дата обращения: 19.06.2015)

92. Обозначения программ и программных документов : ГОСТ 19.103-77 – Введ. 01.01.1980 – М., 1980

93. Особенности памяти как психологического процесса [Электронный ресурс] - Режим доступа: http://psixologiya.org/obshhaya/pamyat/1647-osobennosti-pamyati-kak-psixologicheskogo-proczessa-diplom.html (дата обращения: 09.10.2016)

94. Оценка качества программных средств. Общие положения : ГОСТ 28195-89 – Введ. 01.07.1990 – М., 1990

95. Пароджанов В. Д. Как улучшить работу ума: Алгоритмы без программистов — это очень просто! — М.: Дело, 2001. — 360с

96. Рыжиков Ю. И. Работа над диссертацией по техническим наукам. 3-е изд., перераб. и доп. СПб.; БХВ-Петербург, 2012 - 512 с

97. Система стандартов по информации, библиотечному и издательскому делу диссертация и автореферат диссертации. Структура и правила оформления : ГОСТ Р 7.0.11—2011. - Введ. 2012-09-01. – М., 2012.

98. Система стандартов по информации, библиотечному и издательскому делу. Библиографическая ссылка. Общие требования и правила составления : ГОСТ Р 7.0.5—2008. – Введ. 2009-01-01. – М., 2009.

99. Соснин П. И. Архитектурное моделирование автоматизированных систем. /П.И.Соснин. –Ульяновск: УлГТУ, 2007.– 146 с.

100. Соснин П. И. Вопросно-ответное программирование человеко-компьютерной деятельности / П. И. Соснин. – Ульяновск : УлГТУ, 2010. – 240 с.

101. Соснин П.И. Концептуальное моделирование компьютеризованных систем - Ульяновск : УлГТУ, 2008. – 198 с.

102. Соснин П.И. Онтологическая Поддержка Концептуального Экспериментирования в Вопросно-Ответных Моделирующих Средах / Труды Конгресса по интеллектуальным системам и информационным технологиям» : науч. издание в 4-х т. – М. : Физматлит, 2014. – Т. 1.-С. 488-495.

103. Схемы алгоритмов, программ, данных и систем : ГОСТ 19.701-90 – Введ. 01.01.1992 – М., 1992

104. Техническое задание на создание автоматизированной системы : ГОСТ 34.602-89 (Взамен ГОСТ 24.201-85) – Введ. 01.01.1990 – М., 1990

105. Фаулер М. UML. Основы, 3е издание. – Пер. с англ. – СПб: СимволПлюс, 2004. – 192 с., ил. ISBN 593286060X

Приложение 1. Акты внедрения

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное
образовательное учреждение высшего образования
**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(УлГТУ)**
Северный венец ул., д. 32,
г. Ульяновск, 432027, Россия
Тел.: (8422) 43-06-43; факс: (8422) 43-02-37
E-mail: rector@ulstu.ru http://www.ulstu.ru
ОКПО 02069378, ОГРН 1027301160226
ИНН/КПП 7325000052/732501001

УТВЕРЖДАЮ
Исполняющий обязанности
ректора УлГТУ
_____ А.П. Пинков
12 2016 г.

АКТ

о внедрении в учебный процесс

результатов диссертационной работы Галочкина М. В.

Результаты диссертационной работы Галочкина М.В. «Методы и средства образно-семантического сопровождения процессов решения проектных задач», представленной на соискание ученой степени кандидата технических наук, внедрены в учебный процесс Ульяновского государственного технического университета при обучении студентов направления «Информатика и вычислительная техника» по специальностям бакалавр техники и технологии 23010063, магистр техники и технологии 23010068 (лекции, лабораторные и практические занятия).

Заведующий кафедрой «Вычислительная техника»
ФГБОУ ВО
«Ульяновский государственный технический университет»
д.т.н., профессор

П.И. Соснин

АКТ
о внедрении в производственный процесс результатов диссертационной работы Галочкина М.В

Настоящим Актом удостоверяется, что результаты диссертационного исследования Галочкина М.В. на тему "Методы и средства образно-семантического сопровождения процессов решения проектных задач" а именно предложенный метод представления задачи в виде совокупности графических и текстовых моделей и метод пошагового уточнения этих моделей, а также реализация этих методов в виде программного комплекса были успешно применены в решении проектных задач при разработки новых версий библиотек Intel® Math Kernel Library и Intel® Data Analytics Acceleration Library.

Кузьмин Иван Юрьевич
Руководитель отдела численных методов Intel в России

Акционерное общество * ОГРН 1027700... * МОСКВА
Интел А/О

24.11.2016

АО «Интел А/О»
Россия, 121614, Москва
Бизнес Центр Крылатские Холмы,
ул. Крылатская, 17
Тел. +7 (495) 641-45-00
Факс +7 (495) 641-45-10

ОБЩЕСТВО С ОГРАНИЧЕННОЙ ОТВЕТСТВЕННОСТЬЮ **«ФБ Групп»**

143409, Московская область, г. Красногорск, ул. Успенская, д.5, помещение 13, офис 408-2
ОГРН: 1127746599276; ИНН/КПП: 7706779078/ 502401001; БИК: 044525593;
Банк: АО «Альфа- Банк»; Р/с: 40702810801300003079 тел. 8(499)506-74-24

30.09.2016 г.

АКТ

о внедрении в производственный процесс результатов диссертационной работы Галочкина М.В

Результаты диссертационной работы Галочкина М.В. «Методы и средства образно-семантического сопровождения процессов решения проектных задач», представленной на соискание ученой степени кандидата технических наук, внедрены в рабочий процесс ООО «ФБ Групп», где они используются для прототипирования проектных решений, что позволило сократить время решения, тем самым улучшив результативность, качество работы и финансовые показатели организации.

Директор технического департамента _____ Аверьянов С.В.
ООО «ФБ Групп»

УТВЕРЖДАЮ
Генеральный директор,
председатель НТС
ФНПЦ АО «НПО «Марс», к.т.н.
_____ В.А. Маклаев
«16» 12 2016 г.

АКТ

об использовании результатов кандидатской диссертации М.В. Галочкина «Методы и средства образно-семантического сопровождения процессов решения проектных задач»

Научно-техническая комиссия в составе:

Председатель комиссии: первый заместитель генерального директора по науке, к.т.н. Э.Д. Павлыгин.

Члены комиссии: начальник НИЛ-202, к.т.н. А. И. Моисеев;
заместитель начальника КНИО-1 - заместитель генерального конструктора, Ю.Л. Корноухов;
ведущий инженер-программист отдела ИАСУП, к.т.н. А.Н. Подобрий.

Настоящим актом подтверждается использование следующих научных и практических результатов диссертационной работы М.В. Галочкина «Методы и средства образно-семантического сопровождения процессов решения проектных задач» при разработке «Унифицированного имитатора радиолокационных сигналов»:

1. Комплекс инструментально-технологических средств обслуживающий авторский подход к прецедентно-ориентированному решению профессиональных задач с использованием образно-семантической поддержки, способствующей конструктивному и управляемому включению в процессы решения мысленного воображения и повышающий эффективность человеко-компьютерного взаимодействия.

2. Средства контроля версий вопросно-ответных проекций графических моделей, учитывающие специфику такого представления и позволяющие фиксировать этапы работы над задачей и переключаться между ними с целью возврата или проверки корректности полученных результатов.

3. Средства перевода декларативного представления в согласованное с ним прологоподобное представление с целью экспериментирования, работы со словарем онтологий и поиска семантических ошибок.

4. Средства перевода концептуально-алгоритмического представления в согласованное с ним псевдокодовое представление и отладка его в псевдокодовом интерпретаторе.

Указанные научно-технические результаты использованы в рамках работ по созданию унифицированного имитатора радиолокационных сигналов в рамках работ инновационного развития ФНПЦ АО «НПО «Марс» (проект «Унифицированный имитатор РЛС»).

Работа обладает научно-технической новизной и может быть использована в проектных организациях, разрабатывающих сложные автоматизированные системы, для рационализации оперативного планирования проектных работ, учета возникающих в процессе работы рисков, обеспечения контроля и приоритезации работ.

Председатель комиссии:
Первый заместитель генерального директора по науке,
к.т.н. Э.Д. Павлыгин

Члены комиссии:

Начальник НИЛ – 202,
к.т.н. А. И. Моисеев

Заместитель начальника КНИО-1 - заместитель
генерального конструктора Ю.Л. Корноухов

Ведущий инженер-программист
отдела ИАСУП, к.т.н. А.Н. Подобрий