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

На правах рукописи

Куликова Анна Александровна

МЕТОДЫ И СРЕДСТВА ФОРМИРОВАНИЯ И ИСПОЛЬЗОВАНИЯ ОНТОЛОГИЙ ПРОЕКТОВ В ПРОЦЕССЕ ПРОЕКТИРОВАНИЯ АВТОМАТИЗИРОВАННЫХ СИСТЕМ

Специальность: 05.13.12 – Системы автоматизации проектирования (информационные технологии и промышленность)

Диссертация на соискание ученой степени
кандидата технических наук

Научный руководитель – доктор технических наук, доцент,
Негода Виктор Николаевич

Ульяновск – 2021

Оглавление

Введение 6

Глава 1. Анализ подходов и методов онтологической поддержки процесса разработки АС 16

1.1 Современное состояние технологий информационной поддержки производственного процесса 18

1.2 Понятие семантического разрыва между этапами разработки АС и обзор факторов его возникновения 21

1.2.1 Понятие семантического разрыва 21

1.2.2 Обзор факторов возникновения семантического разрыва в проектировании АС и его негативных последствий 23

1.3 Тематико-аналитический обзор применения онтологий на различных фазах ЖЦ разработки АС, в том числе на этапах концептуального проектирования 26

1.3.1 Понятие онтологии 26

1.3.2 Использование онтологии на различных фазах ЖЦ проекта 28

1.3.3 Использование онтологий в проектировании 30

1.4 Тематико-аналитический обзор подходов к онтологическому моделированию и способов построения онтологий 35

1.4.1 Онтологическое моделирование 35

1.4.2 Семантические технологии для онтологического инжиниринга 36

1.4.3 Сравнительный анализ инструментальных средств для онтологического инжиниринга 38

1.5 Методология проектного мышления и ее место в разработке АС 42

1.6 Методы и средства поддержки генерации кода 45

Выводы по главе 1 48

Глава 2. Подход к онтологической поддержке проектирования и обслуживающая его модель проектного процесса 50

2.1 Разработка подхода к онтологической поддержке проектирования 50

2.1.1 DT-подход, как основа процесса формирования онтологических моделей 50

2.1.2 Онтологическая модель проекта, как система связанных онтологий . 57

2.1.3 Онтологическое моделирование, как система инкрементальных информационных процессов 63

2.2 Формальная структура онтологической модели проекта и процедура ее формирования 65

2.3 Модель проектного процесса разработки АС с применением механизмов онтологической поддержки 69

2.3.1 Анализ требований и формирование онтологической модели требований 71

2.3.2 Формирования онтологической модели прецедентов 72

2.3.3 Формирование концепции проекта и предварительных проектных решений 75

2.3.4 Формирование онтологии реализации и генерация исходного кода .. 76

2.4. Онтологическое моделирование в инфраструктуре удовлетворения потребительских запросов 78

Выводы по главе 2 82

Глава 3. Разработка технологии и инструментальных средств онтологической поддержки проектирования 83

3.1 Организация онтологической модели проекта 85

3.1.1 Формирование онтологии требований 87

3.1.2 Формирование онтологии проектирования 90

3.1.3 Формирование онтологии реализации 95

3.2 Модуль обработки текстовой информации..... 97

3.2.1 Фильтрация токенов 98

3.2.2 Выявление многословных концептов (словосочетаний)..... 99

3.3.3 Поиск следов семантических связей в тексте..... 102

3.3 Модуль генерирования UML-диаграмм 106

3.4 Модуль генерирования кода..... 109

Выводы по главе 3 112

Глава 4. Использование средств онтологической поддержки..... 114

4.1 Использование онтологического моделирования при разработке системы логического управления плиткоукладчиками..... 114

4.1.1 Формирование онтологии требований 115

4.1.2 Формирование онтологии проектирования и онтологии реализации как процесс последовательных трансформаций 116

4.1.3 Генерация UML и исходного кода программы 123

4.1.4 Сравнительный анализ производительности труда проектировщика с учетом использования методов и средств онтологического сопровождения проектирования..... 127

4.2 Создание средств поддержки проектирования прототипов базовых средств программного управления транспортными роботами на основе онтологического моделирования 131

4.2.1 Структура онтологической модели 132

4.2.1 Архитектура средств онтологической поддержки прототипирования..... 135

4.3 Разработка управляющей продвижением товаров народного потребления подсистемы АС с использованием средств онтологического моделирования. 139

4.3.1 Специфицирование исходных прототипов и формирование ОМ исходных проектных решений 140

4.3.2 Трансформирование ОМ исходных проектных решений формирование ОМ целевой подсистемы 144

4.3.3 Оценка сокращения семантического разрыва 147

Выводы по главе 4 149

Заключение 150

Список сокращений и условных обозначений 152

Список литературы 153

Приложение 1. Вопросно-ответный протокол анализа дискурсов 168

Приложение 2. Вопросно-ответный протокол анализа задачи исследования в промежуточном состоянии 185

Приложение 3. Мотивационно-целевой анализ задачи исследования в промежуточном состоянии 189

Приложение 4. Анализ трудозатрат при разработке АС с учетом онтологического моделирования 192

Приложение 5. Анализ трудозатрат на онтологическое моделирование АС 199

Приложение 6. Свидетельства о регистрации программ для ЭВМ и программно-информационных продуктов 201

Приложение 7. Акты внедрения 205

Введение

В настоящее время специалистами в области автоматизированного проектирования (АП) проводится большое количество исследований, связанных с ***вовлечением онтологий в проектный процесс***. Анализ публикаций по данной тематике показывает, что объекты проектирования, которые активно вовлекаются в онтологические модели (ОМ), на практике зачастую представляют собой технические объекты; однако, когда в качестве объектов проектирования выступают автоматизированные системы (АС), доминирует подход к онтологическому моделированию, обслуживающий повышение эффективности разработки программного обеспечения (ПО). Иными словами, АС рассматриваются исключительно как сложные системы, интенсивно использующие ПО (Software Intensive Systems, SIS), в то время как основные конкурентные преимущества разрабатываемых АС достигаются за счет применения такой технологии автоматизированного проектирования, которая вовлекает в проектный процесс модели сущностей объектов автоматизации.

Использование инструментов программной инженерии, с одной стороны, повышает производительность труда специалистов, с другой стороны, приводят к доминированию таких технологий разработки ПО, которые предполагают активное разделение труда, что, в свою очередь, порождает эффект, для которого широко используется термин ***«информационный разрыв»*** или ***«семантический разрыв»***. Наиболее остро проявляется данный эффект при переходе от анализа требований к концептуальному проектированию и далее – к разработке ПО: это, среди прочего, связано с тем, что значительно различаются совокупности компетенций и ценностей специалистов, работающих на данных этапах. А доминирование направленности онтологического моделирования на повышение эффективности процесса разработки ПО не решает указанную проблему, поскольку при таком подходе из фокуса уходит объект автоматизации, а на первый план выходят аспекты технологии программирования.

Помимо этого, возрастающая с каждым годом сложность АС приводит к необходимости вовлекать в их создание большое количество специалистов самого разного профиля, задействовать широкое пространство ***прототипов***, которые представляют собой прецеденты в накопленном опыте автоматизации и зачастую нуждаются в разной степени модификациях, а также инструментальные средства, сложность которых также возрастает. Все это неизбежно приводит к появлению ***несогласованности в работе***, а также к ***недостатку понимания*** проектировщиками проектных задач, вызванному информационными разрывами.

Более того, одним из негативных проявлений «лоскутности» прототипов является вовлечение в пространство АП многих ***пространств имен***, что является причиной ***нарушения концептуального единства***: когнитивные усилия, прикладываемые к осознанию того, что разными именами называется одно и то же понятие, настолько велики, что доминирует тактика изоляции одних частей проекта от других (например, микросервисная или мультиагентная архитектура), однако на этапе разработки и эксплуатации сложных АС многообразие пространств имен заметно и неизбежно снижает производительность труда.

Под особое внимание исследователей и специалистов в области АП попадают ранние этапы проектирования (в частности, концептуальное проектирование), поскольку ошибки, допускаемые на ранних стадиях, сказываются на каждом последующем этапе разработки, и их исправление требует тем больших затрат, чем позднее они обнаружены. Следовательно, важным представляется уделять особое внимание тщательности проработки и специфицирования проектных решений на этапе концептуального проектирования, т.к. ***концептуальное единство***, ***непротиворечивость*** и ***полнота*** разрабатываемых проектных решений обеспечивается в последующем за счет активного использования проектных спецификаций именно этого этапа. Согласно зарубежным исследованиям, на этапе концептуального проектирования широко распространена практика проектирования, называемая ***Design Thinking (DT)***, суть которой – исследование объекта автоматизации на протяжении всех фаз его жизненного цикла и поиск решения поставленных задач в условиях значительной неопределенности.

На основании вышеизложенного перспективной представляется разработка такого ***подхода к онтологическому моделированию АС***, который учитывал бы свойства объекта автоматизации и обслуживал бы проектный процесс на всех этапах – начиная от анализа требований и заканчивая реализацией. При этом целесообразно рассматривать уровень доли сущностей, сохраняющихся в онтологических моделях системы на разных этапах, как характеристику успеха по сокращению семантического разрыва, а также способ противостоять доминированию технологий программной инженерии в проектном процессе и поддерживать высокий уровень проникновения результатов DT, полученных на уровне концептуального проектирования, во все дальнейшие этапы разработки АС.

Все вышеизложенное указывает на **актуальность исследований и разработок, направленных на снижение негативных последствий ошибок концептуального проектирования, повышение концептуальной целостности разрабатываемых проектных решений, а также сокращение описанного семантического разрыва за счет более активного повторного использования проектных спецификаций концептуального проектирования на этапах технического проектирования и реализации.**

Цель исследования – снизить трудозатраты на проектирование автоматизированных систем за счет сокращения семантического разрыва между спецификациями проектных решений различных стадий проектного процесса, а также автоматизации разработки проектных решений и их реализации.

Задачи диссертационного исследования:

1) провести тематико-аналитический обзор исследований в области формирования и использования онтологий в проектировании систем, интенсивно использующих программное обеспечение, в том числе АС;

2) исследовать возможность применения DT-подхода при формирования онтологических моделей проекта АС;

3) разработать систему требований и архитектурные решения для технологии и инструментария онтологического сопровождения проекта;

4) исследовать возможную степень покрытия онтологической моделью необходимых проектных решений в процессе проектирования АС;

5) разработать совокупность аналитических моделей системы онтологий проекта, обслуживающих анализ требований, проектирование и реализацию, а также поддерживающую частичную автоматическую генерацию UML-диаграмм и исходного кода программ автоматизации;

6) разработать технологию для основных видов работ проектировщиков при формировании и использовании онтологии, включая действия по контролю языка проекта, извлечению из артефактов проектирования новых онтологических сущностей, применению онтологических моделей проекта для создания онтологических спецификаций проектных решений;

7) разработать функциональный прототип инструментально-технологического комплекса формирования и использования онтологий;

8) провести апробацию предложенной технологии и разработанного функционального прототипа в условиях проектирования АС и оценить результаты проведенных экспериментов.

В рамках диссертационного исследования разработана технология и средства прецедентно-ориентированного формирования и использования онтологий в проектировании АС, использование которых призвано повысить производительность труда проектировщиков и способствовать снижению негативных проявлений человеческого фактора в решении проектных задач.

Научная новизна результатов исследования состоит в следующем:

1) предложена ***технология прецедентно-ориентированного онтологического сопровождения процесса проектирования АС***, отличающаяся от известных интегрированным в процесс решения проектных задач и осуществляемым параллельно с этим процессом формированием и использованием онтологии проекта, по ходу которого, оперативно взаимодействуя с доступным опытом, проектировщики применяют механизмы конструктивного проектного мышления, нацеленные на подготовку решений для их повторного использования,

в условиях сохранения доминирующей роли сущностей, характеризующих объекты и процессы автоматизации;

2) предложено семейство ***аналитических моделей системы онтологий проекта***, охватывающих этапы анализа требований, формирования предварительных проектных решений и их реализации, отличающаяся от известных более высокой долей присутствия сущностей объектов и процессов автоматизации в проектных спецификациях, а также наличием таких типов понятий, свойств, отношений, аксиом и функций интерпретации, которые обеспечивают возможность автоматизации проектного процесса, в том числе генерирования UML-диаграмм и исходных кодов программ автоматизации;

3) разработаны ***алгоритмы формирования спецификаций онтологических моделей***, отличающиеся от известных поддержкой автоматической генерации агрегатов сущностей и отношений, направленные на сокращение трудозатрат на онтологическое моделирование и поддерживающие контроль концептуальной целостности и полноты проектных решений;

4) разработаны ***алгоритмы формирования проектных решений АС на основе онтологических моделей***, в том числе UML-диаграмм и исходного кода программ автоматизации, отличающиеся от известных поддержкой управления изменениями за счет реализации правил логического вывода, связывающих онтологию требований с онтологиями проектирования и реализации.

Материальное воплощение новаций привело к созданию технологии и инструментария, применение которых вводит в процесс проектирования новые возможности по достижению проектировщиками необходимого и достаточного понимания разрабатываемой АС и регистрации его актов, когда в этом появляется необходимость, а также новые возможности по построению концептуально целостных проектных решений.

Новые возможности обусловлены формированием по ходу реализации проекта его уникального естественно-профессионального языка с онтологией в основе, содержание которой регистрирует семантику языка с ее материальным воплощением в моделях проекта и в составляющих реализованной АС.

Применение онтологии способствует контролируемому использованию языка проекта, и в первую очередь его семантики, в рассуждениях проектировщиков, в проектных документах, а также в других артефактах проектирования. Тем самым, предлагаемая версия формирования и использования онтологии проектирования приводит к снижению негативных проявлений человеческого фактора в решении проектных задач, прежде всего связанных с замещением сущностей объектов и процессов автоматизации сущностями программной инженерии, что способствует повышению степени успешности проектных работ и их результатов.

Область исследования соответствует паспорту специальности 05.13.12 «Системы автоматизации проектирования (информационные технологии и промышленность)», а именно – п. 3 «Разработка научных основ построения средств САПР, разработка и исследование моделей, алгоритмов и методов для синтеза и анализа проектных решений, включая конструкторские и технологические решения в САПР и АСТПП».

Объектом исследования в диссертации является процесс проектирования сложных АС.

Предметом исследования являются методы и средства формирования и специфицирования проектных решений АС на основе онтологического моделирования, которые позволяют существенно повысить производительность труда проектировщика за счет сокращения семантического разрыва между различными стадиями проектного процесса, а также обеспечения концептуальной целостности, непротиворечивости и полноты разрабатываемых проектных решений.

Методы исследования. При выполнении работы использованы основные положения и методы онтологического анализа, системного анализа, искусственного интеллекта, а также объектно-ориентированного программирования при построении программного комплекса.

Теоретическая значимость работы заключается в разработке подхода к онтологической поддержке проектов в условиях оперативного взаимодействия проектировщиков с доступным опытом с обеспечением доминирующей роли

проектных спецификаций, формируемых на основе DT-подхода в ходе концептуального проектирования, в ходе создания проектных спецификаций всех последующих этапов проектирования вплоть до реализации.

Практическая значимость полученных результатов состоит в разработке программного обеспечения, включающего следующие компоненты:

1. Средства формирования системы онтологий проекта, охватывающей требования, предварительные проектные решения и реализацию, в соответствии с разработанными моделями на основе текстов проектной документации и неформальных описаний объектов и процессов автоматизации.

2. Средства трансформации онтологических спецификаций за счет реализации правил логического вывода, связывающих онтологию требований с онтологиями проектирования и реализации.

3. Средства использования системы онтологий проекта для генерации проектных решений, в том числе UML-диаграмм и исходного кода программ.

При выполнении работы использованы основные положения и методы онтологического анализа, системного анализа, искусственного интеллекта, а также объектно-ориентированного программирования при построении программного комплекса.

Основные положения, выносимые на защиту:

1. Методика интегрированного в проектный процесс прецедентно-ориентированного онтологического сопровождения процесса проектирования АС в условиях оперативного взаимодействия проектировщиков с доступным опытом, обеспечивающая доминирующую роль результатов применения DT-подхода, полученных в ходе концептуального проектирования, на всех этапах разработки АС вплоть до реализации.

2. Аналитические модели системы онтологий проекта АС, охватывающие этапы анализа требований, формирования предварительных проектных решений и реализации, а также обладающие высокой долей присутствия продуктов DT-подхода в проектных спецификациях всех этапов разработки вплоть до создания исходного кода программ АС.

3. Средства формирования и использования системы онтологий, поддерживающей единство пространств имен, задействованных на различных этапах проектного процесса, а также обладающей набором функций интерпретации, обеспечивающим возможность автоматизации проектного процесса, в том числе генерацию UML-диаграмм и исходного кода целевых программ.

Достоверность результатов работы. Достоверность полученных результатов обеспечена результатами практического использования, в том числе в ряде НИОКР, выполненных в Ульяновском государственном техническом университете, направленных на решение научно-технических задач. К наиболее важным результатам следует отнести участие в выполнении грантов РФФИ:

- №18-07-00989 «Технология и инструментарий образно-семантического прототипирования в концептуальном проектировании систем с программным обеспечением»;
- №18-47-730016 «Технология и инструментарий прецедентно-ориентированного формирования и использования онтологий проектирования автоматизированных систем»;
- №18-47-732012 «Методы и средства содержательно-эволюционной теоретизации человеко-компьютерной деятельности в процессах проектирования и эксплуатации автоматизированных систем».

Реализация и внедрение результатов работы. Разработанные программные средства внедрены в практику работы ФНПЦ АО «НПО «Марс» (г. Ульяновск) и учебный процесс Ульяновского государственного технического университета (г. Ульяновск).

Апробация работы. Основные положения и результаты диссертационной работы докладывались и обсуждались на:

- научном семинаре «Онтология проектирования» (Самарский университет, ИПУСС РАН, 2021 г.);

- Международной конференции «Creativity in Intelligent Technologies and Data Science» (CIT&DS 2019), г. Волгоград;
- 11-ой, 12-ой и 14-ой Международных конференциях «Интерактивные системы: проблемы человеко-компьютерного взаимодействия» (IS-2015, IS-2017, IS-2019), г. Ульяновск;
- 18-ой Международной конференции по компьютерным наукам и приложениям (International Conference on Computational Science and Applications, ICCSA 2018), г. Мельбурн, Австралия;
- 2-ой и 3-ей Международных научных конференциях «Интеллектуальные информационные технологии в технике и на производстве» (IITI'17, IITI'18), г. Варна, Болгария и г. Сочи;
- 26-ом Форуме по телекоммуникациям (TELFOR-2018), г. Белград, Сербия;
- 2-ой Международной научно-практической конференции «Нечеткие системы и мягкие вычисления. Промышленные применения» (FTI 2018), г. Ульяновск;
- 5-ой Международной научно-практической конференции «Электронное обучение в непрерывном образовании 2018» (ЭОНО-2018), г. Ульяновск;
- Международных Конгрессах по интеллектуальным системам и информационным технологиям (IS&IT'17, IS&IT'18, IS&IT'21), Дивноморское;
- 15-ой Международной конференции по компьютерной и когнитивной лингвистике (TEL'2018), г. Казань;
- 7-ой, 8-ой, 9-ой, 10-ой и 12-ой Всероссийских научно-технических конференциях аспирантов, студентов и молодых ученых (ИВТ-2015, ИВТ-2016, ИВТ-2017, ИВТ-2018, ИВТ-2021), г. Ульяновск;
- 7-ом, 8-ом, 9-ом и 10-ом Всероссийских школах-семинарах аспирантов, студентов и молодых ученых «Информатика, моделирование, автоматизация

проектирования» (ИМАП-2015, ИМАП-2016, ИМАП-2017, ИМАП-2018), г. Ульяновск;

- 17-ой Международной конференции по компьютерным наукам и приложениям (2017), г. Триест, Италия;
- IV молодежном инновационном форуме Приволжского федерального округа, г. Ульяновск.

Публикации. По теме диссертационной работы опубликовано 31 печатная научная работа (из них 1 статья из перечня ВАК, а также 9 статей в изданиях, индексируемых в Scopus), 1 учебно-методическое пособие и 3 свидетельства о регистрации ПО для ЭВМ.

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

Структура и объем работы. Диссертация изложена на 207 страницах машинописного текста, содержит 40 рисунков, 6 таблиц, состоит из введения, четырех глав, заключения, списка литературы из 126 наименований на 15 страницах и 7 приложений на 40 страницах, включая акты о внедрении.

Глава 1. Анализ подходов и методов онтологической поддержки процесса разработки АС

Современные исследователи отмечают, что производительность труда в сфере производства с начала 20 века возросла в сотни раз, а в области проектирования *новых объектов* – только в 1.5-2 раза. «Это обусловливает большие сроки проектирования новых объектов, что не отвечает потребностям развития экономики. Очевидность того факта, что развитие новой техники в современных условиях замедляется не столько отсутствием научных достижений и инженерных идей, сколько сроками и не всегда удовлетворительным качеством их реализации при конструкторско-технологической разработке» [26].

И.В. Лофицкий и С.А. Матюнин описывают сложившуюся практику в области АП следующим образом: «в процессе конструирования и разработки технологии может потребоваться коррекция принципиальных схем, структуры системы и даже исходных данных, поэтому процесс проектирования является не только многоэтапным, но и многократно корректируемым по мере его выполнения» [27]. Очевидно, что данная практика приводит к дополнительным трудозатратам, увеличению семантического разрыва между концептуальной моделью и артефактами проектирования АС, а также в некоторых случаях является следствием ошибок раннего проектирования, неполноты анализа требований и отсутствия системного подхода, целью которого и является обеспечение концептуальной целостности проектных решений.

В работе [28] отмечается, что современные системы автоматизированного проектирования (САПР) активно используются на поздних этапах проектирования – в частности для моделирования, прототипирования и даже непосредственно для обеспечения некоторых производственных процессов, – однако практически не применяются на этапе концептуального проектирования, что происходит по причине того, что, как утверждается в [29], в распространенных на сегодняшний день САПР практически отсутствуют средства поддержки данного процесса.

По мнению В.Ф. Хорошевского [12], «общим трендом в области автоматизации проектирования ПО является использование методов и средств онтологического моделирования процессов проектирования и спецификаций разрабатываемых систем». При этом, развитые модели программной инженерии, повышающие производительность труда программистов, являются абстрактными относительно объектов автоматизации, что порождает следующую проблему: успехи, достигнутые в области онтологического моделирования, обслуживающего процесс проектирования ПО, сами по себе могут провоцировать информационный разрыв за счет того, что в модельных конструкциях проектирования ПО происходит абстрагирование от объекта автоматизации и начинают доминировать широко распространенные конструкции программной инженерии.

В тематико-аналитическом обзоре [12] приведены примеры успешного использования онтологических моделей как для моделирования процессов проектирования ПО, так и специфицирования разрабатываемых систем. Следует отметить, что, несмотря на активное развитие нового направления – Ontology-Based Software Engineering – в области программной инженерии, в области промышленных САПР использование онтологий для моделирования непосредственно процесса проектирования не является распространенной практикой. При этом В.Ф. Хорошевский утверждает, что «современная постановка задачи автоматизации проектирования … предполагает наличие адекватных моделей всего ЖЦ», что в итоге должно привести к смещению фокуса к «моделям генерации блоков из согласованной системы онтологических паттернов их внутренних спецификаций».

Кроме того, В.Ф. Хорошевский заключает, что «в настоящее время общепризнанной классификации онтологических моделей не существует», а имеющиеся «являются либо слишком общими, чтобы использоваться непосредственно в качестве основы для построения системы онтологических моделей процессов проектирования ПО, либо настолько частными, что их использование в данной предметной области представляется затруднительным».

С учетом вышеизложенного, в первой главе диссертации исследуется современное состояние технологий информационной поддержки производственного процесса, анализируется понятие семантического разрыва и факторы его возникновения в процессе проектирования; приводится обзор существующих методов и средств онтологической поддержки систем, интенсивно использующих ПО, к классу которых относятся АС; исследуются возможности применения онтологий на различных этапах жизненного цикла (ЖЦ) разрабатываемых систем, в том числе на этапах концептуального проектирования; проводится тематико-аналитический обзор подходов к онтологическому моделированию и способов построения онтологий.

1.1 Современное состояние технологий информационной поддержки производственного процесса

CALS-технологии (англ. Continuous Acquisition and Life cycle Support — непрерывная информационная поддержка поставок и ЖЦ изделий), или ИПИ (информационная поддержка процессов ЖЦ изделий) – на сегодняшний день является одним из трендов в развитии производственных процессов. Это подход к проектированию и производству высокотехнологичной и наукоемкой продукции, заключающийся в использовании компьютерной техники и информационных технологий на всех стадиях ЖЦ производства.

Впервые о CALS теоретики и практики в области АП заговорили в конце XX века, и за прошедшие годы применение данной технологии показало хорошие результаты: так, в США использование CALS в оборонной промышленности и военно-технической инфраструктуре «позволило ускорить выполнение НИОКР на 30-40%, уменьшить затраты на закупку военной продукции на 30%, сократить сроки закупки ЗИП на 22%, а также в 9 раз сократить время на корректировку проектов» [30]. Несмотря на это, как отмечает автор статьи [31], в некоторых странах, включая Россию, внедрение CALS все еще находится на ранних стадиях и

носит несистемный характер, что свидетельствует об актуальности данного вопроса.

На рис. 1.1 проиллюстрирован процесс внедрения CALS и технологий электронной коммерции на начальных стадиях развития проекта, описанный в статье [14]. Как видно на иллюстрации, все стейкхолдеры, вовлеченные в производственный процесс, имеют доступ к общей базе данных. Информация, хранящаяся, а этой базе данных, может быть использована на различных стадиях ЖЦ проекта.

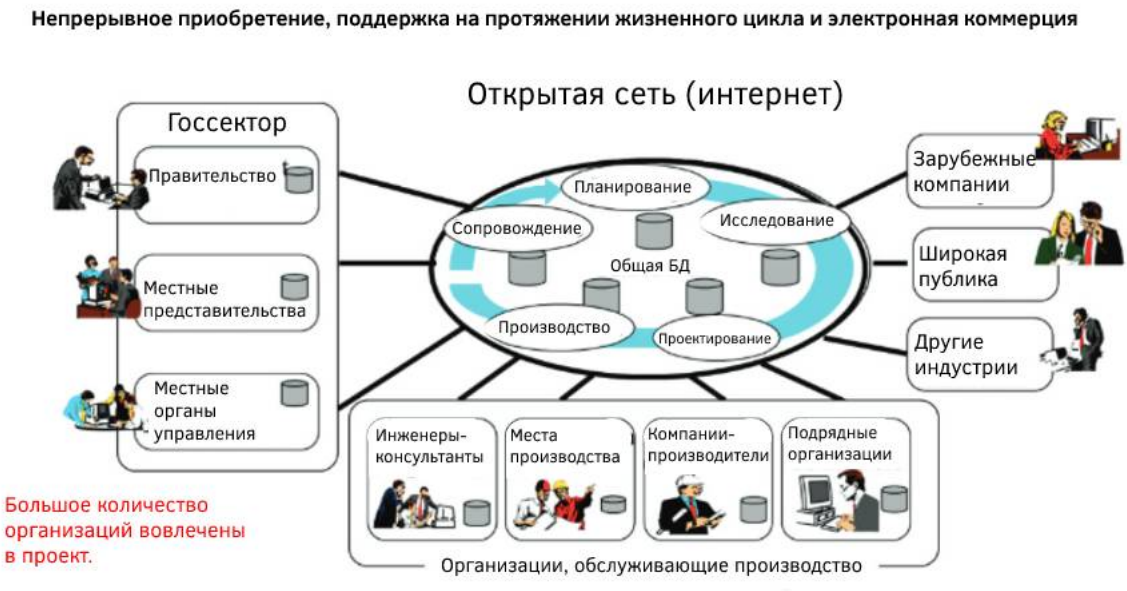


Рисунок 1.1 - CALS и E-Commerce в производстве

Автор исследования [32] утверждают, что преимущества применения гибкого подхода к организации интегрированной инфраструктуры производства и поставок в традиционном производственном процессе существенно недооценены. Под гибким подходом в данном случае понимается способность оперативно реагировать на постоянно меняющиеся требования потребителей. Таким образом, одна из целей развития современных САПР – *повышение реактивности инфраструктуры удовлетворения потребительских запросов*. Для развития такого подхода важнейшим аспектом является глубокое внедрение сложных

информационно-коммуникационных технологий (ИКТ) на всех стадиях производственного процесса, включая поставки.

К классу сложных ИКТ, несомненно, относятся технологии Интернета вещей (Internet of Things, IoT), применение которых может обеспечить повышение упомянутой реактивности проектного процесса, обслуживаемого различными САПР. Авторы статьи [33] отмечают, что такие процессы, как децентрализация, модуляризация и автоматизация производственных технологий, которые в настоящее время можно отнести к мировым трендам развития производства, способствуют применению технологий IoT. Также они утверждают, что данные технологии могут быть эффективно использованы в области поддержки принятия решений на различных стадиях производственного процесса, при этом, данное утверждение справедливо для самых разных ПрО; следовательно, возможности технологии IoT обязательно должны учитываться при создании новых САПР. На рис. 1.2 наглядно показано, как, благодаря применению IoT, необходимые данные для принятия решений могут быть легко доступны, даже если производственные и обслуживающие команды распределены географически.

Рисунок 1.2 - Применение IoT в производстве

Данная концепция вписывается в парадигму активно развивающейся технологии CALS, целью которой является построение открытых распределенных АС для проектирования и управления в промышленности, а также существенное сокращение объемов проектных работ благодаря непрерывной информационной поддержке. По мнению И.П. Норенкова, главная проблема построения таких систем – «обеспечение единообразного описания и интерпретации данных, независимо от места и времени их получения в общей системе» [34]. Одним из способов решения данной проблемы может стать внедрение технологий *онтологического моделирования* различных продуктов проектирования, начиная со спецификаций потребительских запросов.

Заключение 1.1. Применение технологии онтологического моделирования является одним из способов повышения степени единообразия описания и интерпретации разнородных данных, что, в свою очередь, способствует более широкому внедрению в производственный процесс современной технологии CALS, уже доказавшей свою эффективность. Для обеспечения этой возможности необходимо создание соответствующих методов и средств.

1.2 Понятие семантического разрыва между этапами разработки АС и обзор факторов его возникновения

1.2.1 Понятие семантического разрыва

В литературе, посвященной компьютерным наукам, можно встретить различные определения понятия «семантический разрыв» (некоторые исследователи также называют это явление информационным разрывом – в контексте данного исследования будем считать эти понятия синонимичными). Авторы статьи [13] приводят следующее его определение: «семантический разрыв есть разница в значении понятий, сформированных в различных системах представления». К таким системам представления в процессе проектирования АС могут относиться:

- естественный язык (при этом к разным системам представления можно отнести естественный язык, на котором «говорит» заказчик; более формализованный естественный, на котором пишется техническое задание и другая проектная документация; естественный язык, на котором общаются друг с другом различные проектные команды);
- различные графические нотации (например, UML), используемые для описания разрабатываемых моделей;
- другие формальные языки, в том числе языки программирования и языки разметки, которые используются для создания средств автоматизации проектирования.

Еще одно, более философское, определение семантического разрыва приводится в [15]: «различие между тем, что компьютер может обработать, и тем, что человек может понять».

Кроме того, в исследованиях встречается понятие «разрыв в знаниях» (knowledge gap), которое имеет схожий смысл: «разрыв между тем, что нам необходимо знать для успешного завершения проекта, и тем, что мы реально знаем» [16].

Согласно концепции анализа разрывов (gap-разрывов) существует ***5 видов разрывов*** [20]:

1) разрыв между потребительскими запросами и представлением производителя продукта о потребительских запросах;

2) разрыв между представлением производителя продукта о потребительских запросах и спецификациями потребительских запросов;

3) разрыв между спецификациями потребительских запросов и производимым продуктом;

4) разрыв между производимым продуктом и информациях, которую получает потребитель о продукте;

5) разрыв между производимым продуктом и восприятием потребителя данного продукта.

На рис. 1.3 [14] (данные информационного центра строительства Японии при министерстве строительства Японии) схематически показан информационный разрыв между различными фазами ЖЦ проекта, такими как планирование, проектирование, производство и сопровождение. По оси X показаны проектные фазы, по оси Y – качество и количество информации, которой оперируют задействованные специалисты. Как видно из рисунка, информация, накопленная на одной фазе, не может быть полностью перенесена в следующую фазу, т.е. регулярно происходит ее количественное и качественное сокращение. На каждой новой фазе производится дополнительная информация, вследствие чего происходит общее снижение продуктивности.

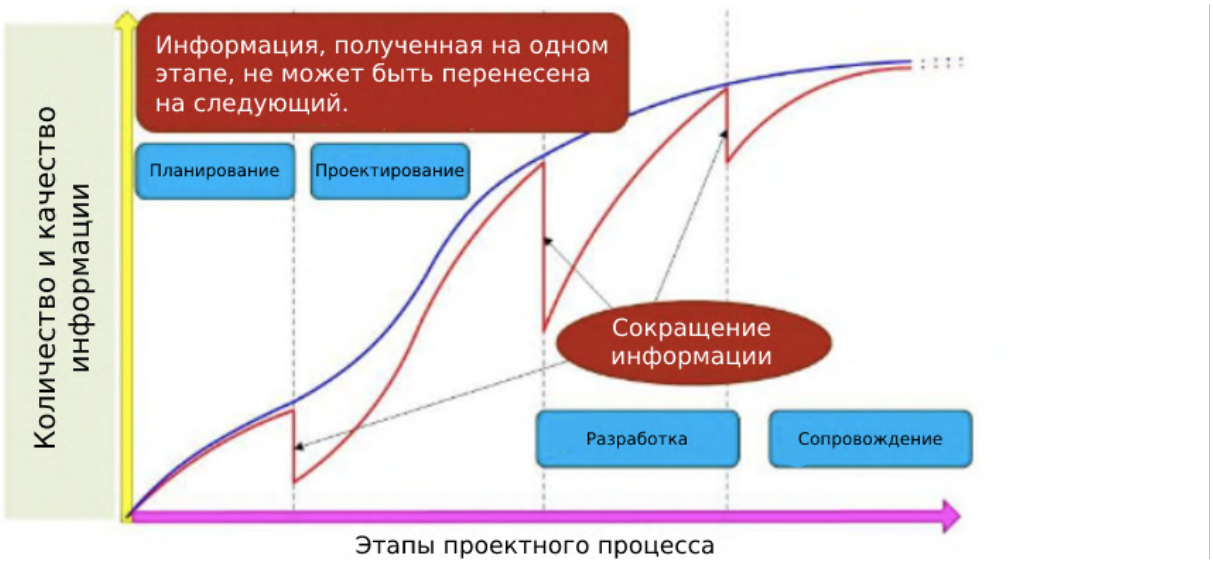


Рисунок 1.3 - Информационный (семантический) разрыв

1.2.2 Обзор факторов возникновения семантического разрыва в проектировании АС и его негативных последствий

Среди факторов возникновения семантического разрыва в проектном процессе исследователи отмечают следующие:

- использование в проектном процессе разных систем и способов представления информации [13];

- неполнота генерируемой информации, приводящая к неприемлемости ее использования на последующих стадиях проектного процесса [14];
- вовлечение в проектный процесс, в том числе в процесс решения взаимосвязанных задач, различных проектных команд [17];
- вовлечение в проектный процесс различных стейкхолдеров: системных аналитиков, проектировщиков, консультантов, поставщиков, – которые зачастую получают информацию фрагментарно [18];
- постоянно возрастающая сложность разрабатываемых проектов [19];
- вовлечение в проектный процесс большого количества людей [35];
- вовлечение людей с разными компетенциями на разных проектных фазах (нарушение преемственности) [35].

Согласно исследованию [35] о причинах и последствиях информационных разрывов в разработке масштабных систем с программным обеспечением (к классу которых относятся и АС), информационные разрывы приводят к серьезным ***негативным последствиям***, в частности:

- ожидания потребителей не удовлетворяются;
- снижение мотивации в отношении работы над специфицированием требований;
- неполнота покрытия требований (не все требования могут быть выполнены);
- несоответствие сценариев тестирования исходным требованиям;
- снижение качества;
- снижение эффективности (часть затраченных усилий не приводит к результату) и др.

Часть перечисленных негативных последствий связаны с нарушением ***концептуальной целостности***. Специалисты в области разработки сложных автоматизированных систем определяют данное понятие схожим образом – многие называют концептуальную целостность (концептуальное единство, согласованность, непротиворечивость) главным требованием к разрабатываемой

сложной системе. Так, С.В. Назаров в своей монографии пишет следующее: «Концептуальная целостность представляет собой меру единообразия способа взаимодействия с пользователем ... Система, лишенная концептуальной целостности, – это система, в основе которой нет единообразия. В результате такая система характеризуется сложным взаимодействием с пользователем и излишне сложной структурой» [94]. Ф. Брукс называет концептуальной целостностью «всепроникающее видение программы» [95], которое, по его мнению, и есть главный ингредиент успеха.

В.В. Соловьев в работе [26] отмечает, что «концептуальная целостность определяется информационным обеспечением САПР», к которому, как известно, относятся и проектные онтологии. А поскольку онтология, по своему определению, является средством концептуализации, то справедливо предположить, что онтологическая поддержка проектирования может являться наиболее эффективным инструментом достижения концептуальной целостности.

Заключение 1.2. Между фазами проектного процесса всегда наблюдается семантический (информационный) разрыв, и чем он больше, тем к большему количеству негативных последствий он приводит, тем самым, влияя на успешность разработок. Онтологическое моделирование является эффективным средством преодоления такого разрыва при условии, что созданы соответствующие инструменты и обеспечивается высокий уровень присутствия сущностей объектов и процессов автоматизации в проектных спецификациях всех этапов проектирования.

1.3 Тематико-аналитический обзор применения онтологий на различных фазах ЖЦ разработки АС, в том числе на этапах концептуального проектирования

1.3.1 Понятие онтологии

Впервые понятие «онтология», в контексте ее использования в компьютерных науках, появилась в работах Томаса Грубера в 1993 году. Именно он дал первое, наиболее общее определение онтологии, которое известно и актуально и по сей день: «онтология – это точная спецификация концептуализации» [36]. В одной из его статей [37] раскрывается сущность понятия онтологии, которая представляется как «словарь репрезентативных понятий – классов, отношений, функций и констант с согласованными определениями, представленными в виде текста на естественном языке, и машинноисполняемыми декларативными ограничениями на их правильное использование».

В 1997 году Борст сформулировал еще одно, уточненное определение онтологии, назвав ее «формальной спецификацией согласованной концептуализации» [38]. В 1998 году Штудер и др. совместили два определения, сформулировав следующее: «онтология – это формальная и точная спецификация согласованной концептуализации» [39].

Современное понимание онтологии, как утверждает В.А. Лапшин, сводится к любому «описанию декларативных знаний, сделанное на формальном языке и снабженное некоторой классификацией специфицируемых знаний, позволяющей человеку удобно воспринимать их» [40]. При этом, данное описание обязательно должно включать в себя представление декларативных знаний в виде иерархии объектов (классов) – только в этом случае это описание может считаться онтологией.

В общем случае онтология характеризуется наличием структуры, состоящей из следующих компонентов:

1. ***Понятия (концепты)*** – термины, характерные для области знаний (предметной области, проекта, системы), описываемой с помощью онтологии. В некоторых источниках под понятиями скрываются *классы* объектов, которые образуют иерархию: в этом случае в структуре онтологии также предусмотрены *экземпляры* таких классов.

2. ***Атрибуты*** – свойства понятий. Служат для хранения информации, характерной для того или иного понятия (являются опциональным компонентом).

3. ***Отношения*** – зависимости между понятиями, несущие семантическую нагрузку. В зависимости от конфигурации онтологии отношения могут являться как отдельными сущностями, так и частными случаями *атрибутов*, значениями которых выступают другие понятия.

4. ***Функции интерпретации*** – способ содержательного толкования онтологической модели, с помощью которых одним сущностям онтологической модели ставятся в соответствие другие.

5. ***Аксиомы*** – утверждения относительно понятий, атрибутов и отношений онтологии, которые всегда считаются истинными. Аксиомы служат для различных целей: например, проверки модели на непротиворечивость, задания некоторых ограничений модели или вывода новых знаний.

В настоящее время онтологии применяются на различных этапах разработки ПО, начиная с анализа требований, заканчивая тестированием и повторным использованием компонентов таких систем. Однако, как утверждают авторы статьи [41], несмотря на большое количество существующих примеров такого использования, дальнейшие исследования способов семантического описания систем с ПО, а также применения онтологий в данной области является одной из актуальнейших задач в сфере управления знаний об артефактах в процессе создания ПО.

Значительный вклад в разработку методов представления предметных знаний на основе онтологии внесли такие исследователи, как Соснин П.И. Гаврилова Т.А., Загорулько Ю.А., Соловьев В.Д., Хорошевский В.Ф., Gruber T., Ushold M. В работах исследователей Норенкова И.П., Малюх В.Н., Голенкова В.В.,

Смирнова С.В., Боргеста Н.М. подчеркивается актуальность применения онтологического анализа в процедурах проектирования сложных технических систем.

С. Горшков в своем труде «Введение в онтологическое моделирование» подчеркивает, что «одной из наиболее актуальных проблем развития ИТ в настоящий момент является доведение до широкой промышленной эксплуатации таких технологий, которые позволяют строить действительно сложные и комплексные модели, и решать с их помощью те оптимизационные, аналитические, оперативные задачи, перед которыми другие технические средства оказываются бессильны. Многообещающим, но несколько недооцененным на сегодняшний день направлением решения этой задачи является использование так называемых семантических технологий» [42], к классу которых относятся онтологии.

Заключение 1.3. Развитие технологии онтологического моделирования является важнейшим направлением решения сложных аналитических и оперативных задач, к классу которых относится разработка АС. Онтологическое моделирование обеспечивает снижение сложности проектирования многокомпонентных АС со сложным поведением с многообразными сценариями поведением при условии создания соответствующих инструментальных средств. В силу своей направленности (формализация знаний) применение технологии онтологического моделирования может привести к существенному сокращению семантического разрыва между опытом проектировщика и набором проектных ситуаций, который порождает ошибочные ассоциации и, следовательно, ведет к нарушению концептуальной целостности.

1.3.2 Использование онтологии на различных фазах ЖЦ проекта

Как было отмечено во введении, применение онтологий в процессе проектирования АС не является распространенной практикой, в то время как в области программной инженерии данное направление уже получило значительное

развитие. Следовательно, будет полезно исследовать практики применения онтологий на различных фазах ЖЦ ПО.

Полезный обзор возможностей использования онтологий в зависимости от фазы ЖЦ представлена в работе Бхатии и др. [43] – согласно ему, существуют примеры применения онтологий для следующих целей:

- онтологии процессов разработки ПО (планирование процесса реализации проекта, управление проектами);
- онтологии требований (извлечение вопросов, относящихся к функциональным требованиям; оценка и повышение качества сформированных требований; предположения насчет возможных изменений требований);
- онтологии функциональности системы (рекомендации относительно того, какая функциональность является необходимой, опциональной, альтернативной, требуемой или неприменимой);
- онтологии поведения системы (хранение семантической информации о вариантах использования системы);
- онтология проектирования (моделирование архитектуры, логики системы);
- онтологии паттернов (генерация проектных решений);
- онтологии версий (хранение семантической информации о версиях системы, обучение новых членов команды);
- онтологии артефактов проектирования (генерация документации);
- онтологии конфигурации (выявление необходимой конфигурации системы/машины для установки ПО или версии, поддерживаемой текущей конфигурацией);
- онтологии документации (автоматизация процесса документирования, поиск информации);
- онтологии документов (организация порядка в документации);
- онтологии качества (извлечение нефункциональных требований);
- онтологии тестирования (генерация тест-кейсов);

- онтологии дефектов (анализ недостатков системы и определение необходимых средств для их устранения; оценка качества);
- онтологии сопровождения ПО (описание механизма поддержки ПО);
- онтологии технологий (выбор необходимой технологии или инструмента).

В обзоре примеров использования онтологий в сфере ***управления проектами*** [44] упоминается значительной количество онтологий, созданных с целью описания процесса разработки систем с ПО, а именно возможные активности, которые разворачиваются в рамках этого процесса, этапы разработки и применяемые модели.

Авторы работы [45] подчеркивают следующие преимущества использования онтологий в сфере программной инженерии:

- в онтологиях используются логические формализмы, что делает их удобными для применения в сфере разработки сложных систем с ПО, к классу которых относится АС;
- онтологии отличаются гибкостью, что позволяет комбинировать в них информацию из различных источников и строить на ее основе новые знания;
- основной целью создания любой онтологии является моделирование определенной ПрО, а ПрО, в свою очередь, – это то, вокруг чего строится система; соответственно, благодаря этому использование онтологий становится возможным на самых разных стадиях ЖЦ разрабатываемой системы;
- знания, хранимые в онтологической форме, обладают высоким потенциалом для повторного использования.

1.3.3 Использование онтологий в проектировании

В России текущее состояние исследований и разработок в области использования онтологий в проектировании находит свое отражение в журнале «Онтология проектирования», первый выпуск которого состоялся в 2011 году. В

первом номере за 2017 год в журнале представлен полезный обзор [46], раскрывающий возрастающий интерес к тематике онтологии проектирования с помощью гистограммы проиндексированных работ по годам (2008 – 2017) в Scopus и Sciencedirect, содержащих слова «ontology and design» и «ontology», а также результаты поиска по ключевым словам с помощью различных поисковых систем (Google, Яндекс, Rambler, Yahoo и Aol).

Обзор интересен и полезен тем, что в нем представлены обоснованные спецификации ПрО «Онтология проектирования» и названы близкими тематические «родственники» журнала «Онтологии проектирования», такие как: междисциплинарный журнал «Прикладная онтология» (Applied Ontology), аффилированный с Международной ассоциацией по онтологиям и их применениям (International Association for Ontology and its Applications, IAOA); электронный международный журнал «Наука проектирования» (Design Science), выпускаемый в сотрудничестве с Кембриджским университетским издательством (Cambridge University Press) и международным сообществом дизайнеров (Design Society), а также отечественный журнал «Искусственный интеллект и принятие решений».

В работах [1-11] описываются различные задачи, для решения которых полезно их онтологическое сопровождение, такие как:

- поддержка принятия стратегических решений, связанных с оценкой состояния объектов энергетики и топливно-энергетического комплекса в целом, а также с выбором основных направлений их дальнейшего функционирования и развития, с учетом требований энергетической безопасности (ситуационное управление, в т.ч. ситуационный анализ и ситуационное моделирование);
- разработка структурного представления о технологических процессах изготовления деталей машиностроения (онтология технологического процесса);
- автоматизация информационного обслуживания сотрудников рецептурного производства, приведение производства в соответствие стандартам и формальная оценка такого соответствия через формализацию стандартов, на которых осуществляется деятельность рецептурного производства;

- интеграция разнородных по структуре и тематике пространственных баз данных в единую региональную базу данных для организации информационной поддержки принятия решений по управлению крупным промышленным регионом (онтология структуры баз данных);
- накопление, классификация, контроль и поиск прецедентов, представляющих экземпляры конкретных ситуаций и сценариев, которые возникают при практической деятельности в той или иной ПрО (поддержка принятия решений);
- формирование интегрированного пространства знаний для информационно-аналитической поддержки научных исследований и разработок (онтология интегрированного пространства знаний различных научных областей, система онтологических паттернов);
- структурирование и визуализация данных с целью достижения целостного видения и наиболее полного понимания бизнес-процессов в различных областях деятельности и др.

Переходя к представлению состояния исследований и разработок в ПрО «Онтологии проектирования», отметим работы [47-61], которые описывают разнообразные практики применения онтологий в автоматизированном проектировании, а именно:

- онтология контрактов на сервисное обслуживание, как расширения языка ArchiMate (разработка архитектуры предприятия) [47];
- онтологический анализ ценностных предложений, основанный на исследовании современных теорий в области бизнеса и маркетинга, проводимый с целью конкретизации семантики данного понятия, повышения эффективности коммуникации между заинтересованными сторонами, а также гармонизации существующих теорий в области бизнес-стратегий [48];
- FBS-онтология (Function-Behaviour-Structure) – онтология, применяемая в проектировании и построенная по принципу «функция-поведение-

структура»; функция – это то, для чего разрабатывается артефакт, поведение – то, что этот артефакт делает, структура – то, из чего он состоит [49];

- онтология процесса проектирования, направленная на рационализацию теорий проектирования; данная онтология отражает важнейшие тенденции в области развития проектирования за последние 50 лет, а также значимые факторы, которые оказывают влияние на результат проектирования [50];
- трехуровневая онтология, которая построена на основе извлекаемой из CAD-программы (программы автоматизированного проектирования) семантической информации, которая используется роботом для планирования процесса сборки: первый уровень хранит фундаментальные знания, второй – знания ПрО, а третий – знания о разрабатываемом артефакте [51];
- онтология проектирования, построенная с использованием технологии MFBS с целью установления «контакта» между моделированием знаний и непосредственно решением задач [52];
- расширенная онтология, охватывающая такую ПрО, как инновации и процесс проектирования, для достижения более полного понимания того, что влияет на разработку инновационных технологий [53];
- опыт внедрения онтологии в системы управления записями и поддержки принятия решений, который привел к достижению семантической интеграции и повышению эффективности взаимодействия в процессе проектирования [54];
- использование методов представления предметных знаний на основе онтологии в разработке интеллектуального проектного репозитория, обеспечивающая взаимодействие проектировщика с электронным архивом, содержащим информацию о проекте, на семантическом уровне; притом, при реализации данного репозитория используется не одна онтология, а интегрированная система онтологий, компонентами которой являются: онтология ПрО, концептуальные сети проектов, онтология проектных диаграмм, онтология

ЖЦ, тезаурус проектной организации и онтология анализа технических временных рядов [55-61].

Одной из наиболее перспективных задач в области онтологического моделирования, по мнению исследователей, работающих в данной ПрО, в т.ч. В.Ф. Хорошевского, является моделирование процессов проектирования, т.к. ее решение позволит «сместить центр тяжести от моделей сборки программного обеспечения прикладных систем из крупных функциональных блоков к моделям генерации блоков из согласованной системы онтологических паттернов их внутренних спецификаций» [12]. В упомянутом обзоре рассматриваются следующие примеры использования онтологий проектирования в рамках такого нового направления, как Ontology-Based (-Driven) Software Engineering (OBSE-ODSE, программная инженерия под управлением онтологий):

- европейский проект MOST (Marrying Ontology and Software Technology), который включает в себя как методологию, так и инструментальные средства, предполагающие интегрирование онтологий непосредственно в разработку ПО – знания, хранящиеся в онтологии, используются для создания моделей и наоборот, что позволяет осуществлять поддержку принятия решений разработчиком и обеспечивает более точное выполнение требований [62];
- база знаний KOntoR, содержащая семантическую информацию о различных инструментах разработки (в том числе модулях и библиотеках), которая позволяет извлекать из нее новые факты, тем самым повышая их ценность для повторного использования [63];
- онтология ODYSSEY – формальная спецификация водопадной модели ЖЦ разработки ПО с использованием формализмов дескрипционной логики [64];
- DKDOnto – онтология, обеспечивающая поддержку распределенных программных проектов путем извлечения лучших практик, применимых в похожей конфигурации проекта [65];
- OnToology – проект, позволяющий автоматизировать вспомогательные действия по разработке онтологии, поддерживает непрерывную интеграцию

онтологии и объединяет деятельность по работе с онтологией с практиками по разработке ПО с помощью системы управления версиями Git [66].

В.Ф. Хорошевский заключает, что «современная постановка задачи автоматизации проектирования систем ПО предполагает наличие адекватных моделей всего ЖЦ, включая технологии разработки и реализации, модели архитектуры систем и их программных компонент, а также модели генерации ПО и модели ПрО, в которых предполагается функционирование соответствующих систем» [12].

Заключение 1.4. Развитие технологий онтологического моделирования, нацеленное на повышение эффективности процесса разработки ПО, приводит к абстрагированию от объектов и процессов автоматизации, в том числе, когда объектом проектирования выступает АС; данное явление нацеливает на создание таких средств онтологического моделирования, которые обеспечивали бы доминирование сущностей объектов и процессов автоматизации в проектных спецификациях всех этапах проектирования.

1.4 Тематико-аналитический обзор подходов к онтологическому моделированию и способов построения онтологий

1.4.1 Онтологическое моделирование

Онтологическая модель есть «концептуализированное представление информации о какой-либо области реальности, представленное в электронном виде» [42]. По мнению С. Горшкова, построение онтологической модели может происходить в несколько этапов:

1. Выделение объектов – мысленных образов, соответствующих нашим представлениям о тех или иных элементах реальности (некоторые исследователи называют их понятиями). Данный когнитивный процесс является первым в ходе построения информационной модели – иначе его можно назвать декомпозицией. Глубина декомпозиции, а также количество объектов, которые необходимо

включить в онтологическую модель, определяется прагматикой, т.е. конкретным практическим назначением модели.

2. Идентификация объектов – генерация уникальных обозначений для всех понятий, выявленных на предыдущих этапах.

3. Классификация объектов и построение иерархии классов по определенному принципу. Глубина классификации также определяется прагматикой.

4. Описание свойств объектов. Такие свойства могут быть двух типов: свойства-литералы (такие свойства служат для описания характеристик объектов) и свойства-связи (такие свойства служат для описания связей между объектами). На значения, которые могут принимать те или иные свойства, можно накладывать следующие ограничения: тип принимаемого значения; диапазон возможных значений; количество возможных значений; обязательность.

1.4.2 Семантические технологии для онтологического инжиниринга

Семантические технологии представляют собой средства, позволяющие материализовать в машинно-читаемом виде концептуальные модели, основанные на знаниях, а также обрабатывать ее с помощью вычислительных машин.

RDF (Resource Description Framework) – разработанная консорциумом Всемирной паутины модель для представления данных в виде утверждений-троек (триплетов): субъект – предикат – объект, «описывающих направленную связь от субъекта к объекту» [67].

RDFS (RDF Schema) – это уже готовый набор классов и свойств на основе модели RDF, который представляет собой основу для описания онтологий.

Некоторые компоненты RDF и RDFS нашли свое воплощение в языке описания онтологий OWL (Web Ontology Language), который уже позволяет описывать более сложные взаимоотношения классов и свойств онтологии. «OWL имеет несколько «диалектов»: редко используемый, очень упрощенный OWL Lite, наиболее часто применяемый OWL DL (Description Logic), и богатый по

выразительным возможностям OWL Full. Одно из различий между OWL DL и OWL Full состоит в том, что для OWL DL гарантируется вычислимость любого логического выражения, а для OWL Full – нет» [42].

ПО для работы с языками RDF, RDFS и OWL представляет собой различные редакторы, визуализаторы и машины логического вывода. Чаще всего данные инструменты объединены в приложения для работы с онтологиями. Приведем примеры таких приложений.

Наиболее известной средой для онтологического инжиниринга, базирующаяся на стандартах W3C (Консорциум Всемирной паутины) является инструмент Protégé [68]. Данный инструмент является разработкой Стэндфордского университета – он работает как локальная свободно распространяемая программа, а также как веб-сервис. Инструмент имеет графический интерфейс, удобный для использования неопытными пользователями, снабжен справками и примерами; в настоящее время разработано большое количество дополнительных модулей и плагинов для работы с Protégé, в том числе модуль генерации кода на Java на основе онтологической модели однако, как отмечают авторы статьи [69], которые решают задачу генерации кода для обмена информации между IoT-устройствами, структура кода, генерируемая данным модулем (классы онтологии трансформируются в классы программы), не подходит для многих прикладных систем.

Помимо приложений, существуют программные продукты класса Triple store (хранилища триплетов), функционирующие аналогично базам данных, такие как Apache Jena, Virtuoso, Sesame, GraphDB и другие.

Кроме того, существует стандарт SPARQL – язык запросов к данным, представленных в виде онтологических моделей и материализованных с помощью описанных выше стандартов, а также протокол для передачи этих запросов и ответов на них. Данный язык также является рекомендацией консорциума W3C.

Альтернативой RDF и родственным ему форматам хранения онтологий является активно развивающаяся на сегодняшний день модель Labeled Property Graph (LPG) – ориентированный граф, в котором вершины соответствуют

«записям», а ребра – «связям» между ними. Дополнительно и ребра, и связи могут быть снабжены скалярными атрибутами, а также специальными метками (лейблами). Наиболее известной, а также, по мнению В.И. Городецкого [122], одной из наиболее зрелых и мощных СУБД, работающих с моделью Labeled Property Graph, является Neo4j [70].

1.4.3 Сравнительный анализ инструментальных средств для онтологического инжиниринга

Проведем сравнительный анализ инструментальных средств по следующим базовым характеристикам: формат импорта и экспорта; формат хранения онтологии; поддержка проверки на непротиворечивость; наличие механизмов логического вывода; поддержка визуализации; оперативная расширяемость; многопользовательский режим; а также по характеристикам, находящимся в фокусе данного исследования:

1) возможность связывать единицы проектной онтологии с результатами активностей, проводимых в рамках концептуального проектирования (КП) АС, в том числе программно;
2) поддержка интеграции нескольких пространств имен;
3) принятый способ систематизации концептов.

С позиции вышеперечисленных характеристик проведем сравнительный анализ наиболее известных и широко используемых инструментариев, разработанных специально для построения онтологий, таких как Protégé, Apollo, SWOOP, IsaViz, OilEd [71], OWLGrEd [72], Fluent Editor [73], модуль Owlready2 [74] для языка программирования Python, включив в этот список инструментарий OwnWIQA [75], а также графовую СУБД Neo4j. Полезные обзоры на эту тему представлен в статьях [76-81]. Результат анализа представлен в таблице 1.1.

Таблица 1.1 - Сравнение существующих инструментариев построения онтологий

Характеристика	Protégé	Apollo	WIQA	SWOOP	IsaViz	OWLGrEd	Fluent Editor	OliEd	Owlready 2	Neo4j
Формат импорта	XML, RDF(S), OWL, XML Schema	OCML	XML	RDF(S), OIL, DAML	XSLT, RDF(S), OIL, DAML+ OIL, OWL	XML, OWL	XML, RDF(S), OWL, HTML, ENCNL	RDF(S), OIL, DAML+ OIL, SHIQ	OWL	RDF(S)
Формат экспорта	XML, RDF(S), OWL, XML Schema	OCML	XML	RDF(S), OIL, DAML	XSLT, RDF(S), OIL, DAML+ OIL, OWL	XML, OWL, SVG	XML, RDF(S), OWL, HTML, ENCNL	RDF(S), OIL, DAML+ OIL, SHIQ	OWL	JSON
Формат хранения онтологий	Файлы, БД	Файлы	Файлы, БД	Файлы	Файлы	Файлы	Файлы, БД	Файлы	Файлы	Файлы, БД
Поддержка проверки на непротиворечивость	+	+	-	-	+	-	+	+	+	+
Наличие механизмов логического вывода	+	-	+	-	+	-	+	+	+	+
Поддержка визуализации	+	-	+	-	+	+	+	-	-	+

Расширяемость	+	-	+	-	-	-	-	-		+
Многопользовательский режим	+	-	+	-	-	-	-	-	-	+
Возможность интеграции с процессом КП	-	-	+	-	-	-	-	-	+	+
Возможность связывание с материализацией	-	-	+	-	-	+/-	-	-	+/-	+
Поддержка нескольких пространств имен	+/-		-			-	+/-		+/-	+
Способ систематизации	Классификация	Классификация	Агрегация	Классификация	Классификация	Классификация	Классификация	Классификация	Классификация	Агрегация

Преимуществом редактора онтологий, встроенного в среду автоматизированного проектирования WIQA, является то, что он позволяет связывать элементы онтологии с их материализацией в разрабатываемом проекте, а также использовать онтологию в одной среде с другими активностями проектировщика (такими, как работа с документацией, фиксирование и систематизация рассуждений, построение концептуальных рисунков и схем и др.) – без необходимости экспортировать онтологию из внешнего инструмента или связывать ее со средой проектирования дополнительно, что также создает возможность оперативной изменяемости проектной онтологии непосредственно в процессе ее использования. Однако, как и многие другие средства онтологического инжиниринга, не поддерживает интеграцию нескольких пространств имен, что является актуальным для нашего исследования.

Некоторые инструменты, в частности Protégé, поддерживают мультиязычность онтологий, что является схожей задачей в рамках онтологического инжиниринга, однако данное свойство не в полной мере отвечает нашим требованиям.

Инструмент Neo4j, на наш взгляд, обеспечивает наибольшую гибкость в построении онтологии, а также наилучшим образом удовлетворяет целям и задачам данного исследования. Кроме того, в работах [82 и 121-122] отмечаются преимущества использования LPG-модели для реализации онтологических моделей перед общепринятыми RDF, среди которых – возможность отслеживания истории изменений; значительно более высокая скорость выполнения запросов к базе благодаря структуре данных, свободных от индексов; более высокие выразительные возможности благодаря возможности задавать неограниченное количество свойств-атрибутов как вершинам, так и дугам графа; многообразие типов и структур данных и др.

Что касается реализации правил логического вывода, описываемых на языке SWRL, являющегося традиционным для стандартов W3C, то при работе с LPG-схемой они легко трансформируются в набор запросов к СУБД Neo4j

на языке Cypher [120]. Аналогичная трансформация возможна и для языка SPARQL, работающего с данными по моделям RDF, однако, как отмечено в [42], для ее осуществления необходимо писать очень сложные запросы, что обусловлено моделью представления данных, поэтому данная практика не распространена: «машины логического вывода позволяют оперировать именами классов, свойств и сущностей, и «задавать модели вопросы», абстрагируя пользователя от подробностей внутреннего строения модели». Модель LPG, на которой базируется СУБД Neo4j, в силу своей организации, позволяет оперировать именами классов, свойств и сущностей уже на уровне запросов. Кроме того, СУБД Neo4j реализует функцию логического вывода за счет официального плагина Neosemantics [83], разработанного командой Neo4j Labs.

Заключение 1.5. Модель данных типа Labeled Property Graph позволяет организовывать хранилища онтологического типа, позволяя при этом, в силу своей гибкости, поддерживать несколько пространств имен концептов и отношений онтологии, отражающих структуру и поведение объекта проектирования, и формировать пространство агрегатов таких концептов с помощью меток (в отличие от модели RDF, являющейся частью концепции семантической паутины, и родственных ей), что наилучшим образом отвечает целям и задачам диссертационного исследования.

1.5 Методология проектного мышления и ее место в разработке АС

Разрабатываемая методика онтологической поддержки проектирования и обслуживающий ее инструментарий предполагает активное использование механизмов автоматизированного проектного мышления, что обусловлено следующим: данная методология направлена на достижение необходимого и достаточного понимания проектной задачи и, следовательно, позволяет сфокусироваться на ее семантической составляющей на всех стадиях ее решения. Применительно к предметной области «проектирование

автоматизированных систем» такая направленность обеспечивает доминирование сущностей объекта проектирования в проектный процессе и разрабатываемых проектных решениях, что, как было отмечено во введении, является актуальной задачей.

Термин «проектное мышление» или «дизайн-мышление» (Design Thinking) применяется в различных областях науки в разнообразных смыслах. Одним из его дефиниций является способность человека находить совершенно новые решения в известной ситуации, которая при этом содержит существенные неопределенности. Особенно широкое распространение эта методология получила в сфере информационных технологий, а также среди управленцев.

Когда речь идет об использовании проектного мышления в проектировании, оно понимается как соответствующая методология (подход или дисциплина), которая поддерживает специализированный процесс, реализация которого направлена на поиск желаемого значения [84]. В настоящее время существует множество подходов к построению рассуждений с использованием методологии проектного мышления: специалисты выделяют разные уровни и фазы проектного мышления, а также описывают всевозможные виды активностей, которые могут осуществляться в рамках этого процесса.

Согласно модели, предложенной в 1990-х годах учеными Института дизайна Хассо Платтнера в Стэнфорде, и развитой специалистами компании IDEO выделяют 3 фазы дизайн-мышления (понимание, исследование и материализация), которые, в свою очередь, делятся на 6 основных этапов (см. рис. 1.4 [85]):

1) **emphasize** – формирование так называемых дискурсов (утверждений, рассуждений, состоящих из 2-3 предложения) по поводу стоящей перед проектировщиком задачи и выражающих интересы стейкхолдеров проекта; данный этап можно сопоставить с этапом анализа требований;

2) **define** – определение проблемы на основе дискурсов, полученных на предыдущем этапе, и формулирование задачи;

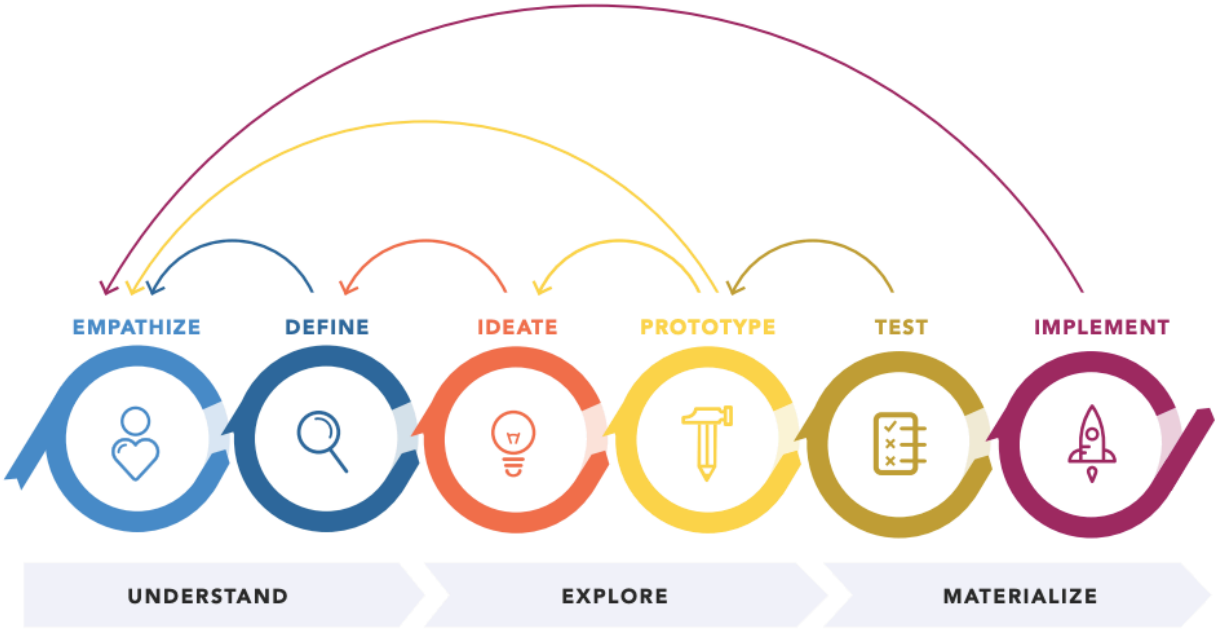


Рисунок 1.4 - Этапы дизайн-мышления

3) **ideate** – поиск решения, формирование идей о том, как можно решить поставленную задачу;

4) **prototype** – создание прототипов на основе сгенерированных на предыдущем этапе идей и выбор наилучшего решения;

5) **test** – проверка, насколько выбранный прототип отвечает требованиям, материализованных в дискурсах, полученных на первом этапе;

6) за прохождением пяти основных этапов дизайн-мышления следует внедрение (**implementation**).

В основе первых трех этапов реализации DT-подхода лежит активное использование рассуждений, порождение текстовых единиц и их анализ, которые требуют контролированного употребления языка решаемых задач, в том числе и для достижения необходимого понимания.

Для исследования особо принципиальны особенности Design Thinking при применении этого подхода в проектном процессе. Такие особенности раскрываются в работах [21-25]:

- в публикации [21] приводится версия рефлективных отображений составляющих процесса проектирования на семантику, которая является частью рефлективного подхода к принятию решений в процессе проектирования и призвана снизить негативное влияние человеческого фактора на процесс проектирования;
- в работе [22] описывается модель, которая использует DT-подход в рамках методологии управления проектами Scrum – данная модель применяется в некоторых академических проектных командах и показывает хорошие результаты.

Заключение 1.6. Внедрение технологии дизайн-мышления в проектный процесс наряду с технологией онтологического моделирования обладает значительным потенциалом для обеспечения концептуального единства разрабатываемых проектных решений. Поддержка прототипирования, тестирования и реализации на основе онтологического моделирования в случае создания АС наиболее эффективна при наличии средств автоматической генерации кода.

1.6 Методы и средства поддержки генерации кода

Как справедливо отмечает С. Горшков в [86], сегодня «исполняемый код большинства приложений представляет собой сложно организованную иерархию специализированных и достаточно абстрактных алгоритмов, порядок применения которых в значительной мере определяется настройками». К настройкам в этом смысле можно отнести как данные, используемые программой, так и свойства объекта автоматизации, если речь идет о разработке АС. Такое понимание исполняемого кода приводит к появлению технологий автоматической его генерации, которая, во-первых, снижает количество ошибок в нем, вызванных человеческим фактором, а во-вторых, в конечном позволяет снизить информационный разрыв между

концептуальной моделью системы и ее реализацией. Рассмотрим распространенные на сегодняшний день методы и средства генерации кода.

Разработка, управляемая моделями (***MDD***, model-driven development или ***MDE***, model-driven engineering) [87] – подход к кодогенерации, который предполагает рассмотрение моделей, как основных артефактов разработки, на основе которых генерируется код.

В данном случае под моделью понимается абстрактное описание системы, целью которого является представление ее в некоем упрощенном виде. Для кодогенерации модель становится пригодной в том случае, если она представляет информацию в форме, ориентированной на обработку инструментами автоматизации.

Одной из наиболее известных MDE-концепций является концепция ***MDA*** (Model Driven Architecture, архитектура, управляемая моделью) [88]. Она представляет собой модельно-ориентированный подход к разработке ПО, разработанный консорциумом OMG (Object Management Group), занимающимся разработкой и продвижением объектно-ориентированных технологий и стандартов. Суть концепции состоит в построении абстрактной формально точной метамодели управления и обмена метаданными (моделями) с помощью технологии моделирования MOF (Meta Object Facility) и задании способов ее трансформации в поддерживаемые технологии программирования. Методика разработки сводится к следующим шагам:

- разработка PIM (Platform Independent Model) – модели ПрО проектируемой системы, независимой от применяемых технологий программной инженерии;
- трансформация PIM в PSM (Platform Specifical Model) – платформо-зависимую модель;
- перевод PSM в исходный код на соответствующем языке программирования.

Языково-ориентированное программирование (ЯОП, Language Oriented Programming) – это еще одна парадигма, ориентированная на автоматическую генерацию высокоуровнего кода. Иначе ЯОП называют расходящейся разработкой (middle out development) или разработкой, опирающаяся на предметно-ориентированный язык ***DSL*** (domain-specific language) [89]. Ее суть заключается в описании решении задач с использованием DSL, направленных на отделение низкоуровневой функциональности от высокоуровневой (человеко-ориентированной), что в целом служит повышению абстрактности кода и обеспечению эффективности человеко-компьютерного взаимодействия. В основе подхода лежит идея о том, что язык, специально разработанный под задачу (предметную область), обеспечивает более высокие показатели качества кода, чем язык общего назначения, и что для решения сложных промышленных задач более эффективным будет изобрести более простой в понимании язык, нежели преодолевать трудности использования имеющегося, даже укоренившегося в промышленности.

Существуют примеры языков общего назначения, применяемых в качестве DSL для некоторых задач, и, наоборот, предметно-ориентированных языков, применяемых в качестве языков общего назначения. Так, например, язык БНФ (и компилятор с него Lex/Yacc) – это одновременно и пример метаязыка, и языка, предназначенного для конкретной задачи (описания синтаксиса ЯП).

Среди популярных предметно-ориентированных языков:

- AutoLisp для компьютерного моделирования (САПР);
- TEX для компьютерной верстки текстовых документов;
- Perl для манипулирования текстами;
- SQL для СУБД;
- Mathematica и Maple для символьных вычислений и др.

В известных прецедентах автоматизации генерации с применением онтологий [69, 90-91] рассматриваются только спецификации онтологии реализации вне связи с онтологиями предыдущих этапов проектирования.

Заключение 1.7. Эффективное использование онтологического моделирования на этапах реализации АС требует разработки средств автоматической генерации кода из онтологических спецификаций.

Выводы по главе 1

Перечислим основные заключения, сформулированные в главе 1:

Заключение 1.1. Применение технологии онтологического моделирования является одним из способов повышения степени единообразия описания и интерпретации разнородных данных, что, в свою очередь, способствует более широкому внедрению в производственный процесс современной технологии CALS, уже доказавшей свою эффективность. Для обеспечения этой возможности необходимо создание соответствующих методов и средств.

Заключение 1.2. Между фазами проектного процесса всегда наблюдается семантический (информационный) разрыв, и чем он больше, тем к большему количеству негативных последствий он приводит, тем самым, влияя на успешность разработок. Онтологическое моделирование является эффективным средством преодоления такого разрыва при условии, что созданы соответствующие инструменты.

Заключение 1.3. Развитие технологии онтологического моделирования является важнейшим направлением решения сложных аналитических и оперативных задач, к классу которых относится разработка АС. Онтологическое моделирование обеспечивает снижение сложности проектирования многокомпонентных АС со сложным поведением с многообразными сценариями поведением при условии создания соответствующих инструментальных средств. В силу своей направленности

(формализация знаний) применение технологии онтологического моделирования может привести к существенному сокращению семантического разрыва между опытом проектировщика и набором проектных ситуаций, который порождает ошибочные ассоциации и, следовательно, ведет к нарушению концептуальной целостности.

Заключение 1.4. Развитие технологий онтологического моделирования, нацеленное на повышение эффективности процесса разработки ПО, приводит к абстрагированию от объектов и процессов автоматизации, в том числе когда объектом проектирования выступает АС; данное явление нацеливает на создание таких средств онтологического моделирования, которые обеспечивали бы доминирование сущностей объектов и процессов автоматизации в проектных спецификациях всех этапах проектирования.

Заключение 1.5. Модель данных типа Labeled Property Graph позволяет организовывать хранилища онтологического типа, позволяя при этом, в силу своей гибкости, поддерживать несколько пространств имен концептов и отношений онтологии, отражающих структуру и поведение объекта проектирования, и формировать пространство агрегатов таких концептов (в отличие от модели RDF, являющейся частью концепции семантической паутины, и родственных ей), что наилучшим образом отвечает целям и задачам диссертационного исследования.

Заключение 1.6. Внедрение технологии дизайн-мышления в проектный процесс наряду с технологией онтологического моделирования обладает значительным потенциалом для обеспечения концептуального единства разрабатываемых проектных решений. Поддержка прототипирования, тестирования и реализации на основе онтологического моделирования в случае создания АС наиболее эффективна при наличии средств автоматической генерации кода.

Заключение 1.7. Эффективное использование онтологического моделирования на этапах реализации АС требует разработки средств автоматической генерации кода из онтологических спецификаций.

Глава 2. Подход к онтологической поддержке проектирования и обслуживающая его модель проектного процесса

В данной главе представлен подход к онтологической поддержке проектирования, предлагаемый в рамках диссертационного исследования, в формате системы положений, снабженных соответствующей аргументацией и нацеленных на достижение позитивных эффектов в рамках поставленной цели исследования. Описана модель проектного процесса разработки АС с применением онтологической поддержки, а также раскрыты возможности применения технологии онтологического моделирования в инфраструктуре удовлетворения потребительских запросов в рамках производственного процесса.

2.1 Разработка подхода к онтологической поддержке проектирования

2.1.1 DT-подход, как основа процесса формирования онтологических моделей

Как уже было отмечено ранее, одной из составляющих новизны предлагаемых проектных решений является реализация механизмов проектного мышления в автоматизированном проектировании с применением технологий онтологического моделирования на основе инструментария моделирования рассуждений, которое, согласно **Заключению 1.5**, может способствовать появлению значительных положительных эффектов в проектировании АС.

Сформулируем соответствующее положение:

Положение 1. **В основе автоматизируемого процесса разработки АС лежат механизмы проектного мышления (Design Thinking) и вопросно-ответного моделирования, обслуживающие процесс**

моделирования рассуждений и активно вовлекающие в проектный процесс онтологические модели. Последовательная реализация данных механизмов способствует снижению неопределенности проектной ситуации и достижению проектировщиком необходимого и достаточного уровня понимания стоящей перед ним задачи, а также направлена на обеспечение концептуальной целостности, непротиворечивости и полноты разрабатываемых проектных решений.

Первый этап DT-подхода получил название ***Empathize*** (эмпатия, сопереживание), т.к. его целью является определение интересов будущих потребителей продуктов труда проектировщика.

Как описывается в [92], когда проектировщик обнаруживает новую задачу ***Z*** в проектной ситуации ***St(Z, t_0)***, реализация DT-подхода начинается с ***регистрации данной ситуации в вербальной форме*** (например, в виде списка ключевых слов), выражающей исходную неопределенность ***$\nabla U(Z, t_0)$*** ее восприятия.

При наличии такого списка ключевых слов у проектировщика появляется возможность ***формирования совокупности дискурсов*** – связной совокупности коротких текстов на естественном языке, описывающих потребности будущих благополучателей проекта (стейкхолдеров): данный этап соответствует этапу анализа требований в традиционной парадигме разработки АС, однако отличается от него фокусом на мотивах и целях стейкхолдеров. Отметим, что создание дискурсов приводят к результатам ***$\nabla U(Z, t_1)$***, неопределенность которых будет меньше исходной ***$\nabla U(Z, t_0)$***.

Данные дискурсы, продукты их последовательного анализа (для детализации рассуждений предлагается использовать технологию вопросно-ответного или QA-анализа [93]), а также традиционное представление требований в виде проектной документации материализуются а набор текстов на естественно-профессиональном языке и являются проявлением ***языка проекта $L^P(t)$***. В силу своей гетерогенности язык проекта усложняет

интерпретируемость и согласованность проектных спецификаций, в связи с чем целесообразным видится представлять его ядро в форме ***онтологии проекта*** $O^P(t)$, которая, в силу своей структуры, а также возможности осуществлять логический вывод над собственными сущностями и, таким образом, порождать новые сущности, обеспечивает возможность производить согласованную трансформацию проектных решений.

На основе вышесказанного сформулируем следующие положения, нашего подхода.

Положение 1.1. **Изначально в ходе реализации DT-подхода формулируются дискурсы, отражающие потребности будущих потребителей АС (требования) и проводится их последовательный вопросно-ответный анализ для достижения требуемого уровня детализации.**

Положение 1.2. **Сформулированные дискурсы и продукты их анализа являются проявлением формируемого в ходе проектного процесса языка проекта, гетерогенность которого, связанная с разнообразием проектных задач и вовлекаемых в них сущностей, усложняет интерпретируемость и согласованность проектных спецификаций, что служит основанием для выбора онтологии проекта в качестве основы для представления знаний в проекте. При этом в пространство сущностей онтологии включаются отношения, обеспечивающие согласованную трансформацию одних спецификаций проектных решений в другие в ходе последовательной детализации проектных решений.**

Следующий этап DT-подхода – ***Define*** (от англ. «определить») – нацелен на ***понимание*** проектировщиком задачи в текущем состоянии ее постановки.

Неуспехи в реализации проектов АС, как и любых других сложных проектов, в подавляющем большинстве случаев связаны с проявлениями человеческого фактора. Человеку свойственно совершать ошибки, а их появление зачастую связано с ***феноменом понимания***, отвечающий за и

проявляющийся в корректности употребления языка. Если у проектировщика не сложилось достаточное понимание задачи, которая перед ним стоит, и, тем более если, оно отсутствует, то он неизбежно будет совершать ошибки в ходе своей работы. В то же время, если необходимое понимание сформировано, то вероятность появления ошибок снижается.

Кроме того, для того чтобы понимание можно было «передать» другим членам команды, которые работают над проектом, или иным заинтересованных лицам, а также для обеспечения возможности его повторного использования, необходимо регистрировать акты понимания и связывать их в системные образования. Наиболее удобной формой регистрации понимания, исходя из опыта проектирования, представляется вербально-графическая форма, т.е. тексты рассуждений проектировщика, а также схемы и рисунки, которые появились по ходу его рассуждений.

Чтобы достичь необходимого и достаточного уровня понимания, проектировщик формулирует и формулирует постановки проектных задач, выражая сущность их содержимого с соответствующей неопределенностью и подвергая его вопросно-ответному анализу, который позволяет ассоциировать опыт проектирования аналогичных объектов с текущими проектными задачами. На этом этапе осуществляется активное мысленное воображение (mental imagery), которое включает способность проектировщика создавать и использовать образы в процессе понимания.

Положение 1.3. **Дискурсы и продукты их анализа в ходе реализации DT-подхода подвергаются вопросно-ответному моделированию с целью достичь такого уровня понимания состава и содержания проектных задач, который являлся бы необходимым и достаточным для перехода к фазе поиска способов их решения активным использованием опыта проектирования аналогичных объектов. При этом вопросно-ответное моделирование обеспечивает ассоциирование текущих проектных задач с единицами опыта.**

Положение 1.4. **Понимание проектировщика материализуется в соответствующие постановки задач, которые через вопросно-ответное моделирование порождает метаданные онтологических спецификаций, обеспечивающие ассоциирование сущностей дискурсов и содержания задач с сущностями, которые поддерживают осуществление способов решения задач.**

Положение 1.5. **Постановки задачи (как промежуточные, так и целевая), а также результаты их анализа являются источником новых сущностей онтологической модели проекта, при чем связываемых с сущностями предшествующих этапов проектирования.**

На третьем этапе ***Ideate*** проектировщик пытается придумать подходящую идею для решения задачи. Найденная идея представляет собой ответ на вопрос «Как решить задачу?» Этот ответ будет продолжать уменьшать неопределенность постановки задачи. Следует подчеркнуть, что действия этого шага основаны на интеллектуальных способностях проектировщика, среди которых принципиально важны интуиция и воображение в контексте использования языковых средств. Этот шаг также ведет к уменьшению неопределенности описания задачи.

Положение 1.6. **Спецификации постановок задач в ходе реализации DT-подхода образуют основу для порождения идеи решения поставленной задачи, что позволяет сформировать концепцию проекта АС, т.е. определить архитектурные решения, общую структурно-функциональную организацию и общие аспекты поведения, что фиксируется в спецификациях онтологии проекта.**

Шаги ***Prototype*** и ***Test*** предназначены для предварительного построения решения задачи, которое соответствует изобретенной идее и имеет проверяемую форму для его тестирования.

Положение 1.7. **Спецификации онтологии проекта организованы таким образом, что могут быть трансформированы в тестируемый**

прототип АС, создание которого предусмотрено рамками DT-подхода. При этом связывание сущностей такого прототипа с онтологическими спецификациями порождает свойство «быть материализованным» понятий онтологии на мета-уровне.

Как было отмечено ранее, по ходу и вследствие проектных рассуждений формируется ***язык проекта*** – таким образом, каждая единица данного языка оказывается неизбежно связанной с содержанием проекта и, следовательно, находит свое овеществление в его процессе и результатах.

Отметим, что проектные рассуждения не только служат «строительным материалом» для языка проекта – они еще и являются отражением процесса проектирования, т.е. рассуждения и выводы, сформулированные на том или ином этапе определяют последующие. Соответственно, если на ранних этапах были допущены ошибки, то они сказываются и на последующих этапах.

С учетом вышесказанного справедливо утверждать, что если тексты рассуждений и проектные документы служат основой для создания языка проекта, то язык проекта играет основную управляющую роль в концептуальной активности проектировщика. А сопоставление текущего состояния его материализации (т.е. онтологии проекта) с новыми рассуждениями и проектными решениями позволяет обнаруживать и предотвращать возможные ошибки в них.

На этом основании сформулируем еще одно положение.

Положение 1.8. **Проектный процесс должен управляться онтологией проекта, а рассуждения проектировщика и разрабатываемые проектных решения на всех этапах проектирования вплоть до реализации – подвергаться онтологическому специфицированию.**

Целесообразно разработать инструмент, который позволял бы автоматически или автоматизировано наполнять онтологию сущностями. Поскольку нам приходится иметь дело с текстами, написанными на

естественном языке, для этого могут быть использованы различные средства логико-лингвистической обработки – такие как, например, морфологические и синтаксические анализаторы, парсеры и другие.

Принимая во внимание тот факт, что в силу сложности, многозначности и других свойств естественного языка ни одно из существующих средств для анализа текстов не обеспечивает стопроцентной точности, можно сделать вывод о том, что конечное решение о включении в онтологию тех или иных единиц предстоит принимать проектировщику, но при этом часть работ целесообразно передать компьютеру, что существенно облегчит ему работу и позволит снизить количество затрачиваемых на нее ресурсов.

На основе вышесказанного сформируем еще одно положение:

Положение 1.9. **В ходе решения задач проектировщик должен принимать решение о включении в онтологию проекта новых сущностей исходя из того, найдут ли они свое овеществление в проекте, опираясь при этом на предлагаемые решения используемых программных средств логико-лингвистической обработки.**

Обобщим рассуждения: проектная задача, с которой сталкивается проектировщик в ходе своей деятельности, обладает существенными неопределенностями, которые снижаются в процессе перехода от одной фазы DT к другой, в том числе за счет регистрации понимания проектной задачи, то есть описания результатов ее понимания на языке проекта, который материализуется в форме онтологии проекта. Онтология проекта, в свою очередь, включает в себя понятия, без которых обеспечение понимания и повторное использование его зарегистрированных следов существенно затрудняется.

Описанный выше процесс схематически изображен на рис. 2.1.

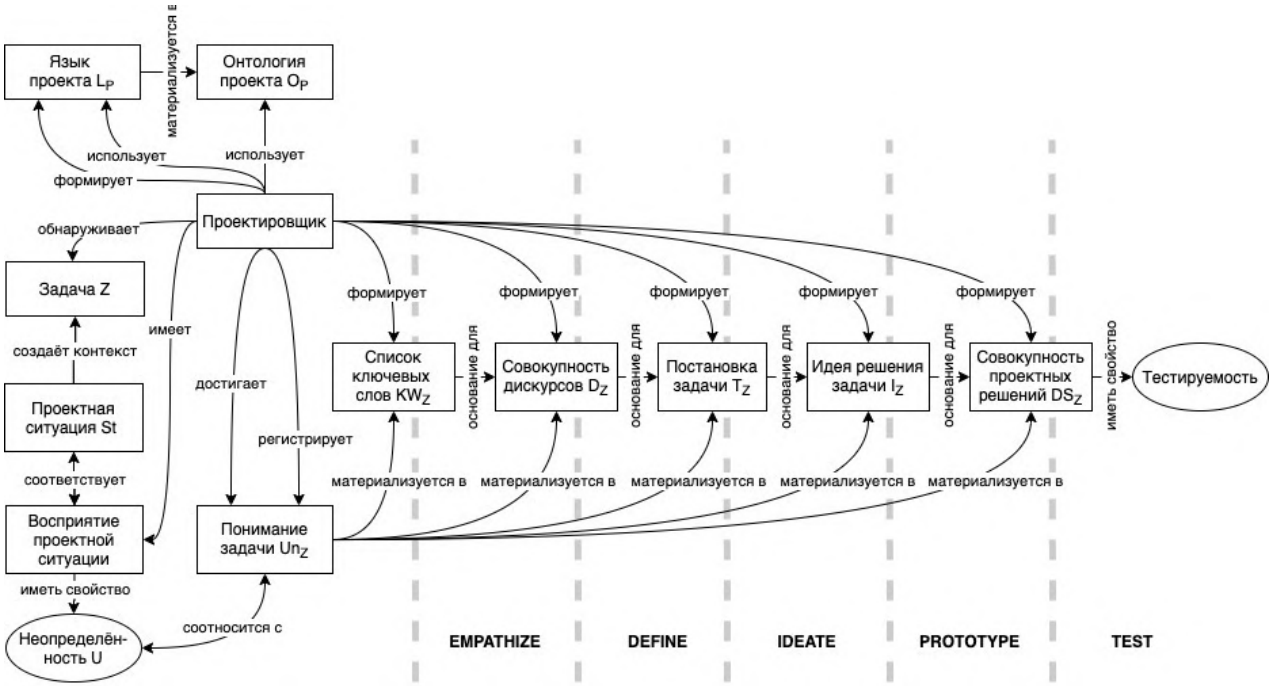


Рисунок 2.1 - Парадигма Design Thinking в автоматизированном проектировании АС

Отметим, что данное понимание проектного процесса достигнуто в результате длительных исследований и цепочки рассуждений, результаты которых представлены в Приложениях 1 и 2 в формате вопросно-ответных протоколов, а также в Приложении 3 в формате мотивационно-целевого анализа.

2.1.2 Онтологическая модель проекта, как система связанных онтологий

С учетом накопленного исследователями и специалистами в области АП опыта применения онтологий проектирования, примеры которого представлены в параграфе 1.3, справедливо утверждать следующее.

Процесс проектирования АС представляет собой последовательность проектных действий, каждое из которых модифицирует часть проектных спецификаций. Эти спецификации в различных проектных задачах

существенно различаются по своей структуре и зачастую по форме представления. Обеспечение непротиворечивости, полноты и концептуальной целостности в этих условиях приходится в большинстве случаев осуществлять вручную, и в этой работе интерпретатором спецификаций является проектировщик. Автоматизация согласования проектных решений и обеспечение полной реализации требований к АС наилучшим образом могут быть осуществлены средствами онтологического моделирования.

Онтологические спецификации по отношению к проектным решениям играют роль метаданных, снабженных унифицированными средствами автоматической интерпретации. Унификация в обязательном порядке охватывает те сущности и отношения, которые обеспечивают взаимозависимость проектных задач и спецификаций. Именно взаимозависимость является основой для автоматизации оценки полноты, непротиворечивости и отчасти концептуальной целостности. В то же время, значительной частью работ с проектными спецификациями, в том числе и формированием онтологических спецификаций, занимаются различные специалисты. Это означает, что представление онтологических спецификаций и их интерпретация должны обслуживать синхронизацию продуктов ручного труда и информационных продуктов в условиях разделения труда между большим числом исполнителей, разбитых по рабочим группам.

Таким образом, следующее основополагающее положение предлагаемого в рамках диссертационного исследования подхода сформулировано на основании представленных выше утверждений:

Положение 2. **В целях сокращения семантических разрывов в проектном процессе, а также повышения степени автоматизации процесса онтологического моделирования онтологию проекта целесообразно представлять как многоуровневую систему онтологических моделей в согласии с традиционным пониманием разбиения проектного процесса разработки АС на этапы; при этом,**

набор этапов, результаты которых будут отражены в онтологии проекта, должен определяться с учетом особенностей конкретного проекта: в том числе требований к нему, предметной области и здравого смысла.

Для детализации идей Положения 2 сформулируем дополнительные положения.

Положение 2.1. **Для каждой из онтологических моделей, включаемых в онтологию проекта, предусмотрен собственный набор структурообразующих элементов, через которые целесообразно определить основные классы задействованных понятий, возможные типы отношений между ними, а также правила логического вывода, обслуживающих трансформацию онтологических спецификаций, а также порождение проектных спецификаций других типов на основе онтологических моделей.**

Положение 2.2. **В целях связывания сущностей онтологических моделей на разных этапах проектного процесса, а также автоматизации процесса онтологического моделирования обеспечивается специфицирование взаимообусловленности проектных решений.**

Здесь и далее в для краткости *будем называть онтологическую модель каждого этапа проектного процесса онтологией*: так, например, на этапе анализа требований может формироваться онтологическая модель требований, которую будем называть онтологией требований; на этапе разработки проектных решений – онтологическая модель, которую будем называть онтологией проектных решений и т.д.

Допустим, онтологию некоего проекта составляют онтология требований, онтология проектных решений, онтология реализации и онтология тестирования. В соответствии с Положением 1 предлагаемого подхода и производными от него, онтология требований формируются на основании системы дискурсов и продуктов их анализа. Онтология

проектирования, в свою очередь, строится уже на базе онтологии требований: т.е. сущности онтологии требований могут быть модифицированы или использованы для порождения сущностей онтологии проектирования и так далее – от уровня к уровню. Такая структурная организация ОМ приводит к двум позитивным явлениям:

- сущности онтологий каждого уровня значительно пересекаются, а доля пересечений определяет снижение трудозатрат по преодолению семантических разрывов;
- наличие конструктивной системы отношений между сущностями онтологий различных уровней позволяют подвергнуть процесс онтологического моделирования частичной автоматизации.

Как было отмечено во введении, одной из причин информационных разрывов является вовлечение в проектный процесс многих ***пространств имен*** вследствие следующих обстоятельств:

- дополнительные пространства имен появляются с каждым новым этапом работы над проектом в силу того, что для разработки тех или иных проектных решений могут использоваться различные способы представления информации, и, как следствие, одно и то же понятие может иметь различную форму выражения (текст на естественном языке, условные обозначения в различных UML-диаграммах, имена классов и переменных в исходном коде программ и др.);
- дополнительные пространства имен вовлекаются в проектный процесс в случае, если для разработки проектных решений применяются внешние прототипы (тексты рассуждений, элементы проектных решений, разработанных в рамках других проектов, части систем, подсистемы и модули с аналогичными или похожими свойствами и др.);

- практика АП такова, что работа над созданием проектных решений осуществляется разными группами специалистов, каждой из которых свойственен свой собственный язык, сформировавшийся по ходу его/их профессиональной деятельности, а также свое собственное восприятие ситуаций, в которых необходимо принимать решения относительно именования.

Все это приводит к нарушению ***концептуальной целостности*** проекта. Применение технологии онтологического моделирования является естественным решением проблемы концептуальной целостности, поскольку онтология, как средство управления знаниями, оперирует понятиями, а не словами (именами, терминами).

Положение 2.3. **Для обеспечения концептуальной целостности проекта и сокращения семантического разрыва между стадиями проектирования предусмотрена инструментальная поддержка существования многих пространств имен в онтологической модели проекта на всех стадиях работы над проектом. Кроме того, поддержка отношения взаимообусловленности в онтологических спецификациях позволяет автоматизировать выявление таких сущностей и отношений, которые не связаны с онтологическими спецификациями предыдущих этапов напрямую, что способствует выявлению обнаружения нарушений концептуальной целостности.**

Традиционное вовлечение в проектный процесс пространства прототипов обуславливает ***прецедентно-ориентированный характер*** онтологического сопровождения проектирования. Как отмечает П.И. Соснин в [123], «любой прецедент – это активность человека или группы лиц, связанная с действием, решением или поведением, осуществленным в прошлом, которая полезна как образец для повторных использований и/или оправдания повторных действий по такому образцу», а модель прецедента для конкретной проектной задачи может быть представлена в виде текста

постановки задачи (текстовая модель, $P^T(t)$), формулы логики предикатов (логическая модель, $P^L(t)$), UML-диаграммы (графическая модель, $P^G(t)$), вопросно-ответного протокола (QA-модель, $P^{QA}(t)$), исходного кода программы (поведенческая модель, $P^I(t)$), исполняемого кода программы (образец прецедента, $P^E(t)$), а также схемы, интегрирующей все специализированные модели прецедента в единое целое (интегральная модель, MP_i). В рамках подхода предлагается дополнить MP_i онтологической моделью прецедента $P^O(t)$, в формате онтологических спецификаций, ориентированных на включение в онтологию проекта.

Положение 2.5. **Вовлекаемые в проектный процесс прототипы подвергаются онтологическому специфицированию для включения в онтологию проекта в качестве единиц опыта, пригодных для повторного использования, что обуславливает прецедентно-ориентированный характер онтологического сопровождения проекта.**

В рамках подхода предлагается рассматривать онтологию не только как средство систематизации знаний, но и как инструментальное средство порождения проектных решений и других артефактов проектирования. Таким образом, смещается фокус от онтологического моделирования, как с обособленного процесса, требующего дополнительных трудозатрат, которые в некоторых случаях даже могут превосходить позитивные эффекты от применения ОМ, к онтологическому моделированию, интегрированному в проектный процесс и являющемуся дополнительным средством его автоматизации.

Так, при наличии соответствующей структуры онтология проектных решений может использоваться для автоматической генерации UML-диаграмм (например, если в онтологии предусмотрены такие классы, как состояния АС и условия перехода из одного состояния в другое, то с ее помощью может быть сгенерирована диаграмма состояний); а онтология реализации может использоваться для частичной генерации исходного кода программ.

Как было отмечено во введении, производительность труда проектировщиков может быть повышена не только за счет сокращения семантического разрыва между различными стадиями проектного процесса и обеспечения концептуальной целостности, но и за счет обеспечения непротиворечивости и полноты разрабатываемых проектных решений. Для конкретизации способов достижения указанных позитивных эффектов сформулируем дополнительные положения.

Положение 2.6. **Для обеспечения непротиворечивости разрабатываемых проектных решений в качестве структурообразующих элементов в онтологии проекта вводятся семантические отношения казуальных типов, обеспечивающих обусловленность (детерминированность) трансформаций онтологических спецификаций.**

Положение 2.7. **Для обеспечения полноты разрабатываемых проектных решений предусмотрена возможность поддержки проверки покрытия требований через предоставление информации о связывании понятий онтологии требований с понятиями онтологии реализации, в том числе доле связанных и не связанных понятий.**

2.1.3 Онтологическое моделирование, как система инкрементальных информационных процессов

Перейдем к рассмотрению непосредственно процесса онтологического моделирования. Согласно Положению 1.6, проектный процесс должен управляться онтологией проекта, а, согласно Положению 2, результаты, полученные на различных этапах проектного процесса, должны отражаться в онтологической модели проекта – следовательно, справедливо утверждать, что ОМ должна развиваться вместе с проектом, быть динамичной и оперативно отражать модификации проектных решений.

На этом основании сформулируем третье основополагающее положение предлагаемого подхода:

Положение 3. **Для построения такой онтологии проекта, которая способна управлять проектным процессом, носить динамический характер и оперативно отражать модификации разрабатываемых проектных решений и других артефактов проектирования, целесообразно организовать процесс онтологического моделирования как систему инкрементальных процессов, результатом каждого из которых является изменение состояния онтологии.**

Уточнение Положения 3 обеспечивает следующий набор положений:

Положение 3.1. **На каждом новом витке онтологического моделирования необходимо пополнять онтологию проекта новыми сущностями и/или модифицировать существующие – с опорой на формальную структуру онтологических спецификаций.**

Положение 3.2. **Новый виток онтологического моделирования необходимо запускать всякий раз, когда требуется внести изменения в требования к проекту, а также при каждом переходе к новому этапу проектного процесса.**

Положение 3.3. В **дополнение к традиционной процедуре классификации, широко применяемой при использовании семантических технологий Semantic Web, применяется процедура агрегации, использование которой определяется прагматикой исследования и контекстом применения онтологий при разработке АС. Агрегаты формируются с ориентацией на специфицирование составляющих проектного процесса проектирования АС, что обеспечивает конструктивность онтологии.**

Положение 3.4. **Для каждого этапа проектного процесса формируется собственное пространство агрегатов, классов, семантических отношений и аксиом с ориентацией на автоматизацию**

формирования проектных решений и других артефактов проектирования.

2.2 Формальная структура онтологической модели проекта и процедура ее формирования

Общепринятая модель онтологической спецификации достаточно часто представляется следующим образом:

$$O = (C, R, F), \tag{2.1}$$

где $\boldsymbol{C} = \{c_n | n \in \mathbb{N}, n \in [1, |C|]\}$ – множество концептов (или понятий),

$\boldsymbol{R} = \{(c_i, c_j, rt_n) | rt \in RT, c \in C\}$, где $\boldsymbol{RT}$ – множество типов отношений;

$\boldsymbol{F}: \boldsymbol{C} \times \boldsymbol{R}$ – множество функций интерпретации, в самом общем виде задаваемых соответствиями между $\boldsymbol{C}$ и $\boldsymbol{R}$.

Отношения между концептами, как было показано в 1.4.2, чаще всего выражаются через триплеты средствами языков спецификаций RDF, RDFS и OWL. Использование такой структуры в непосредственном виде в АП зачастую порождает очень сложные по своему составу композиции [82]. В то же время в практике АП в большей степени распространены проектные спецификации, эффективно представляемые записями баз данных, что представляет собой объектно-ориентированный подход, где любой объект проектной спецификации может характеризоваться несколькими свойствами, причем, множества объектов разбиваются на классы в соответствии с содержанием проектного процесса и свойствами объектов проектирования. Такие спецификации часто выражаются средствами SQL, LPG, Cypher и иными даталогическими инструментами, позволяющими представлять не только отношения между объектами, но и отображения с областями отправления и прибытия, определяемыми смыслом проектной процедуры или операции.

Для учета этого обстоятельства в рамках описанного выше подхода предлагается следующая расширенная, по сравнению с представленной в формуле (2.1), модель онтологической спецификации:

$$O^P = (C, R, F, Pr, Agg, A), \tag{2.2}$$

где ***F*** переопределено как множество отображений:

$$F = \{f_n : C^* \times R^* \to C^{**} \times R^{**} | C^*, C^{**} \subseteq C; R^*, R^{**} \subseteq R\} \tag{2.3}$$

Pr – множество свойств понятий и отношений:

$$Pr = \{(pr_i, e_j) | pr \in Pr, e \in C \cup R\} \tag{2.4}$$

Agg – множество агрегатов, ***A*** – множество аксиом.

Дополнительно определим структурообразующие типы концептов:

$$C = C^I \cup C^J \cup C^D \cup C^S \tag{2.5}$$

где $\boldsymbol{C^I}$ – концепты, представляющие множество этапов проектирования с целью формирования метаданных, обеспечивающих описание зависимости выходных спецификаций проектных процедур от входных;

$\boldsymbol{C^J}$ – концепты, представляющие множество проектных задач, характерных для процессов проектирования АС;

$\boldsymbol{C^D}$ – концепты, обслуживающие документирование разрабатываемой АС таким образом, чтобы рабочая документация формировалась из проектных спецификаций, представляемых в информационном обеспечении САПР;

$\boldsymbol{C^S}$ – концепты, представляющие спецификации разрабатываемой АС.

Под множествами $\boldsymbol{C^D}$ и $\boldsymbol{C^S}$ также могут скрываться классы понятий $\boldsymbol{CC^D}$ и $\boldsymbol{CC^S}$ и экземпляры таких классов $\boldsymbol{CI^S}$ и $\boldsymbol{CI^D}$, представляющие спецификации конкретной АС.

$$CC^D, CI^D \subset C^D, CC^S, CI^S \subset C^S \tag{2.6}$$

таких, что каждому экземпляру из множества ***CI*** соответствует класс из множества ***CJ***:

$$C = \{(cc_i, ci_j) | cc \in CC, ci \in CI\} \tag{2.7}$$

Аналогично формуле (2.5) могут быть определены типы отношений:

$$R = R^I \cup R^J \cup R^D \cup R^S \tag{2.8}$$

где $\boldsymbol{R^I}$ – множество отношений непосредственного следования между концептами, добавляемыми в O^P на каждом этапе проектирования;

$\boldsymbol{R^J}$ – множество отношений обусловленности между концептами, связываемые с конкретными проектными задачами,

$\boldsymbol{R^D}$ и $\boldsymbol{R^S}$ – заданы по аналогии с $\boldsymbol{C^D}$ и $\boldsymbol{C^S}$.

Процедуру формирования и/или модификации онтологической модели в обобщенном виде можно свести к следующим шагам:

1) Формирование или модификация пространства агрегатов ***Agg***. В рамках предлагаемого подхода целесообразным представляется выделение агрегатов в соответствии со следующими объектами:

- этапами проектного процесса $\boldsymbol{I}$ и производимыми на них артефактами проектирования (требования, проектные решения, реализации), подвергающимися онтологическому специфицированию:

$$a^I: I \to Agg^* \tag{2.9}$$

- проектными задачами, проектными решениями и проектными операциями (функции, реализующие порождение таких агрегатов, раскрыты в параграфе 3.1.2);
- прототипами, представляющих прецеденты $\boldsymbol{P}$ в накопленном опыте автоматизации:

$$a^P: P \to Agg^* \tag{2.10}$$

2) Формирование или модификация множества классов понятий $\boldsymbol{CC^i}$, характеризующих объект проектирования на этапе проектного процесса $\boldsymbol{i}$, конструктивность которого достигается за счет обеспечения связывания элементов множества $\boldsymbol{CI^i}$ с сущностями типичных артефактов проектирования текущего этапа $\boldsymbol{Atf^i}$ (спецификаций требований, UML-диаграмм, исходного кода программы автоматизации и др.) отношением материализации, что задается функцией:

$$m^i: CI^i \to Atf^i \tag{2.11}$$

3) Формирование или модификация аксиом $\boldsymbol{A^i}$, накладывающих ограничения на структуру онтологических спецификаций: а именно – описывающих возможные типы семантических отношений $\boldsymbol{RT^i}$ между понятиями $\boldsymbol{CI^i}$ различных классов $\boldsymbol{CC^i}$, а также возможность связывания этих классов с определенными агрегатами ***Agg***:

$$ax^R: \{(cc_n, cc_m) | cc \in CC; n, m \in \mathbb{N}; n, m \in [1, |Cl|]\} \rightarrow \{rt_p | rt \in RT, p \in \mathbb{N}, p \in [1, |RT|]\} \quad (2.12)$$

$$ax^{Agg}: CC \rightarrow Agg \quad (2.13)$$

4) Формирование множества экземпляров классов понятий $\boldsymbol{CI^i_j}$, соответствующих конкретным сущностям объекта проектирования, за счет реализации функций: $\boldsymbol{l^i}$, обеспечивающей ***связывание дискурсов*** из множества $\boldsymbol{D}$ с понятиями $\boldsymbol{C^i_j}$, и $\boldsymbol{t^i}$, обеспечивающей ***трансформацию продуктов онтологического специфицирования*** предыдущих этапов $\boldsymbol{O^{i-1}}$ и связывание их с понятиями $\boldsymbol{C^i_j}$ в соответствии с результатами вопросно-ответного моделирования $\boldsymbol{QA^i}$, а также контроль покрытия требований содержанием проектных решений:

$$l^i: D \times QA^i \rightarrow \left\{\left(cc^i_j, ci^i_j\right) | cc \in CC, ci \in CI\right\} \times \left\{\left(ci^i_j, ci^i_d, rt\right) | d = f(D), rt \in RT\right\} \quad (2.14)$$

$$t^i: O^{i-1} \times QA^i \rightarrow \left\{\left(cc^i_j, ci^i_j\right) | cc \in CC, ci \in CI\right\} \times \{(ci^{i-1}_j, ci^i_j, r) | ci \in CI\} \quad (2.15)$$

5) Описание множества свойств понятий $\boldsymbol{Pr}$ и их значений $\boldsymbol{V}$. В качестве свойств могут быть заданы, например, имена $\boldsymbol{Nm}$ сущностей $\boldsymbol{E}$ объекта проектирования, материализованные в конкретном артефакте (термины, используемый в спецификациях требований; имена объектов – например, состояний АС, – используемые для создания UML-диаграмм; имена классов, функций, параметров, используемых в исходном коде программы автоматизации):

$$p: \{(e_n, nm_n) | e \in E, nm \in Nm\} \rightarrow \{(pr_m, v_m) | pr \in Pr, v \in V\} \quad (2.16)$$

6) Описание отношений между понятиями $\boldsymbol{R}$, извлекаемых на основе дискурсов $\boldsymbol{D}$ и продуктов его анализа $\boldsymbol{QA}$ в соответствии с аксиомами $\boldsymbol{A}$.

$$r^i: D \times QA \times A \to \{(c_n, c_m, rt)\} \quad (2.17)$$

7) Агрегация сущностей. На этом этапе необходимо установить соответствие между сущностями онтологии и агрегатами в соответствии с аксиомами A^i. При этом, одно и то же понятие может относиться сразу к нескольким агрегатам.

$$agg^i: \{e^i | e \in C \cup R\} \times A^i \to \{(e^i, agg) | agg \in Agg\} \quad (2.18)$$

8) Извлечение онтологических спецификаций, пригодных для генерирования артефактов проектирования (документации, UML, исходного кода) на основе агрегатов онтологических моделей в соответствии с метаданными онтологических спецификаций:

$$g^i: C_i^D \times R_i^D \times Agg^i \to C^* \times R^* \quad (2.19)$$

2.3 Модель проектного процесса разработки АС с применением механизмов онтологической поддержки

В обобщенном виде проектный процесс с применением технологии онтологического моделирования представлен на рисунке 2.2. Он выстроен с опорой на ГОСТ 34.601, в котором определены стадии создания АС [96], а также парадигму Design Thinking; при этом, как было отмечено в предыдущем параграфе, процесс носит инкрементальный характер и на каждой стадии осуществляется взаимодействие с онтологией проекта, в ходе которого она модифицируется и/или пополняется новыми сущностями.

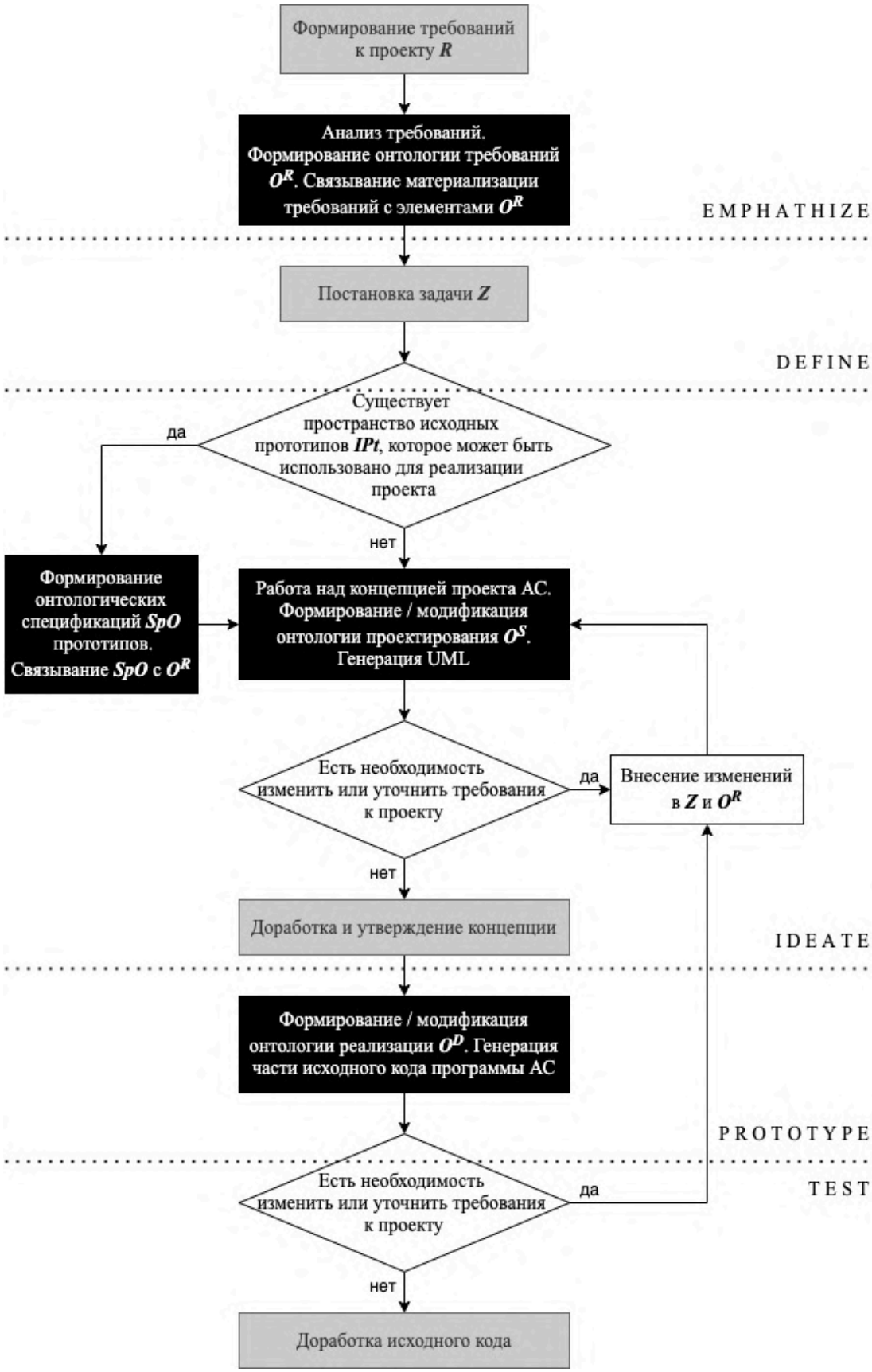


Рисунок 2.2 - Диаграмма деятельности, описывающая проектный процесс с применением технологии онтологического моделирования в парадигме Design Thinking

2.3.1 Анализ требований и формирование онтологической модели требований

Проектный процесс начинается с этапа ***формирования требований RQ*** к проекту, которые специфицируются в форме дискурсов, как обозначено в подходе. После чего каждое требование из множества ***RQ*** подвергается вопросно-ответному анализу (или QA-анализу). Результаты такого анализа служат основой для автоматизированного формирования ***онтологической модели требований*** O^R в соответствии со следующей процедурой, которая является конкретизацией части процедуры, описанной в параграфе 2.2:

1) Выявление сущностей, специфицирующих разрабатываемую АС, которые фигурируют в тексте дискурса и формирование подмножества понятий C^{S*} в соответствии с данными сущностями. Понятия заносятся в онтологию с указанием классов, к которым они принадлежат (набор классов онтологии требований задается на уровне метаданных онтологических спецификаций).

2) Описание свойств понятий ***Pr****. Задание значений свойств.

3) Описание отношений R^{S*} между понятиями C^{S*} в соответствии с аксиомами онтологии требований A^R.

Спецификации онтологической модели требований необходимо связать с материализациями требований в проекте (текстами дискурсов). Сделать это можно разными способами: например, добавить ссылки на требования или непосредственно их формулировки в качестве свойств узлов онтологии и/или сформировать подпространство агрегатов в соответствии с требованиями.

Далее следует этап ***постановки задачи Z***, в результате чего должна быть сформулирована исходная постановка задачи и набор подзадач с опорой на требования ***RQ***.

Данный подпроцесс соответствует этапам DT-подхода Emphathize и Define – он показан на рисунке 2.3 в виде диаграммы деятельности.

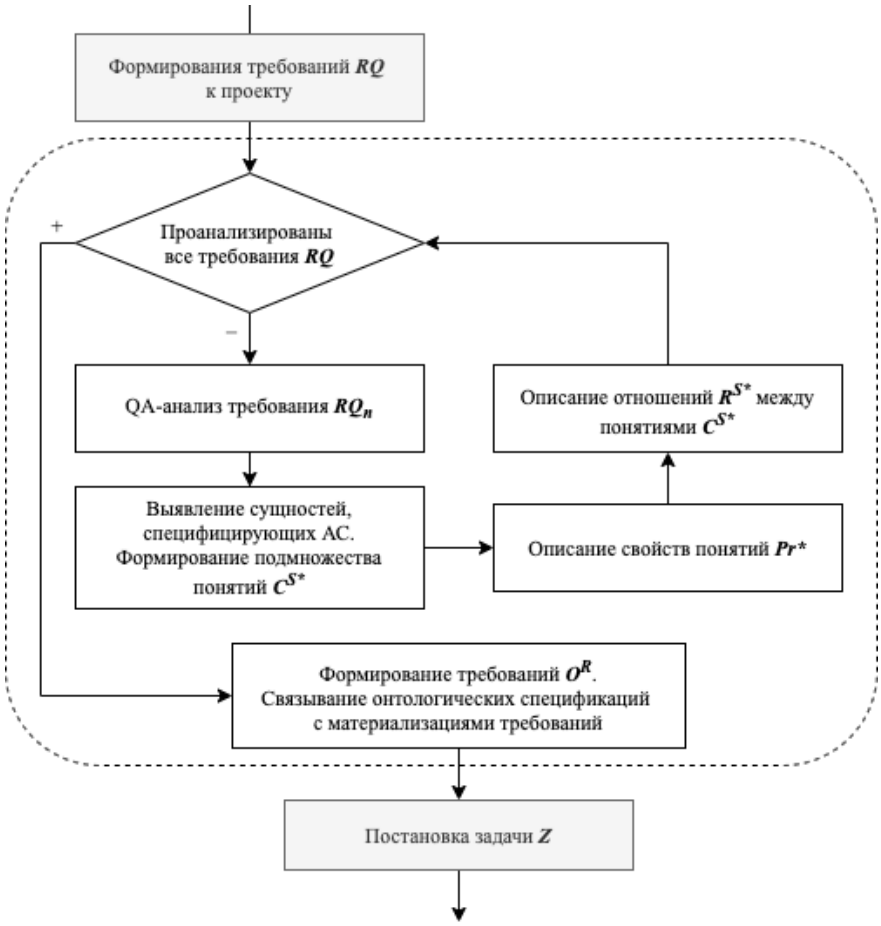


Рисунок 2.3 - Анализ требований и формирование онтологической модели АС соответствующего уровня

2.3.2 Формирования онтологической модели прецедентов

Как уже было отмечено ранее, на практике для разработки АС зачастую используются так называемые прототипы (тексты, UML-диаграммы, фрагменты кода и т.п.) – будем называть их ***исходными прототипами IPt***, которые по сути являются готовыми и подтвердившими свою состоятельность проектными решениями, концептуальные сущности которых связаны с концептуальными сущностями результирующей системы отношением аналогии. Такие прототипы представляют собой прецеденты в накопленном опыте автоматизации. Сущности прототипов могут трансформироваться в сущности результирующей системы без значительных временных затрат и

семантических потерь; кроме того, данный процесс может быть автоматизирован.

Как отмечено в [97], архитектурный прототип может быть отбрасываемым или эволюционирующим. Задача отбрасываемого прототипа состоит лишь в определении осуществимости предлагаемых проектных решений: соответственно, в ходе работы с ним можно использовать всевозможные сокращения, альтернативные технологии или имитации. В последствии прототип просто отбрасывается, а сохраняется только полученное в результате решение. Эволюционирующий же прототип реализуется в той же архитектуре, которая будет использована в конечной версии системы: иными словами, можно строить рабочую версию системы, развивая данный прототип.

Онтологическая модель позволяет работать именно с эволюционирующим типом прототипов, поскольку заложенные в такой модели сущности концептуального плана могут легко трансформироваться непосредственно в спецификации проектных решений, а также в сущности плана реализации (классы, объекты, таблицы, атрибуты, поля и т.д.).

Процесс онтологического специфицирования показан на рис. 2.4. Если проектировщик располагает пространством прототипов, релевантных задаче ***Z***, то ему необходимо проанализировать, может ли он использовать какие-либо из них для удовлетворения требований к проекту. Данный этап представляет собой так называемый обратный инжиниринг, или реверс-инжиниринг, в ходе которого обычно осуществляется исследование уже готового продукта (например, АС) и его документации, с целью понимания принципов его работы, что, в частности, позволяет воспроизвести продукт с аналогичными функциями, но без прямого копирования.

Если такие прототипы существуют, то необходимо подвергнуть их онтологическому специфицированию с ориентацией на структуру онтологии, описанную в параграфе 2.2. Специфицированию подвергаются исходные прототипы ***IPt***, например, такие сущности программной реализации

проектных решений, как классы, атрибуты, методы и объекты, а также таблицы и поля баз данных, которые содержит система. Кроме того, для формирования спецификаций может быть использована текстовая и графическая информация, содержащаяся в проектной документации к продукту.

В результате выполнения данного этапа набор прототипов ***IPt*** трансформируется в набор спецификаций двух типов:

$$Sp^O = (Sp^S, Sp^M) \quad (2.20)$$

где Sp^S – набор спецификаций по сохраняющейся части прототипов (формируются в том случае, когда прототип можно сохранить в неизменном виде);

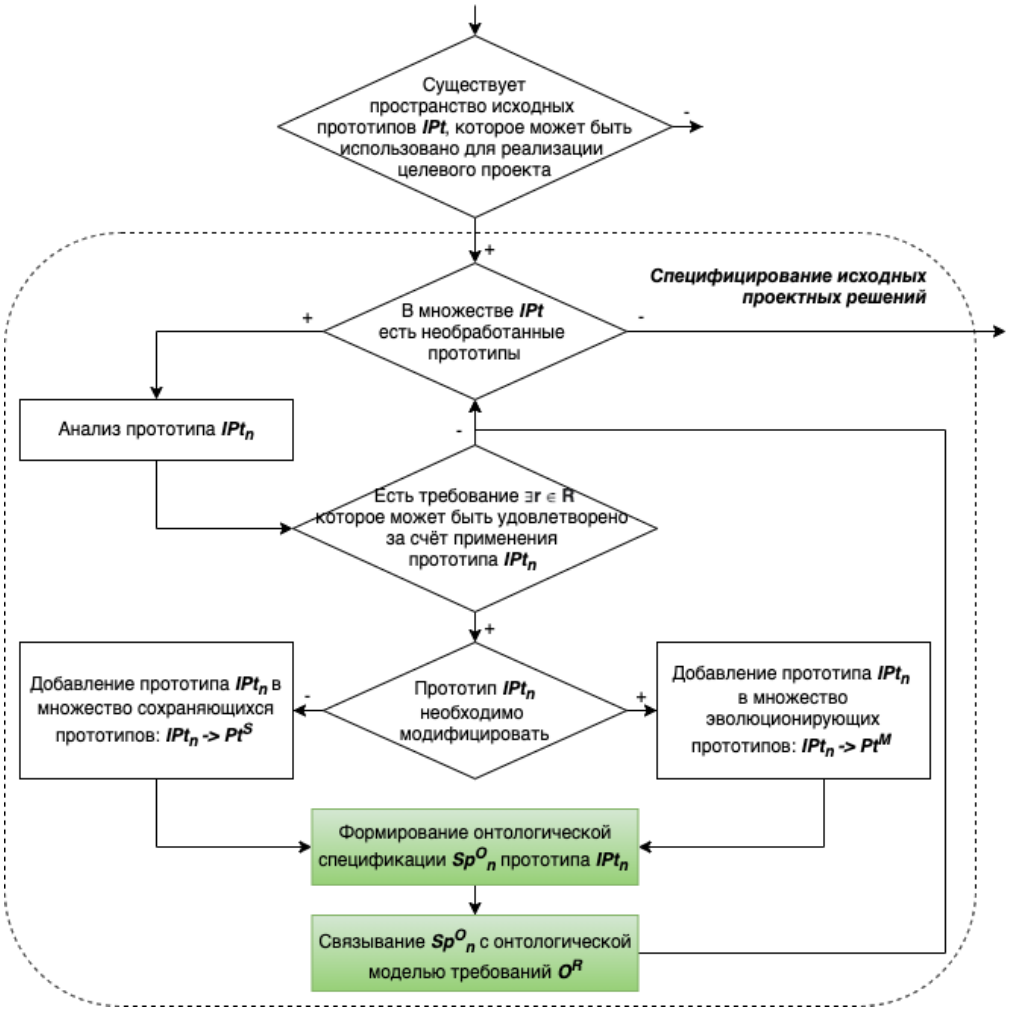


Рисунок 2.4 - Специфицирование исходных проектных решений

Sp^M – набор спецификаций по эволюционирующей части прототипов (формируются в том случае, когда прототип необходимо подвергнуть модификации).

Далее необходимо связать полученные спецификации с онтологической моделью требований. Связывание может быть осуществлено, как и в случае с материализациями требований, на уровне специальных агрегатов. В дальнейшем – при повторном использовании таких прототипов в рамках других проектов – онтологическое специфицирование требоваться уже не будет: необходимо будет обратиться к базе прецедентов и связать существующие онтологические спецификации нужных прототипов с ОМ требований конкретного проекта.

2.3.3 Формирование концепции проекта и предварительных проектных решений

Далее в соответствии с подходом проектировщик анализирует задачу и формулирует идеи для ее решения, чтобы в итоге сформировать концепцию проекта. На этом этапе также целесообразно применять технологию вопросно-ответного анализа, который позволяет осуществлять пошаговую детализацию задачи – вплоть до конкретных проектных решений. В ходе такого анализа формируется ***онтологическая модель проектных решений*** O^S, которая по своей сути является формой выражения концептуальной модели проекта.

Поскольку такая модель содержит в себе максимально полную информацию о проекте на данном этапе, она может быть использована для автоматического или автоматизированного генерирования UML-диаграмм, которые на сегодняшний день являются общепринятой и наиболее распространенной формой представления концептуальных моделей. Более подробно процесс формирования онтологической модели проектирования показан на рисунке 2.5.

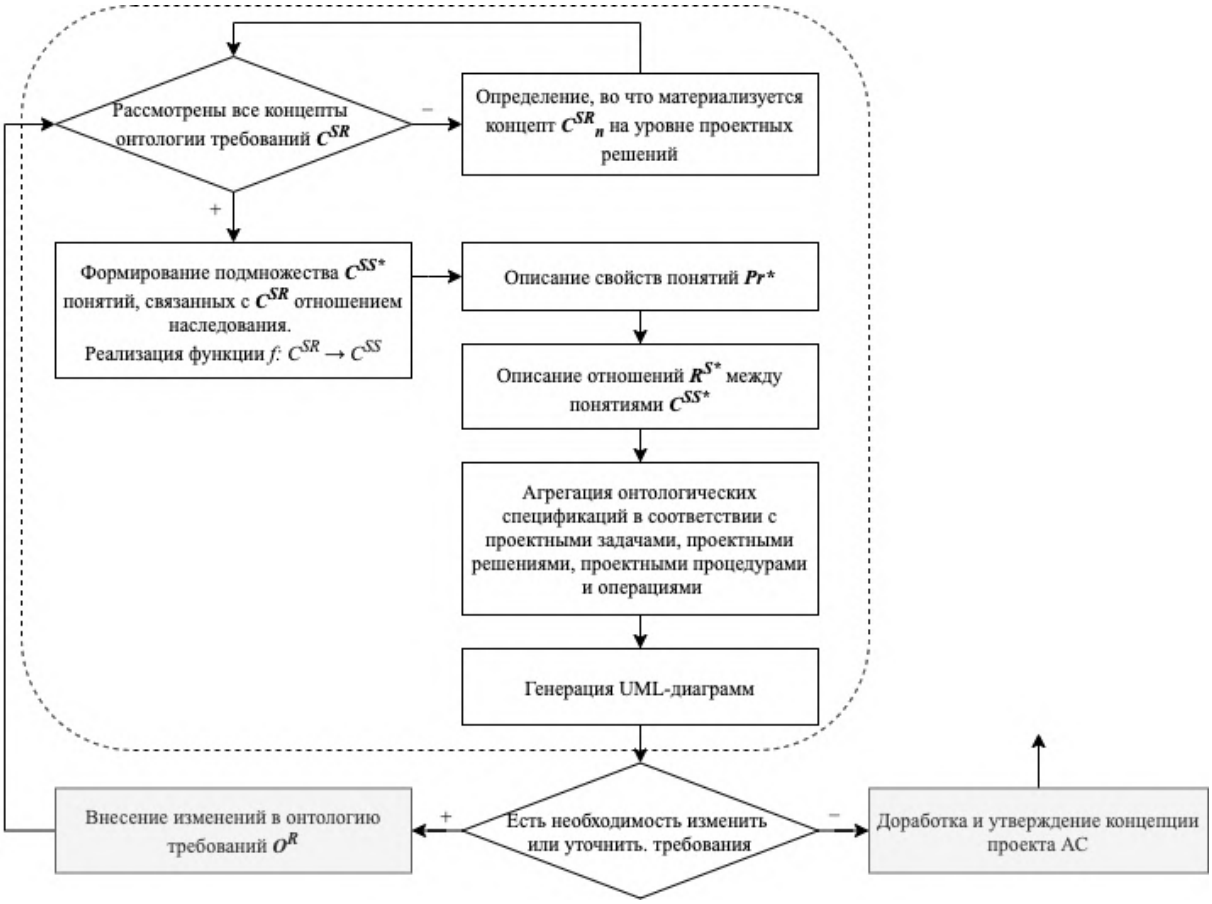


Рисунок 2.5 - Формирование онтологии проектных решений

2.3.4 Формирование онтологии реализации и генерация исходного кода

Дальнейшая ход проектирования АС в соответствии с подходом предполагает формирование ***онтологической модели реализации*** O^I проекта.

На этом этапе задается пространство сущностей, представленных в исходном коде программы; данным сущностям присваиваются соответствующие имена, а между ними и сущностями проектных решений устанавливаются отношения наследования. Такая конфигурация ОМ позволяет автоматически генерировать часть исходного кода программы автоматизации.

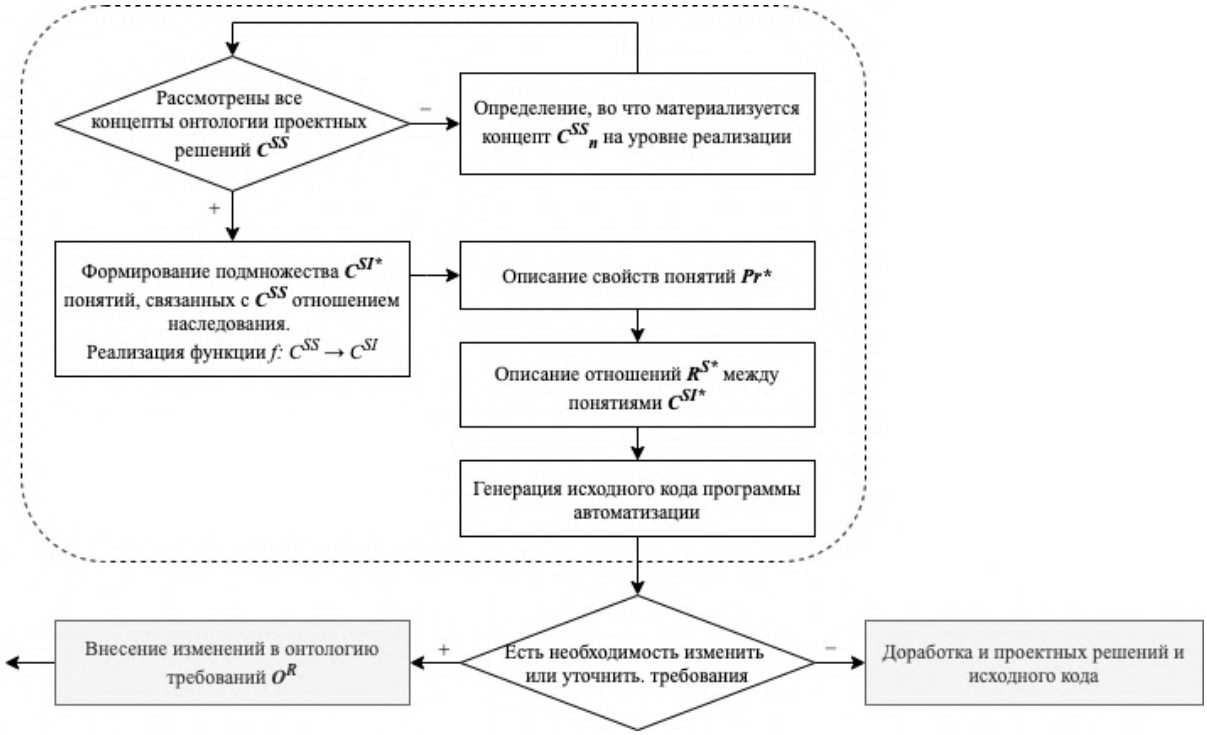


Рисунок 2.6 - Формирование онтологической модели реализации

Понятия онтологии требований сопоставляются с понятиями онтологии проектирования с помощью соответствующих функций интерпретации (как и понятия онтологии проектирования с понятиями онтологии реализации). Важным позитивным свойством такой конфигурации онтологии проекта является возможность ***автоматизированного реинжиниринга*** при возникновении необходимости внесения изменений или уточнений в набор требований: при внесении изменений в ОМ требований, соответствующие изменения также вносятся в ОМ проектирования и в ОМ реализации (вручную или автоматически – в зависимости от заданных функций интерпретации). Следовательно, данные изменения легко проникают через онтологическую модель непосредственно в артефакты проектирования (например, в UML-диаграммы и в исходный код программы), что способствует повышению степени автоматизации управления изменениями.

Дополнительным полезным эффектом является материализация многих пространств имен в онтологической модели проекта: в частности, пространства имен требований, исходных прототипов, проектных решений и

реализации. В рамках каждого из этих пространств одни и те же понятия могут быть сформулированы по-разному, но благодаря их связыванию в ОМ формируется собственный ***язык проекта***, за счет которого достигается его концептуальная целостность и значительно снижается уровень необходимых когнитивных усилий на понимание проектных задач и проектных ситуаций в рамках множества различных пространств имен.

2.4. Онтологическое моделирование в инфраструктуре удовлетворения потребительских запросов

Принимая во внимание **Заключение 1.1**, а также тот факт, что одна из целей развития современных САПР – повышение реактивности инфраструктуры удовлетворения потребительских запросов, предпримем попытку определить место средств онтологического моделирования в данной инфраструктуре в рамках производственного процесса, ориентированного на применение технологий CALS и IoT. На основе детализации процесса, проиллюстрированного на рис. 1.2, а также принимая во внимание структурно-функциональную организацию модуля по продажам и дистрибуции одной из лидирующих систем по планированию ресурсов предприятия SAP [98], построим диаграмму прецедентов (рис. 2.7).

На рис. 2.7 показано место технологии онтологического моделирования в инфраструктуре производственного процесса, обеспечивающего реактивность удовлетворения потребительских запросов. Потребитель контактирует с производителем на входе через индивидуальные запросы, отражающие эксплуатационные свойства изделий. Данные запросы подвергаются специфицированию, а впоследствии – обобщаются до спецификаций требованиям к производимым изделиям. Таким образом, потребитель становится *сопроектировщиком*.

Потребительские запросы могут быть не только индивидуальными, но и массовыми. Сбор и обработка таких запросов могут осуществляться,

например, с помощью специальных форм обратной связи и служить для оценивания уровня спроса на те или иные изделия, выявления трендов относительно их эксплуатационных свойств, а также, в свою очередь, формирования политики ценообразования. Высокий уровень спроса на определенные типы изделий приводят к формированию оптового потребительского запроса и необходимости создания оптовой партии изделий.

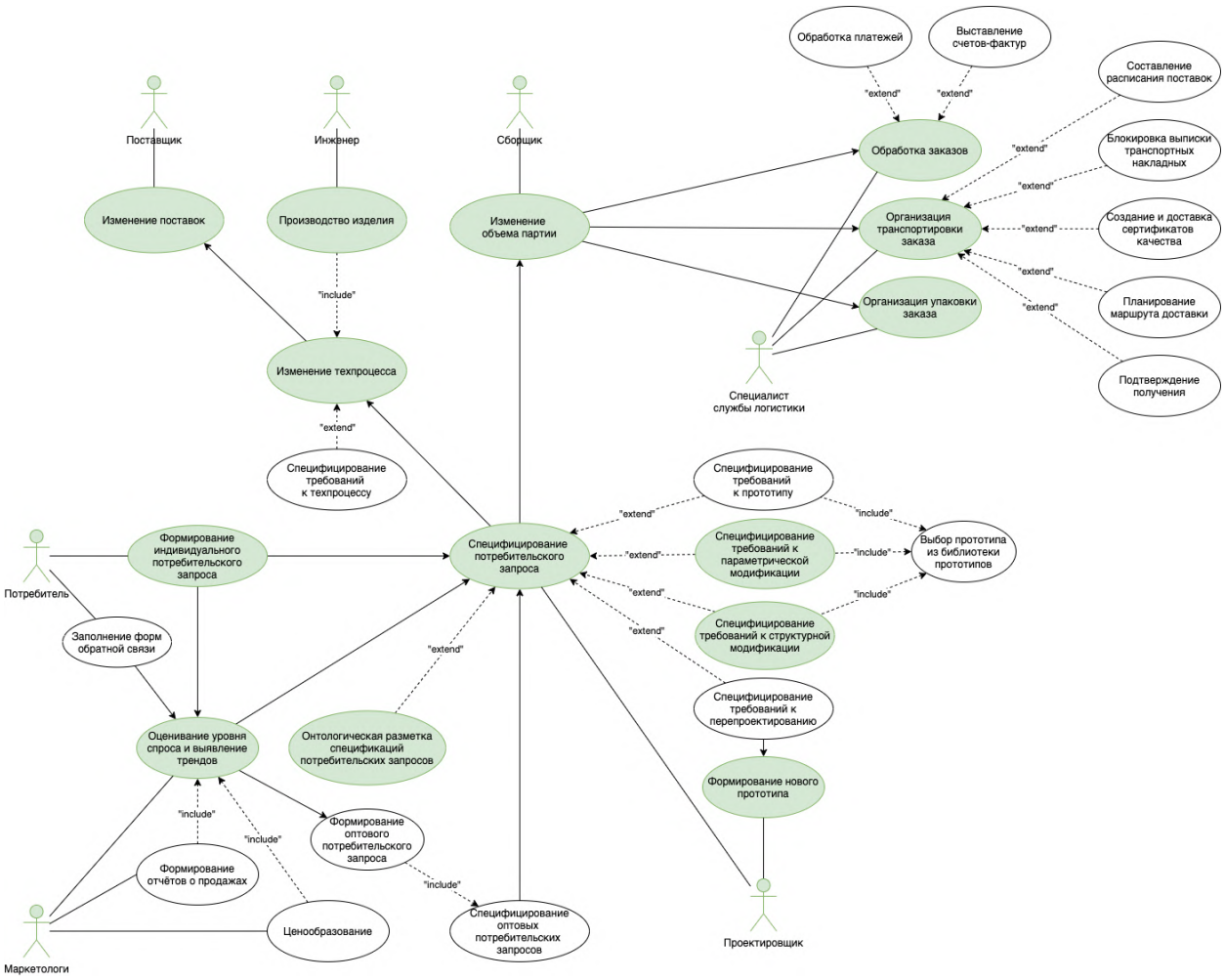


Рисунок 2.7 - Диаграмма прецедентов реактивной инфраструктуры удовлетворения потребительских запросов

Таким образом, САПР обеспечивает значительное сокращение сроков реализации потребностей покупателей, а применение технологий IoT в производственном процессе обеспечивает реактивность оборота

потребительских желаний, которые, в свою очередь, мотивируются благодаря данной реактивности.

Потребительские запросы обладают низкой степенью зрелости, однако реактивность инфраструктуры, в том числе благодаря применению технологий онтологического моделирования, позволяет увеличить ее. Спецификации требований производимых изделий могут быть условно разделены на 4 группы в зависимости от степени повторности использования прототипов при создании изделий, удовлетворяющих потребительские запросы:

- спецификации требований к прототипу (в случае если прототип можно использовать полностью в том виде, в котором он хранится в библиотеке прототипов);
- спецификации требований к параметрической модификации (когда необходимо модифицировать лишь некоторые параметры выбранных прототипов);
- спецификации требований к структурной модификации (когда необходимо модифицировать не только параметры, но и структуру выбранных прототипов);
- спецификации требований к перепроектированию (когда применение прототипа невозможно).

На рис 2.8. показано, каким образом спецификации потребительских запросов могут быть трансформированы в разного вида спецификации требований к производимому изделию. Стоит отметить, что данный процесс является циклическим, поскольку поток потребительских запросов необходимо постоянно обрабатывать и подвергать специфицированию в целях улучшения свойств производимого изделия и наиболее полного удовлетворения данных запросов. Таким образом обработка потребительских запросов обеспечивает не только реактивность, но и полноту специфицирования свойств изделий.

Все виды спецификаций, перечисленные выше, равно как и спецификации потребительских запросов могут быть подвергнуты онтологической разметке. Применение технологий онтологического моделирования в данном процессе обладает следующими положительными свойствами:

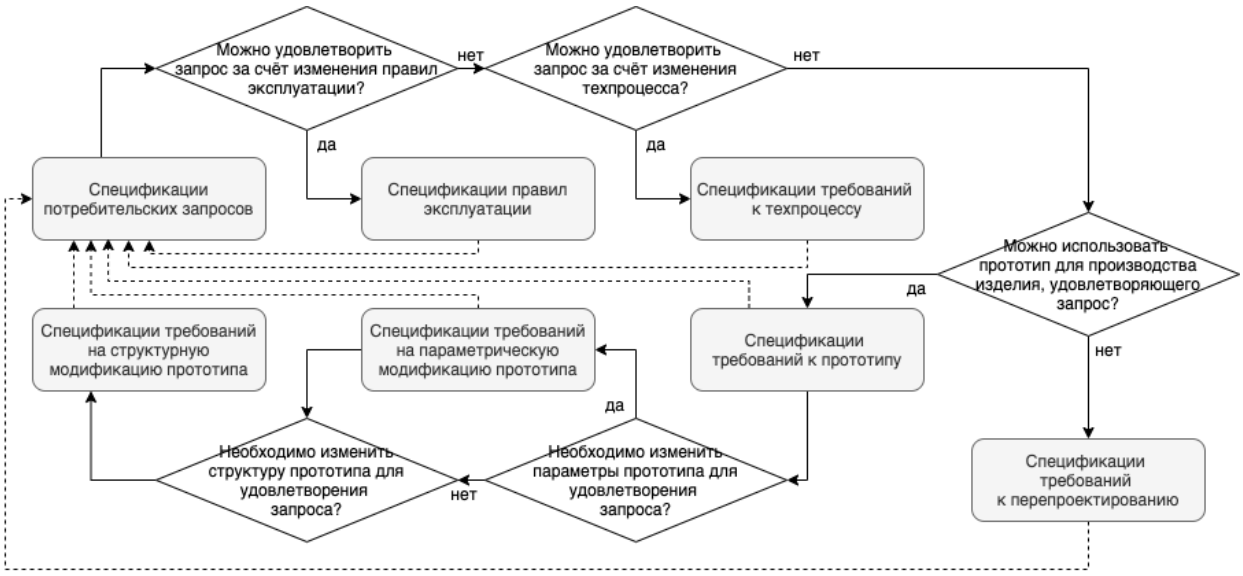


Рисунок 2.8 - Процесс порождения и трансформации спецификаций

- данные технологии позволяют существенно облегчить выбор прототипа, поскольку обеспечивает поиск сущностей, пересекающихся на уровне потребительского запроса и на уровне прототипа, хранящегося в библиотеке прототипов;
- поскольку изменение спецификаций потребительских запросов неизбежно заставляют производителя изменять спецификации свойств изделий, данный процесс облегчается благодаря онтологической модели, так как она позволяет определить, являются ли различные объекты проектирования материализациями одной и той же сущности.

Кроме того, очевидно, потребительские запросы отражают то, что является концептуально важным для производимого изделия: следовательно, онтологическая разметка спецификаций потребительских запросов позволяет

зафиксировать концептуально важные свойства производимых изделий, что впоследствии повышает эффективность производственного процесса в целом.

Выводы по главе 2

В основе подхода к онтологической поддержке проектирования лежат следующие положения:

1) процесс разработки АС базируется на механизмах проектного мышления и QA-моделирования, обслуживающие процесс моделирования рассуждений и активно вовлекающие в проектный процесс ОМ;

2) в целях сокращения семантических разрывов в проектном процессе, а также повышения степени автоматизации процесса онтологического моделирования ОМ проекта целесообразно представлять как многоуровневую систему связанных онтологий в согласии с традиционным пониманием разбиения проектного процесса разработки АС на этапы;

3) для построения такой онтологической модели проекта, которая способна управлять проектным процессом, носить динамический характер и оперативно отражать модификации разрабатываемых проектных решений и других артефактов проектирования, целесообразно организовать процесс онтологического моделирования как систему инкрементальных процессов, результатом каждого из которых является изменение состояния онтологической модели.

Разработана модель проектного процесса с применением механизмов онтологической поддержки, выстроенная с опорой на ГОСТ 34.601, а также парадигму дизайн-мышления. Процесс носит инкрементальный характер; на каждой стадии осуществляется взаимодействие с онтологией проекта, в ходе которого она модифицируется.

Определено место онтологического моделирования в инфраструктуре удовлетворения потребительских запросов, связанное с процессом порождения и трансформации спецификаций потребительских запросов.

Глава 3. Разработка технологии и инструментальных средств онтологической поддержки проектирования

Приступим к рассмотрению архитектуры инструментальных средств онтологической поддержки проектирования АС, опираясь на подход и модель проектного процесса, описанные в предыдущей главе.

Схематично архитектура представлена на рис. 3.1 в виде диаграммы компонентов. Как видно на диаграмме, инструментальные средства состоят из следующих основных блоков:

- модули обработки исходных данных, которые используются для формирования и/или модификации онтологической модели проекта;
- модули, которые используют данные из ОМ для генерирования проектных решений;
- вспомогательные скрипты;
- непосредственно онтология проекта.

Модули обработки исходных данных разграничены в зависимости от типа анализируемых данных – текст, фрагменты баз данных и исходный код программ (модули анализа баз данных и исходного кода отмечены пунктиром, поскольку спроектированы, но не реализованы в рамках настоящего исследования). В качестве исходных данных предлагается рассмотреть следующее:

- данные, относящиеся к ***требованиям к проекту*** (неформальные описаний системы на естественном языке; тексты рассуждений, организованные в формате вопросно-ответных протоколов; проектная документация);
- данные, относящиеся к ***исходным прототипам*** (документация, фрагменты баз данных и исходного кода).

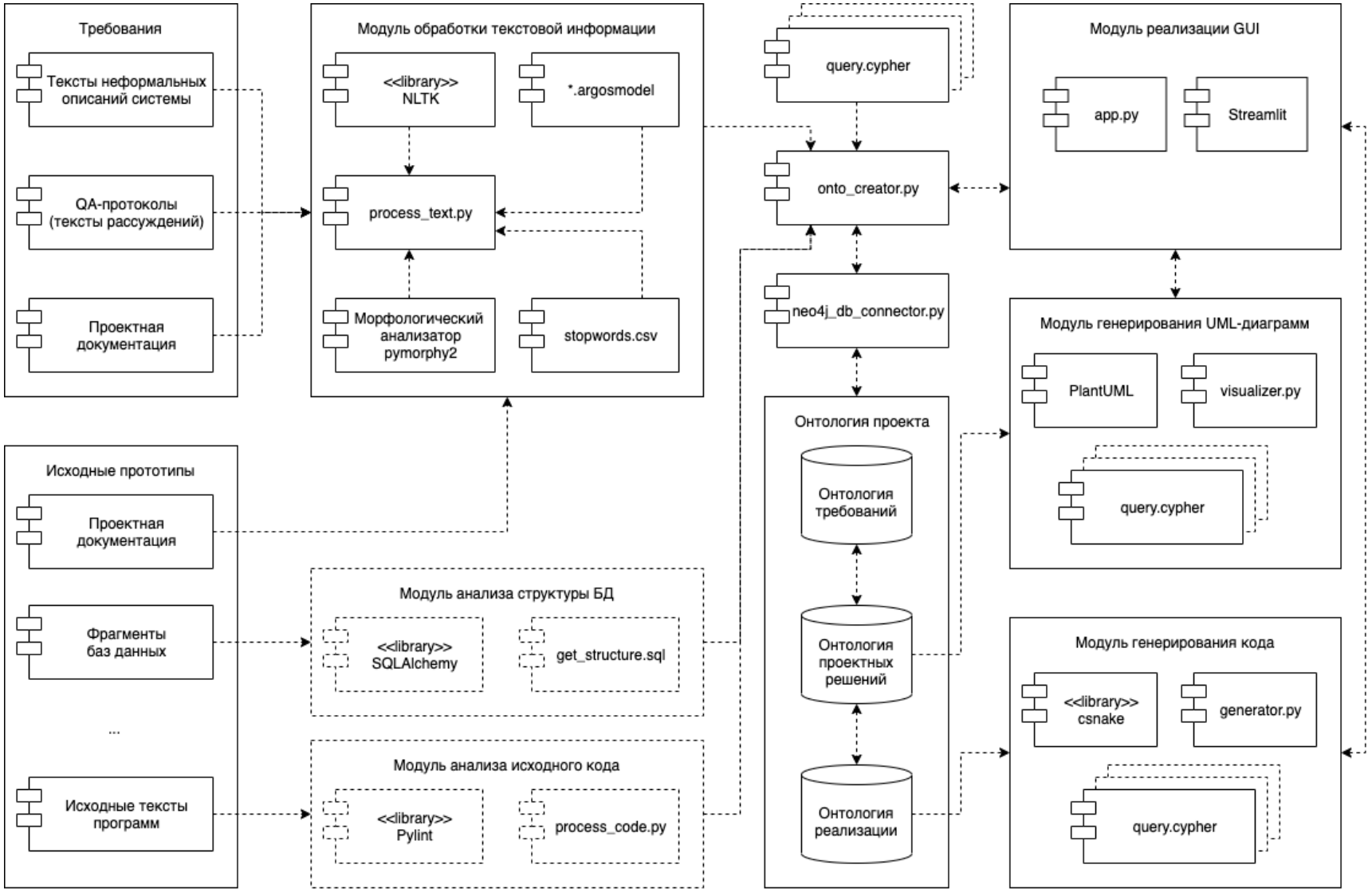


Рисунок 3.1 - Архитектура инструментальных средств

Отметим, что набор таких модулей может быть расширен или модифицирован в зависимости от исходных данных, которые есть в наличии и используются для формирования онтологии проекта.

Также предусмотрено два ***модуля генерирования проектных решений*** на основе данных из онтологии – модуль генерирования UML-диаграмм и модуль генерирования кода. ***Вспомогательные скрипты*** используются для осуществления соединения с базой данных онтологии проекта, выполнения преобразований исходных данных (при необходимости) и осуществления запросов на запись в онтологию.

В качестве основного языка программирования инструментальных средств выбран ***Python***, т.к. он отличается высокой степенью гибкости и динамичности, в том числе за счет возможности подключать внешние пакеты модулей, а также наличием большого количества готовых модулей, позволяющих эффективно обрабатывать информацию в актуальных для данного исследования представлениях. Графический интерфейс пользователя реализован на фреймворке Streamlit.

3.1 Организация онтологической модели проекта

В соответствии с Заключением 1.4, для организации хранения данных в онтологии проекта была выбрана схема LPG (Labeled Property Graph) – в частности графовая система управления базами данных Neo4j, которая является одним из наиболее известных инструментов, базирующихся на схеме LPG. Рассмотрим подробнее структуру данных, которой оперирует данная система.

Вершины графа в терминологии Neo4j называются ***узлами*** (nodes) и представляют собой сущности. С помощью узлов будем задавать ***понятия*** онтологии. Любые узлы в графе могут иметь ***метки*** (labels). Метки используются, чтобы агрегировать узлы в подмножества. При этом любой узел может иметь неограниченное количество таких меток (или не иметь никаких

меток вовсе). В рамках наших инструментальных средств метки используются для двух целей:

- разграничение онтологий каждого из этапов проектного процесса (так, понятия, относящиеся к онтологии требований, будут иметь метку *Requirements*; понятия, относящиеся к онтологии проектных решений, – метку *Design*; понятия, относящиеся к онтологии реализации, – метку *Implementation*);
- разграничение смысловых агрегатов в рамках онтологий (структура агрегатов представлена далее).

Два узла связываются ***отношением*** (relation). Любое отношение обязательно имеет определенное направление, а также единственный тип (задается с помощью строки). В рамках наших инструментальных средств данная структура используется для связывания понятий семантическими отношениями.

Как узлы, так и отношения могут иметь любое количество ***свойств*** (properties). Свойства представляют собой пары вида «ключ – значение» и используются для описания характеристик сущностей. Так, в наших средствах свойства используются для связывания нескольких пространств имен: в качестве ключа задается условное названия пространства (например, *dev_name*, что означает, что имя относится к пространству исходного кода), а в качестве значения – само имя. Таким образом, свойства служат для связывания онтологии с материализациями ее сущностей в проекте.

Аксиоматику онтологических моделей можно формулировать на уровне Cypher-запросов к базе данных. Cypher – это декларативный язык запросов к графовой СУБД Neo4j.

Формальные спецификации композиции онтологий, ориентированных на автоматическую генерацию программ логического управления, целесообразно базировать на логико-алгебраических моделях, позволяющих представлять сущности и отношения всех этапов проектирования таким

образом, чтобы модели обслуживали не только спецификацию проектных решений, но и процесс автоматизации проектирования. Поскольку любой проектный процесс является процессом трансформации одних проектных спецификаций в другие, важное значение для формальных спецификаций имеют функциональные отношения, определяющие отображение исходных спецификаций в результирующие для каждой проектной операции.

3.1.1 Формирование онтологии требований

Обобщенная структура онтологии требований в случае проектирования АС представлена множеством классов концептов CC^{SR}, а также множеством функций интерпретации уровня требований.

$$O^R = (Source, Inputs, Outputs, Actions, Events, Messages, Processes, Protocols, Conditions \mid RInputs, ROutputs, RActions, REvents, RMessages, RProcesses, RProtocols, RConditions) \quad (3.1)$$

где *Source* – множество исходных данных, к которым относятся тексты неформального описания объекта проектирования, протекающих в нем процессов и технических условий, определяющих требования; проектная документация; QA-протоколы, а также исходные прототипы;

- *Inputs* – множество спецификаций входов АС;
- *Outputs* – множество спецификаций выходов АС;
- *Actions* – множество спецификаций операций и действий АС;
- *Events* – множество спецификаций событий, возникающих в АС и объекте автоматизации;
- *Messages* – множество спецификаций сообщений о событиях, действиях, значениях входов и выходов;
- *Conditions* – множество спецификаций элементов технических условий, включающее в себя функциональные, параметрические и иные требования;

- *Processes* – множество спецификаций процессов, осуществляемых в ходе функционирования системы и объекта автоматизации;
- *Protocols* – множество спецификаций протоколов о ходе функционирования системы;
- *RInputs: Source** → *Inputs** – отношение «быть входом», обеспечивающее связывание подмножества предложений неформального описания с подмножеством входов АС; здесь и далее конструкция *S** означает множество всех подмножеств *S*;
- *ROutputs: Source** → *Outputs** – отношение «быть выходом», обеспечивающее связывание подмножества предложений неформального описания с подмножеством выходов;
- *RActions: Source** × *Inputs** × *Outputs** → *Actions** – отношение «быть операцией», обеспечивающее связывание подмножества предложений неформального описания с подмножеством операций и действий, а также определяющее зависимости операций от входов и выходов;
- *REvents: Source** × *Inputs** × *Outputs** × *Actions** → *Events** – отношение «быть событием», обеспечивающее связывание подмножества предложений неформального описания с подмножеством событий в ходе функционирования АС, а также определяющее зависимости событий от входов, выходов и операций;
- *RMessages: Source** × *Events** → *Messages** – отношение «быть сообщением», обеспечивающее связывание подмножества предложений неформального описания с подмножеством сообщений, возникающих в ходе функционирования АС в контексте конкретных событий или их групп;
- *RProcesses: Source** × *Inputs** × *Outputs** × *Actions** × *Events** → *Processes** – отношение «быть процессом», обеспечивающее связывание подмножества предложений неформального описания с подмножеством процессов, протекающих в АС и охватывающих изменения входов и выходов, выполнение операций, возникновение событий;

- *RProtocols: Source* × Inputs* × Outputs* × Actions* × Events* × Messages* × Processes* → Protocols* – отношение «быть протокольной записью», обеспечивающее связывание подмножества предложений неформального описания с подмножеством процессов, протекающих в АС и охватывающих последовательности изменения входов и выходов, выполнения операций, возникновение событий и сообщений;
- *RConditions: Source* × Inputs* × Outputs* × Actions* × Events* × Protocols* × Messages* → Conditions** – отношение «быть условием», обеспечивающее связывание подмножества предложений неформального описания с подмножеством условий, задающих требования к АС в контексте связей этих требований со входами, выходами, событиями, операциями и сообщениями.

Свойства концептов Pr^R определяют имена сущностей (*name*), а также тексты неформального описания объекта проектирования (*description*).

Разработанные средства автоматизации позволяют обработать текст на естественном языке из множества *Source* и получить на выходе список претендентов на включение в онтологию требований в качестве концептов, в том числе на основе частоты вхождения тех или иных слов или словосочетаний в текст (минимальную частоту можно регулировать при необходимости). На рис 3.3 представлен пример интерфейса программы

Добавленные в онтологию концепты из списка претендентов автоматически связываются с элементами множества *Source* отношением «быть источником». Дополнительно между концептами можно установить семантические связи.

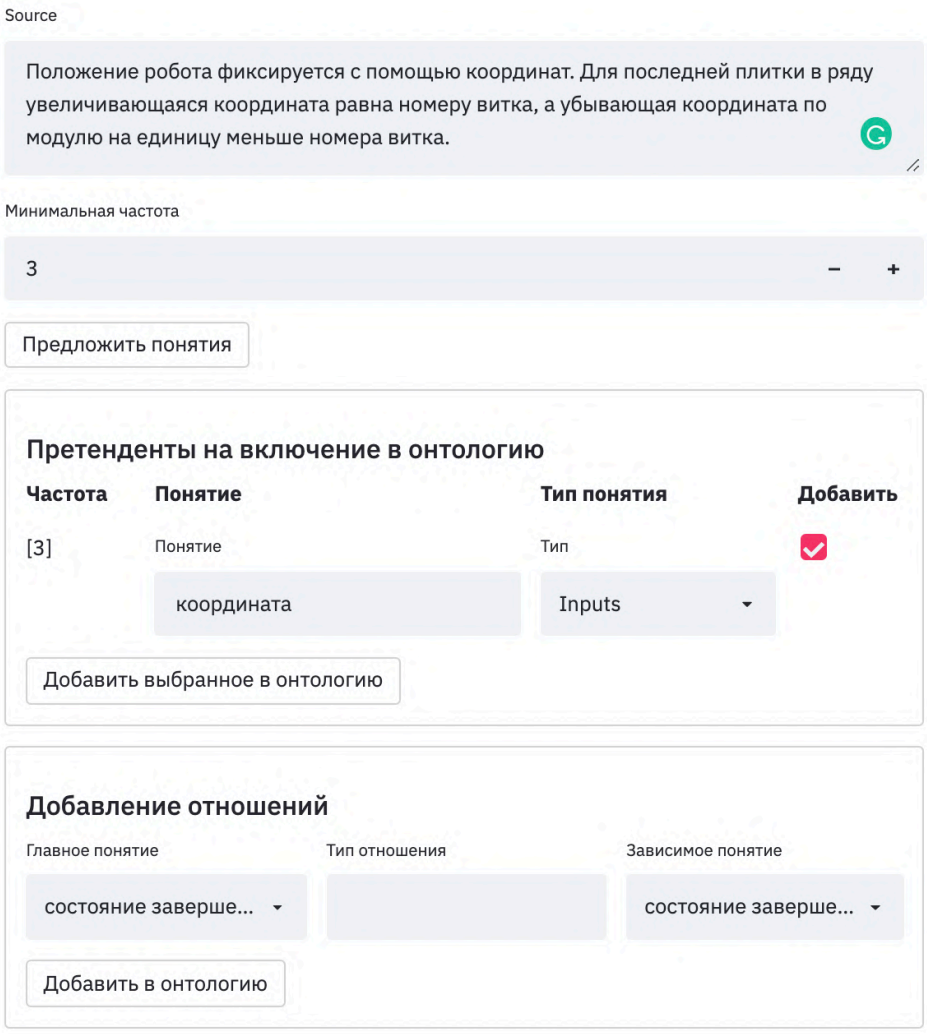


Рисунок 3.2 – Интерфейс программы по пополнению онтологии требований

3.1.2 Формирование онтологии проектирования

Процедурная часть онтологии проектирования определяет проектные операции (*DActions*), проектные задачи (*DTasks*), проектные решения (*DSolutions*) и проектные процессы (*DProcesses*), а также правила их порождения или модификации.

$$O^S = (DSolutions, DActions, DTasks, DProcesses \mid RDStates, RDTransitions, RDPredicates, RDActions, RDTasks, RDSolutions, RDProcesses) \quad (3.2)$$

Элементы множества *DSolutions* представляют собой спецификации методов решения задач автоматизации и соответствующих алгоритмов,

способов представления функциональных зависимостей и данных, в том числе схемы баз данных, спецификации интерфейсов между объектом управления и человеком-оператором и др. Для построения конкретного проектного решения из множества *DSolutions* используется последовательность проектных операций из множества *DActions* в соответствии с проектным процессом из множества *DProcesses* в рамках задачи из множества *DTasks*. Представленная ниже система отношений обслуживает данный процесс:

- *RDActions: DStates* × Inputs* × Outputs* × Actions* × Events* × Messages* × Conditions* × Protocols* → DActions** – отношение «быть проектной операцией», обеспечивающее порождение или модификацию подмножества проектных операций применительно к одному или нескольким состояниям процесса управления и вовлечение в операцию спецификаций из онтологии требований;
- *RDTasks: DActions* × Inputs* × Outputs* × Actions* × Events* × Messages × Conditions* × Protocols* → DSolutions** – отношение «быть проектной задачей», обеспечивающее агрегацию проектных операций и порождение или модификацию подмножества проектных решений из множества *DSolutions* в контексте вовлечения в агрегат операций спецификаций из онтологии требований;
- *RDSolutions: DSolutions* × DTasks* × Inputs* × Outputs* × Actions* × Events* × Messages* × Conditions* × Protocols* → DSolutions** – отношение «быть проектным решением», обеспечивающее порождение или модификацию подмножества проектных решений из множества *DSolutions* в контексте использования наборов проектных операций и спецификаций из онтологии требований;
- *RDProcesses: DTasks* × DSolutions* → DTasks** – отношение «быть проектным процессом», обеспечивающим связывание подмножества проектных решений и проектных задач.

Аналогичным онтологии требований образом может быть построена метамодель, которая на данном этапе служит не только инструкцией для проектировщиков по формированию онтологии проектирования, но и базой для частичной автоматизации трансформации онтологии требований в онтологию проектирования, поскольку содержит в себе правила порождения новых сущностей.

Пример 1. Проектное решение «Автоматная модель» (см. рис 3.4) содержит в себе группы связанных между собой понятий, характеризующих состояния (*State*), переходы между состояниями (*Transition*), условия перехода (*Predicate*), действия (*Action*). Тогда проектная задача по специфицированию некоего процесса управления в соответствии с автоматной моделью будет включать в себя проектные операции по формированию вышеупомянутых сущностей на основе онтологии требований, часть из которых может быть автоматизирована с помощью заданных аксиом.

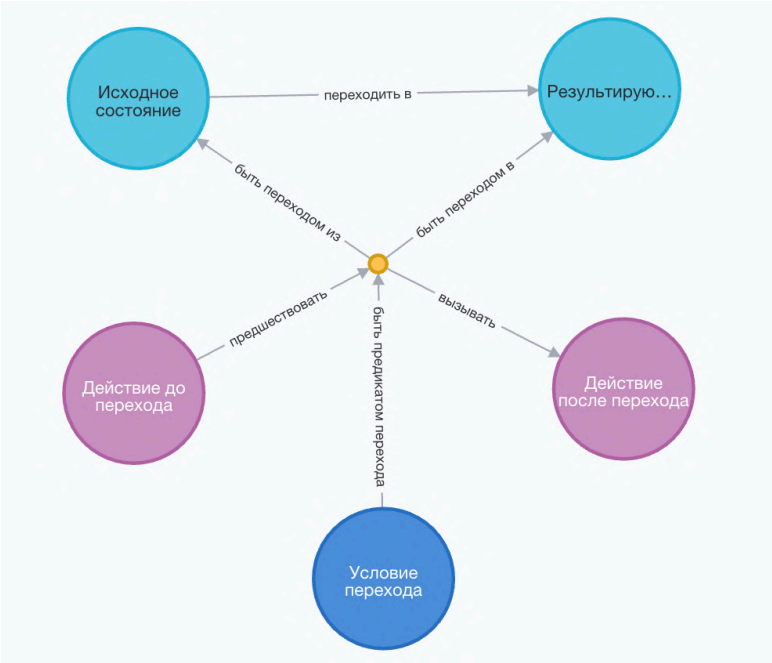


Рисунок 3.3 – Онтологическое представление проектного решения «Автоматная модель»

Ограничения на структуру онтологической модели можно сформулировать на естественном языке следующим образом:

- обязательными элементами онтологической модели в соответствии с проектным решением «Автоматная модель» являются два концепта типа State (состояние), связанные отношением «переходить в», а также концепт типа Predicate (условие перехода);
- опциональными элементами онтологической модели в соответствии с проектным решением «Автоматная модель» являются концепты типа Action (действие), соответствующие действиям, которые необходимо выполнить до либо после перехода; при этом таких действий может быть неограниченное количество.

Правила логического вывода данной онтологической модели представлены в ниже с использованием аппарата логики предикатов:

- ЕСЛИ существуют два концепта типа State, связанные отношением «переходить в», а также связанный с ними концепт типа Predicate (условие перехода), ТО существует концепт типа Transition, соответствующий переходу из одного состояние в другое и связанный с исходным состоянием отношением типа «быть переходом из», с результирующим состоянием – отношением типа «быть переходом в» и с условием перехода – отношением типа «быть условием перехода»:

  $\exists State(s_1) \wedge \exists State(s_2) \wedge \text{переходитьВ}(s_1, s_2) \wedge$
  $\exists Predicate(p) \rightarrow \exists Transition(t) \wedge \text{бытьПереходомИз}(t, s_1) \wedge$
  $\text{бытьПереходомВ}(t, s_2) \wedge \text{бытьУсловиемПерехода}(p, t)$

- ЕСЛИ существуют концепты типа Action, связанные с концептами типа State отношением предшествования, ТО между данными концептами и концептом Transition, соответствующий переходу из указанного состояния, существует связь типа «предшествовать»:

$\exists Action(a) \wedge \exists State(s) \wedge \text{выполняться В Сотоянии}(a, s) \wedge \exists Transition(t) \wedge \text{быть ПереходомИз}(t, s) \rightarrow \text{предшествовать}(a, t)$

- ЕСЛИ существуют концепты типа Action, связанные с концептами типа State причинно-следственным отношением, ТО между данными концептами и концептом Transition, соответствующий переходу в указанное состояние, существует связь типа «вызывать»:

  $\exists Action(a) \wedge \exists State(s) \wedge cause(s, a) \wedge \exists Transition(t) \wedge \text{быть ПереходомВ}(t, s) \rightarrow \text{вызывать}(t, a)$

В Neo4j такие правила реализуются на уровне запросов к БД (левая часть правила задается клаузулой MATCH, правая – клаузулой CREATE с использованием выбранных в правой части сущностей). В качестве примера приведем реализацию последнего правила (см. листинг 3.1).

Листинг 3.1 - Пример реализации правила логического вывода с помощью Cypher-запроса

```
MATCH (s:State), (a:Action), (t:Transition),
(s)-[:Solutions {name: 'cause'}]->(a)
CREATE (t)-[:Solutions {name: 'вызывать'}]->(a)
```

Онтологическая модель проектного решения позволяет управлять интерфейсом наполнения онтологии проектных решений: необходимые поля, а также обязательность их заполнения определяется структурой ОМ проектного решения (см. рис 3.5). В качестве имен состояний, условий перехода и действий можно выбрать концепты, уже занесенные в онтологию требований. При этом их имя в онтологии проектировании может отличаться (если поле оставить пустым, то имя будет наследовано из онтологии требований).

При нажатии на кнопку «Добавить в онтологию» в онтологии проектных решений появится совокупность связанных концептов, каждый из которых

будет связан, в свою очередь, с концептами онтологии требований отношением «наследоваться от».

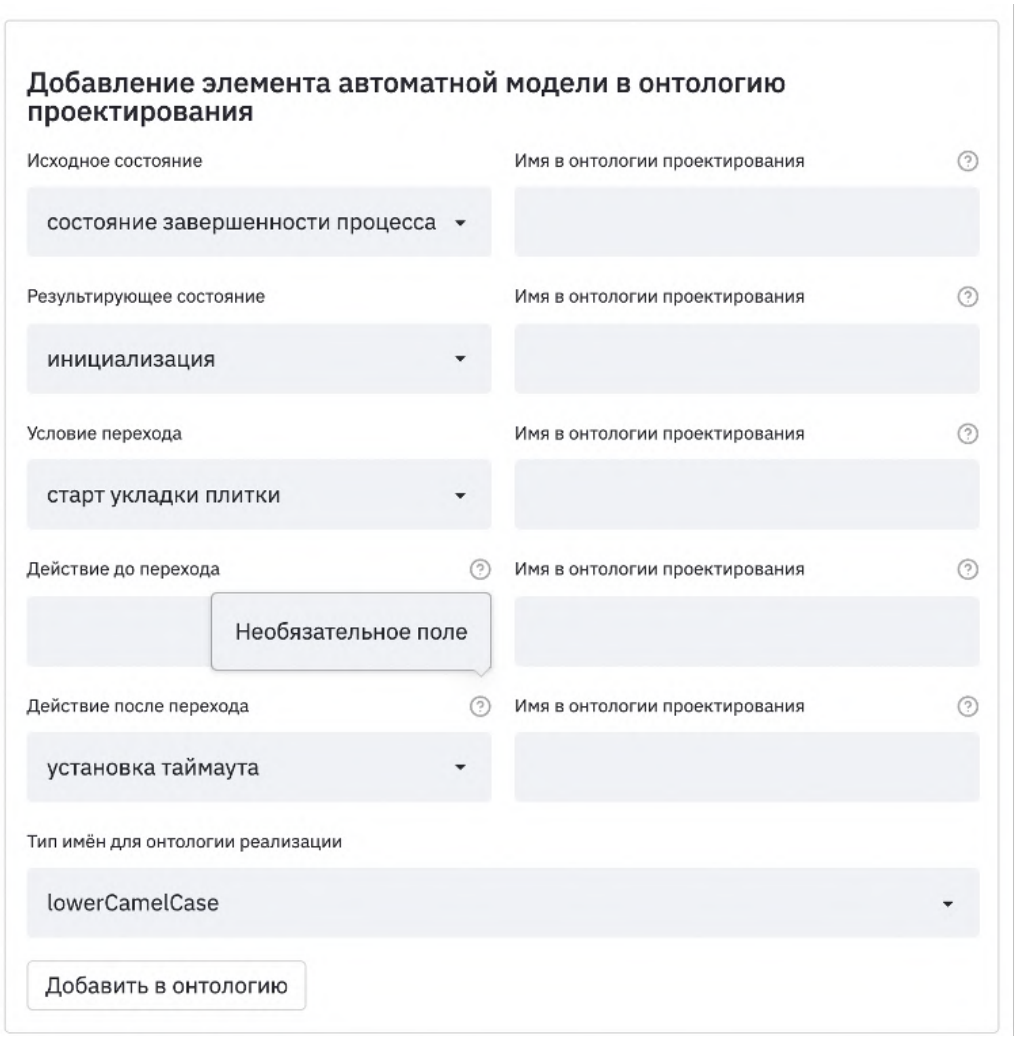


Рисунок 3.4 – Интерфейс программы по пополнению онтологии проектирования

3.1.3 Формирование онтологии реализации

Процедурная часть онтологии реализации, во-первых, обеспечивает порождение новых сущностей, которые связаны с сущностями, заданными на уровне проектных решений, отношением «быть реализованным»; во-вторых, задает новые множества связанных между собой понятий:

$$O^I = (IFunctions, IOperations, IParameters) \quad (3.3)$$

где *IFunctions* – множество функций исходного кода программы; *IOperations* – множество операций исходного кода программы; *IParameter* –

множество параметров функций исходного кода программы автоматизации. Другие множества понятий (например, состояния или классы) наследуются из онтологии проектных решений.

Интерфейс формирования онтологии реализации также управляется онтологической моделью (см. рис. 3.6). Для каждого понятия онтологии проектирования генерируются имена в соответствии с выбранным типом (CamelCase, lowerCamelCase, snake_case), которые могут быть изменены вручную при необходимости.

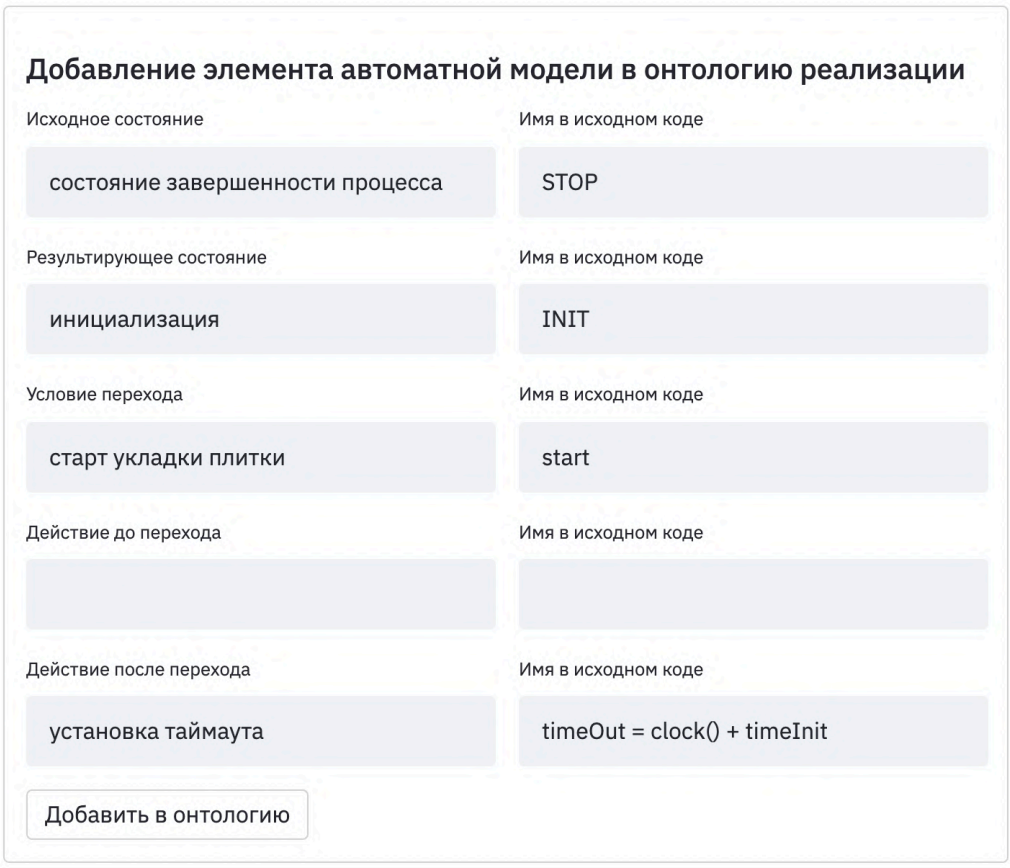


Рисунок 3.5 – Интерфейс программы по пополнению онтологии реализации

При нажатии на кнопку «Добавить в онтологию» в онтологии реализации появится совокупность связанных концептов, каждый из которых будет связан, в свою очередь, с концептами онтологии проектирования отношением «наследоваться от».

3.2 Модуль обработки текстовой информации

Высокий уровень присутствия сущностей объектов и процессов автоматизации в проектных спецификациях различных этапов проектирования обеспечивается прежде всего за счет применения специальных механизмов текстовой обработки данных, инструментально-технологические аспекты которых рассматриваются ниже.

Реализован следующий алгоритм формирования спецификаций, ориентированных на онтологическое моделирование, на основе текстовой информации на естественном языке:

1) токенизация текста (разбиение текста на токены ***{T}*** – словоформы) с использованием Natural Language Toolkit (NLTK) для Python [100];

2) лемматизация токенов (приведение словоформ к исходной –форме) с использованием pymorhy для Python [101];

3) фильтрация токенов для выявления единиц, претендующих на включение в онтологию в виде концептов C^{W*} на основе словаря стоп-слов [102], а также морфологических характеристик;

4) извлечение цепочек токенов длиной 2 или 3 единицы ***{Col}*** из текста;

5) фильтрация словосочетаний для выявления единиц, претендующих на включение в онтологию в виде концептов C^{Col*};

6) автоматическое порождение имен, ориентированных на программную реализацию, с использованием библиотеки автоматического перевода с открытым исходным кодом Argos Translate (инструментальные средства позволяют сгененировать имя класса, функции, параметра и т.д. с учетом выбранного стиля: CamelCase, lowerCamelCase или snake_case);

7) построение гипотез относительно наличия семантических связей R^* того или иного типа между претендентами на включение в онтологию (поиск следов этих связей в тексте).

Рассмотрим подробнее этапы фильтрации токенов, формирования словосочетаний и построения гипотез относительно семантических связей.

3.2.1 Фильтрация токенов

Одной из проблем, стоящих перед проектировщиком во время формирования проектной онтологии, состоит в том, что для создания словаря приходится концептов обработать большой объем текстовой информации, что занимает значительное количество времени, если совершать эту работу вручную. Для ее облегчения применяются различные механизмы фильтрации, которые позволяют сократить объем лексики, претендующей на включение в словарь, а также повысить релевантность данной выборки.

Для целей нашего исследования будем использовать фильтрацию на основе словаря стоп-слов. Данный механизм позволяет фильтровать список слов, уже приведенных к своей в исходной (начальной) форме, на основе открытого словаря стоп-слов. Стоп-слова, с точки зрения лингвистики, – это такие слова, которые не несут существенной семантической нагрузки на текст, чаще всего выполняют служебную роль, однако необходимы для целостного и правильного восприятия текста человеком. К этим словам чаще всего относят предлоги, союзы, частицы, местоимения, вводные слова, междометия и т.д. Соответственно, мы можем игнорировать такие слова при поиске концептов.

Набор стоп-слов для русского языка, который было решено использовать в рамках нашего исследования, представлен в [102]. Данный набор был создан для улучшения работы поисковых машин, что делает его полезным также и для нашей задачи.

Кроме того, полезной может быть фильтрация на основе частеречной принадлежности. Для более детального понимания данного механизма обратимся к классификациям частей речи. Стоит отметить, что таких классификаций существует несколько, однако в нашем случае будем

использовать наиболее общую – классификацию по способу номинации, описанную Н.С. Шарафутдиновой [103].

На ее основании мы можем сделать вывод о том, что при выделении однословных концептов нас интересуют только самостоятельные части речи, которые могут в полной мере нести на себе смысловую нагрузку. Более того, поскольку, согласно [104], «существительное в семантическом аспекте по своей категориальной характеристике как часть речи, обозначающая субстанцию, прототипически денотативно находится в прямой связи с соответствующим объектом (предметом, лицом, вещественной субстанцией) реальной действительности», можно пренебречь другими частями речи при фильтрации и на первом этапе рассматривать в качестве претендентов на включение в онтологию только имена существительные.

3.2.2 Выявление многословных концептов (словосочетаний)

Поскольку концепты могут представлять собой не только слова, но и словосочетания, нам необходимо извлекать их из текста. Для получения многословных лексических единиц, которые могут входить в состав онтологической модели, из списка токенов извлекаются пары и тройки последовательно стоящих токенов, не разделенных знаками препинания – они являются претендентами на словосочетания. Отметим, что не всякие группы лексем можно назвать словосочетаниями с точки зрения естественного языка, поэтому был разработан механизм фильтрации групп лексем.

Предлагается находить словосочетания в тексте, ориентируясь на их лексико-грамматические типы, т.е. модели на основе частеречной принадлежности. В качестве модели для русского языка будем использовать классификацию словосочетаний Н.С. Валгиной [105], т.к. в ходе исследования и сравнения с другими классификациями выяснилось, что именно она является наиболее полной. В таблице 3.1 описаны модели словосочетаний для русского языка.

Таблица 3.1 - Модели словосочетаний в русском языке

Модель словосочетания	**Пример**
Глагол + Существительное / Существительное + глагол	*включить прибор, совершить остановку*
Глагол + Местоимение / Местоимение + глагол	*сделать это*
Глагол + Предлог + Существительное / Предлог + Существительное + Глагол	*перейти в состояние*
Глагол + Предлог + Местоимение / Предлог + Местоимение + Глагол	*двигаться к нему*
Глагол + Деепричастие / Деепричастие + Глагол	*сидеть задумавшись*
Глагол + Инфинитив / Инфинитив + Глагол	*начать двигаться*
Существительное + Существительное	*звено цепи, состояние автомата*
Существительное + Предлог + Существительное / Предлог + Существительное + Существительное	*рукоятка от выключателя*
Существительное + Прилагательное / Прилагательное + Существительное	*контактный рычаг, аварийный режим*
Местоименное прилагательное + Существительное / Существительное + Местоименное прилагательное	*его усилие, их алгоритм*
Существительное + Порядковое числительное / Порядковое числительное + Существительное	*второй шаг, шестая попытка*
Причастие + Существительное / Существительное + Причастие	*выключенный прибор*

Существительное + Наречие / Наречие + Существительное	*удар наотмашь, прогулка пешком*
Существительное + Инфинитив / Инфинитив + Существительное	*намерение возвратиться, умение рассказывать*
Прилагательное + Предлог + Существительное / Предлог + Существительное + Прилагательное	*черный от загара, смелый от рождения*
Прилагательное + Местоимение / Местоимение + Прилагательное	*нужный нам, приятный вам*
Прилагательное + Предлог + Местоимение / Предлог + Местоимение + Прилагательное	*близкий для себя, трудный для нас*
Прилагательное + Наречие / Наречие + Прилагательное	*по-летнему зеленый, дружески заботливый*
Прилагательное + Инфинитив / Инфинитив + Прилагательное	*готовый бороться, способный любить*
Числительное + Прилагательное / Прилагательное + Числительное / Порядковое числительное + Прилагательное / Прилагательное + Порядковое числительное	*третий от конца, первый из трех, пятый из пассажиров, две книги, трое в шинелях*
Наречие + Наречие	*очень ловко, весьма искусно, совершенно одинаково*
Существительное + Предлог + Наречие / Предлог + Наречие + Существительное	*смешно до слез, далеко от друзей*

Процесс выявления многословных концептов разделен на три этапа:

1) выделение групп лексем;
2) фильтрация групп лексем на основе их морфолого-синтаксических моделей;
3) формирование групп многословных концептов.

3.3.3 Поиск следов семантических связей в тексте

Автоматическое извлечение семантических связей из текстов, написанных на естественном языке, является сложнейшей задачей в силу языковой многозначности, многообразия лексических средств и вариативности.

Нами была предпринята попытка разработать алгоритм извлечения текстовых фрагментов (лексических и синтаксических единиц) из текста, предположительно связанных отношением типа «часть – целое» (партитивные отношения) и «причина – следствие» (каузативные отношения), на основе так называемых тегов или маркеров, которые сигнализируют о присутствии в тексте семантической связи данного типа, дополненных собственными правилами вывода, а также набором шаблонов или паттернов, которые выражают подобное отношение. Такой алгоритм (равно как и другие существующие) не может обеспечить нам стопроцентной точности, однако способен значительно облегчить анализ текста, а также помочь проектировщику продвинуться в построении онтологической модели.

Извлечение партитивных отношений

Некоторые авторы выделяют несколько видов отношения типа «часть – целое», каждому из которых свойственны различные способы выражения в языке. Так, в [106] приводится шесть таких видов:

1) компонент – целый объект (педаль – велосипед);
2) участник – множество (корабль – флот);
3) порция – целый объект (долька – апельсин);
4) материал – объект (сталь – машина);
5) функция – деятельность (платить – ходить по магазинам);
6) место – область (Москва – Россия).

В нашей работе мы приняли решения проигнорировать разделение на подтипы, поскольку это не является принципиальным для нашей прикладной задачи.

В русском языке отношение типа «часть – целое» может быть выражено с помощью нескольких предложно-падежных конструкций [107], из которых основными являются три:

1) Генитивные конструкции – именные словосочетания, в котором зависимое слово (чаще всего оно стоит на втором месте) имеет форму родительного падежа: *концепт онтологии, определение концепта*;

2) Именные конструкции с предлогами «у» или «от»: *группа у словаря, определение от концепта*;

3) Локативные именные конструкции с предлогами «на» и «в»: *концепт в онтологии, элемент на рисунке.*

Кроме того, в качестве тегов будем рассматривать предикаты (т.е. глаголы-связки) и некоторые существительные, которые в силу своего значения сигнализируют о наличии связи типа «часть – целое» (лексические маркеры).

Типичными лексическими маркерами – предикатами, обозначающими отношение «целое и его части», являются различные глаголы и глагольные конструкции (они были собраны на основе лингвистической онтологии RuWordNet [108-111]), которые можно разделить на два типа:

1) Маркеры целого: *составлять, состоять из, иметь, быть, подразделяться на, делиться на, содержать (в себе), заключать в себе включать (в себя), иметь в составе, насчитывать*;

2) Маркеры части: *входить в (состав/число), насчитываться в, образовать, образовывать, быть/оказаться в числе, быть (в том числе в нулевой форме) / являться / стать частью / долей / компонентом / фрагментом / деталью / комплектующей, быть (чаще всего в нулевой форме) куском / участком*.

Извлечение причинно-следственных связей

Было проанализировано несколько теоретических работ [112-114], посвященных способам выражения причинно-следственных отношений, на

основании чего сформирована следующая классификация и соответствующие правила выделения связанных фрагментов, представленные в таблице 3.2.

Таблица 3.2 - Классификация тегов, выражающих причинно-следственные отношения в тексте

Тип тега	**Правила выделения фрагментов**	**Примеры тегов**
Маркирует причину	Чаще всего стоит между следствием и причиной. В качестве причины следует рассматривать все предложение до точки, запятой или другого знака препинания. В качестве следствия – начало предложения (если тег встречается посередине предложения), или оставшуюся часть предложения между запятой и точкой (если тег стоит в начале предложения). *Y, <A> X. или <A> X, Y.*	*потому что, поскольку, так как*
	Чаще всего стоит между следствием и причиной. В качестве причины следует рассматривать лексическую группу после тега. В качестве следствия – начало предложения (если тег встречается посередине предложения), или оставшуюся часть предложения между запятой и точкой (если тег стоит в начале предложения). *Y, <B> [X] (…). или <B> [X] (…), Y.*	*из-за, в результате, благодаря*
	Сказуемое (предикат), относящееся к причине. Подлежащее следует	*вызывать, приводить к, быть/стать*

	рассматривать как причину. Именную группу после маркера – как следствие. *<C> (X, Y)* * В пассивном залоге (при наличии формы пассивного залога) – см. тип F.	*причиной, обеспечивать*
Маркирует следствие	Чаще всего стоит в начале предложения и маркирует следствие (до знака препинания). При этом причина может содержаться в предыдущем предложении (если предложение, промаркированное тегом, простое), или в начале предложения (если тег стоит в середине). *X. <D> Y(, ...). или X, <D> Y.*	*следовательно, таким образом, по этой причине, соответственно*
	Чаще всего стоит между причиной и следствием. В качестве следствия следует рассматривать лексическую группу после тега. В качестве причины – начало предложения (если тег встречается посередине предложения), или оставшуюся часть предложения между запятой и точкой (если тег стоит в начале предложения). *X, <E> [Y] (...). или <E> [Y] (...), X.*	*чтобы, для того чтобы*
	Сказуемое (предикат), относящееся к следствию. Подлежащее следует рассматривать как следствие. Именную группу после маркера – как причину. *<F> (Y, X)*	*следовать из*

Маркирует и причину, и следствие	Парные теги. Первый тег маркирует причину, второй – следствие *(…) <G1> X, <G2> Y (…).*	*так… что…*

3.3 Модуль генерирования UML-диаграмм

Для получения спецификаций целевых проектных решений на основе онтологических моделей могут использоваться инструменты автоматизированного генерирования UML-диаграмм – одним из наиболее широко используемых в таких целях на сегодняшний день является PlantUML [115], который гибко подходит к нотации UML: «поддерживая все основные ее элементы, он дает пользователю множество возможностей для их свободного использования в диаграмме» [116].

Для генерирования UML-диаграмм необходимо установить соответствия между отношениями в ОМ и возможными связями между сущностями, характерными для различных типов UML-диаграмм. В таблице 3.3. в демонстрационных целях представлены такие соответствия, сформированные для диаграммы классов и диаграммы прецедентов.

Таблица 3.3. – Соответствия связей диаграммы классов и отношений в ОМ

Тип UML-диаграммы	**Тип связи UML-диаграммы**	**Тип связи в онтологии**
Диаграмма классов	быть атрибутом класса	атрибут
	быть операцией (методом) класса	функция
	Зависимость / Ассоциация	Любое отношение между классами, за исключением ассоциации, обобщения и реализации.

	Агрегация	часть – целое
	Композиция	часть – целое
	Обобщение	род – вид, наследование
	Реализация	реализация
Диаграмма прецедентов	Ассоциация	Любые отношения между актором (концептом, определяемым вручную) и другим объектом онтологии (который становится прецедентом)
	Расширение	–
	Обобщение	род – вид
	Включение	часть – целое

Пример 2. Продемонстрируем возможности генерирования ***диаграммы вариантов использования*** на основе фрагмента онтологической модели (см. рис. 3.7) с применением инструмента PlatUML.

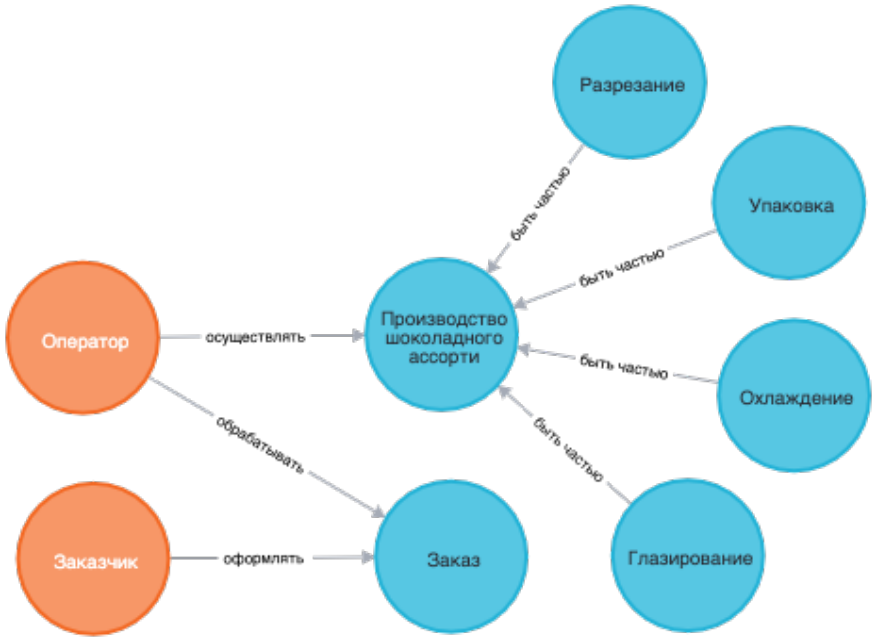


Рисунок 3.6 - Фрагмент онтологической модели

1) Выявим основных акторов: Заказчик и Оператор.

2) Извлечем из ОМ связи, в которых участвуют данные концепты. Связанные с ними другие концепты будем относить к прецедентам.

3) Примем данные связи за отношения ассоциации, включив семантику связи в состав прецедента. Обратим внимание, что концепт Заказ связан как с актором Оператор, так и с актором Заказчик:

- оформлять (Заказ) и обрабатывать (Заказ) -> Оформление и обработка заказа
- осуществлять (Производство шоколадного ассорти) -> Осуществление производства шоколадного ассорти

4) Поскольку связи проектной онтологии типа «часть – целое» могут трансформироваться в отношения включения прецедента, участвующего в диаграмме вариантов использования, извлечем из онтологии связи типа «часть – целое»;

5) Переведем спецификации онтологии, названные выше, в спецификации диаграммы (СД) в соответствии с синтаксисом инструмента plantUML (см. листинг 3.2). На рис. 3.8 представлена сгенерированная диаграмма прецедентов.

Листинг 3.2 - Спецификация диаграммы прецедентов

```
@startuml
left to right direction
skinparam shadowing false
actor Заказчик
actor Оператор
Заказчик -- (Оформление\nи обработка заказа)
Оператор -- (Оформление\nи обработка заказа)
Оператор -- (Осуществление производства\nшоколадного ассорти)
(Осуществление производства\nшоколадного ассорти) .> (Разрезание) : include
(Осуществление производства\nшоколадного ассорти) .> (Глазирование) : include
(Осуществление производства\nшоколадного ассорти) .> (Охлаждение) : include
(Осуществление производства\nшоколадного ассорти) .> (Упаковка) : include
@enduml
```

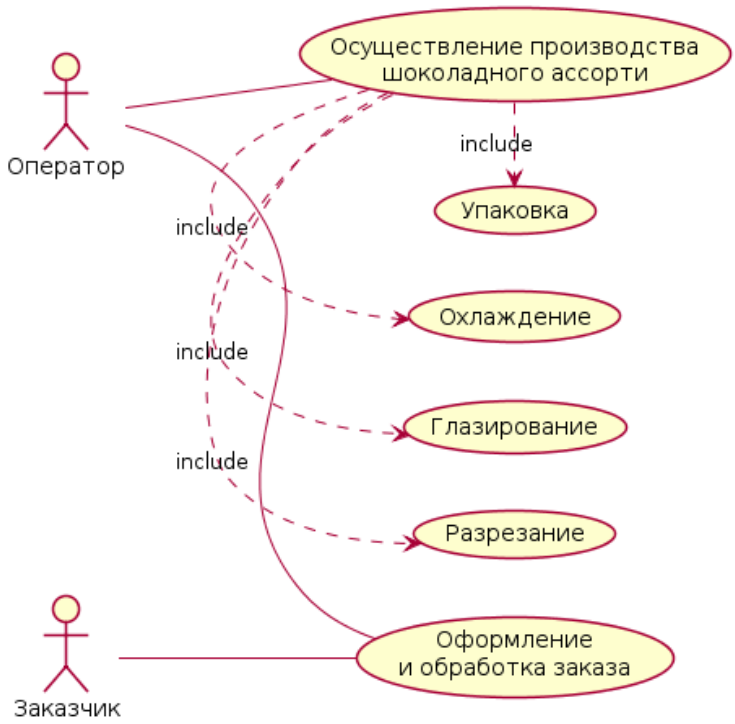


Рисунок 3.7 – Сгенерированная диаграмма прецедентов

3.4 Модуль генерирования кода

Модуль генерирования кода использует данные из онтологической модели реализации для генерирования исходного кода программы в соответствии с заданными правилами.

Выбор языка C в качестве языка программирования для генерирования исходного кода на основе ОМ обусловлен тем фактом, что объектом проектирования в нашем случае является система логического управления.

Программная реализация генератора кода осуществляется с использованием библиотеки ***csnake*** [117]. Обобщенная схема работы генератора представлена на рис. 3.8.

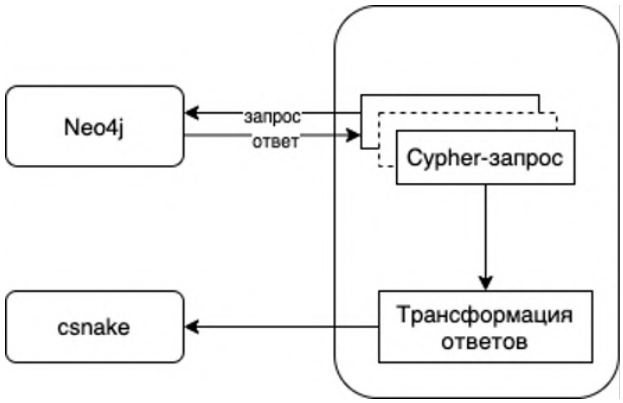


Рисунок 3.8 - Структура модуля генерации кода

Продемонстрируем его работу на примере генерации конструкции SWITCH. В начале необходимо создать экземпляр класса CodeWriter. Для объявления конструкции SWITCH служат несколько методов данного класса:

- start_switch (начало конструкции);
- add_switch_case (добавление нового условия);
- add_switch_break (добавление оператора break).

Кроме того, для генерации используются следующие методы класса CodeWriter:

- add_line (добавление новой строки);
- indent (добавление отступа);
- close_brace (закрытие скобок);
- add_enum (объявление переменной перечисляемого типа);
- add_function_definition (объявление функции).

Для объявления переменной перечисляемого типа используется экземпляр класса Enum. Добавление новых значений осуществляется с помощью метода add_values (в нашем случае в нее записываются состояния).

Для объявления функций используется экземпляр класса Function и его метод add_code (добавление новых строк кода).

При добавлении новых строк в метод add_line можно передать необязательный параметр comment, который служит для добавления комментария к коду. Поскольку онтологическая модель СЛУ содержит также

и имена сущностей на естественном языке, то целесообразно использовать их в качестве комментариев.

Ниже на псевдокоде представлен алгоритм генерирования исходного кода, реализующего функцию логического управления АС с состояниями, на основе онтологических спецификаций:

States.apply(declare_enum) // перечисление состояний

Functions.apply(объявление_функции(

States.apply(объявление_оператора_switch(

исходное_состояние

действия_до_перехода.apply(add_line)

целевые_состояния.apply(

Predicates.apply(add_line)

целевое_состояние

)))))

В листинге 3.2 представлен фрагмент генератора кода, реализующий генерацию конструктора SWITCH.

Листинг 3.2 - Фрагмент генератора кода

```
from csnake import CodeWriter, Function, Variable, Enum


def add_process_lines(p_cwr, df):
    for _, row in df.iterrows():
        line = row['codename']
        if 'Function' in row['labels']:
            line += '()'
        p_cwr.add_line(line, comment=row['name'])


def get_code():
    cwr = CodeWriter()

    # состояния
    states = get_states()
    enum = Enum('STATES')
    enum.add_values(states.keys())
    cwr.add_enum(enum)

    # switch
    cwr_switch = CodeWriter()
    var = Variable('state', 'str')
```

```
    cwr_switch.start_switch(var)

    for key_state in states:
        # состояние, из которого осуществляется переход
        state_var = Variable(key_state, 'str')
        cwr_switch.add_switch_case(state_var)

        # действие, которое необходимо выполнить до перехода
        pr_before = get_operations_before_transition(states[key_state])
        add_process_lines(cwr_switch, pr_before)

        sts_to_transit = get_states_to_transit(states[key_state])

        for key_state_2 in sts_to_transit:
            # условие перехода
            cond, cond_comm = get_condition(states[key_state],
sts_to_transit[key_state_2])
            cwr_switch.add_line('if (%s) {' % cond, comment=cond_comm)
            cwr_switch.indent()

            # изменение состояния
            cwr_switch.add_line('state = %s;' % key_state_2)

            # действие, которое необходимо выполнить после перехода
            pr_after = get_operations_after_transition(cond_comm)
            add_process_lines(cwr_switch, pr_after)
            cwr_switch.close_brace()

        cwr_switch.add_switch_break()

    # объявляем функцию step()
    step_fun = Function('step', return_type='void')
    step_fun.add_code(cwr_switch)
    cwr.add_function_definition(step_fun)
```

Выводы по главе 3

В данной главе представлено описание системы онтологического сопровождения проектирования АС, в рамках которой проводилась апробация предложенных методов.

Приводится описание формальной структуры онтологических моделей требований, проектирования и реализации, а также правил логического вывода на основе информации из данных ОМ, заложенных в основу работы описанной системы. Описаны принципы работы разработанных программных модулей по обработке текстовой информации для наполнения онтологии

требований, а также модулей по генерированию UML-диаграмм и исходного кода программы автоматизации.

Средства разработаны таким образом, что позволяют автоматизировать процесс изменения требований: благодаря наличию ограничений на структуру онтологических моделей и реализации правил логического вывода, внесение изменений в онтологию требований влечет за собой управляемое изменение онтологий проектирования и реализаций. А, поскольку информация из данных онтологий используются для автоматической генерации таких артефактов проектирования, как UML-диаграммы и исходный код программы, использование средств обеспечивает сокращение трудозатрат на доработку.

Глава 4. Использование средств онтологической поддержки

В данной главе представлены эксперименты по практическому применению предлагаемых проектных решений онтологического сопровождения проектирования, а также результаты расчета по оценке эффективности применения данного подхода.

Эксперименты были проведены с целью исследования принципиальной возможности осуществления автоматизации проектного процесса на основе описанных онтологических моделей и сравнительной оценки производительности труда проектировщика.

4.1 Использование онтологического моделирования при разработке системы логического управления плиткоукладчиками

В настоящем параграфе представлена онтологическая модель системы логического управления (СЛУ) плиткоукладчиками [126], а также описаны полезные эффекты от ее применения в процессе проектирования данной системы.

Логическое управление охватывает группу роботов-плиткоукладчиков, укладывающих плитки на негоризонтальной плоскости, что требует различать движение вверх, вниз, влево и вправо, поскольку распределение нагрузок на ходовую часть при каждом виде движения и смене направлений различается. Каждый робот оснащен механизмом укладки, закрепленном на мобильной платформе с правой стороны. Это предопределяет укладку по траектории раскручивающейся спирали с вращением по часовой стрелке. Все плитки квадратные, и в логическом управлении используется дискретное координатное пространство с центром {x: 0, y: 0}, где располагается первая укладываемая плитка. Увеличению координаты y на 1 соответствует перемещение на одно плиткоместо вверх, а уменьшению – на одно плиткоместо вниз. Аналогично увеличению координаты x на 1 соответствует

перемещение на одно плиткоместо вправо, а уменьшению – на одно плиткоместо влево. Одно из требований предписывает формирование для каждого шага процесса протокольной записи следующего формата:

- момент времени начала шага с точностью до миллисекунд;
- номер плиткоукладчика; предполагается общий протокол для всей группы плиткоукладчиков;
- порядковый номер укладываемой плитки; нумерация начинается с 0;
- номер витка спирали; нумерация начинается с 1;
- значение координаты x текущей плитки;
- значение координаты y текущей плитки;
- состояние процесса на данном шаге (направление движения, либо подготовительные состояния);
- состояние автомата, в которое переходит автомат.

Согласно подходу, предлагаемому в диссертационном исследовании, сформируем композицию, состоящую из трех онтологий: онтологию требований, онтологию проектных решений и онтологию реализации.

4.1.1 Формирование онтологии требований

Согласно модели проектного процесса, описанной в параграфе 2.2, процесс построения онтологии требований начинается с анализа требований, представленных в нашем примере множеством неформальных описаний объекта управления *Source*, которые чаще всего являются текстами на естественном языке. Ниже – примеры таких описаний:

Source1: Существует некоторый процесс инициализации, за которым могут скрываться последовательные подпроцессы загрузки контейнера с плитками и подпроцесс движения к начальной точке монтажа. Начало этого процесса задается некоторым внешним воздействием – старт укладки плитки.

Source2: Необходимо установить таймаут – ограничение на время пребывание в состоянии инициализации.

Source3: Необходимо формировать протокольную запись, содержащую информацию о старте и завершении процесса инициализации с точностью до миллисекунд с указанием идентификатора плиткоукладчика.

Последовательный анализ множества *Source* приводит нас к формированию формальных спецификаций онтологии требований.

На рис. 4.1 показан результат формирования онтологической модели требований в СУБД Neo4j на основе множества *Source* с использованием разработанных средств автоматизации. Принадлежность концептов к множествам *Inputs, Outputs, Actions, Events, Messages, Processes, Protocols, Conditions* задано с помощью соответствующих меток (*labels*).

4.1.2 Формирование онтологии проектирования и онтологии реализации как процесс последовательных трансформаций

Одной из составляющей практической ценности настоящей диссертации является создание таких средств трансформации онтологических спецификаций, которые обеспечивают связывание онтологии требований с онтологиями проектирования и реализации и, в комплексе со средствами автоматизированного формирования онтологии проекта и средствами автоматической генерации кода на основе онтологии направлены на повышение автоматизации доработки системы при возникновении необходимости изменения требований и перепроектирования соответственно.

На вход модулю формирования онтологии проектирования поступают ***формализованные требования***, формальная структура которых определяется в том числе типом задействованных, согласно требованиям к проекту, проектных решений. Так, в случае разработки СЛУ плиткоукладчиками формальная структура требований сводится к описанию следующих сущностей (часть этих сущностей наследуется из онтологии требований):

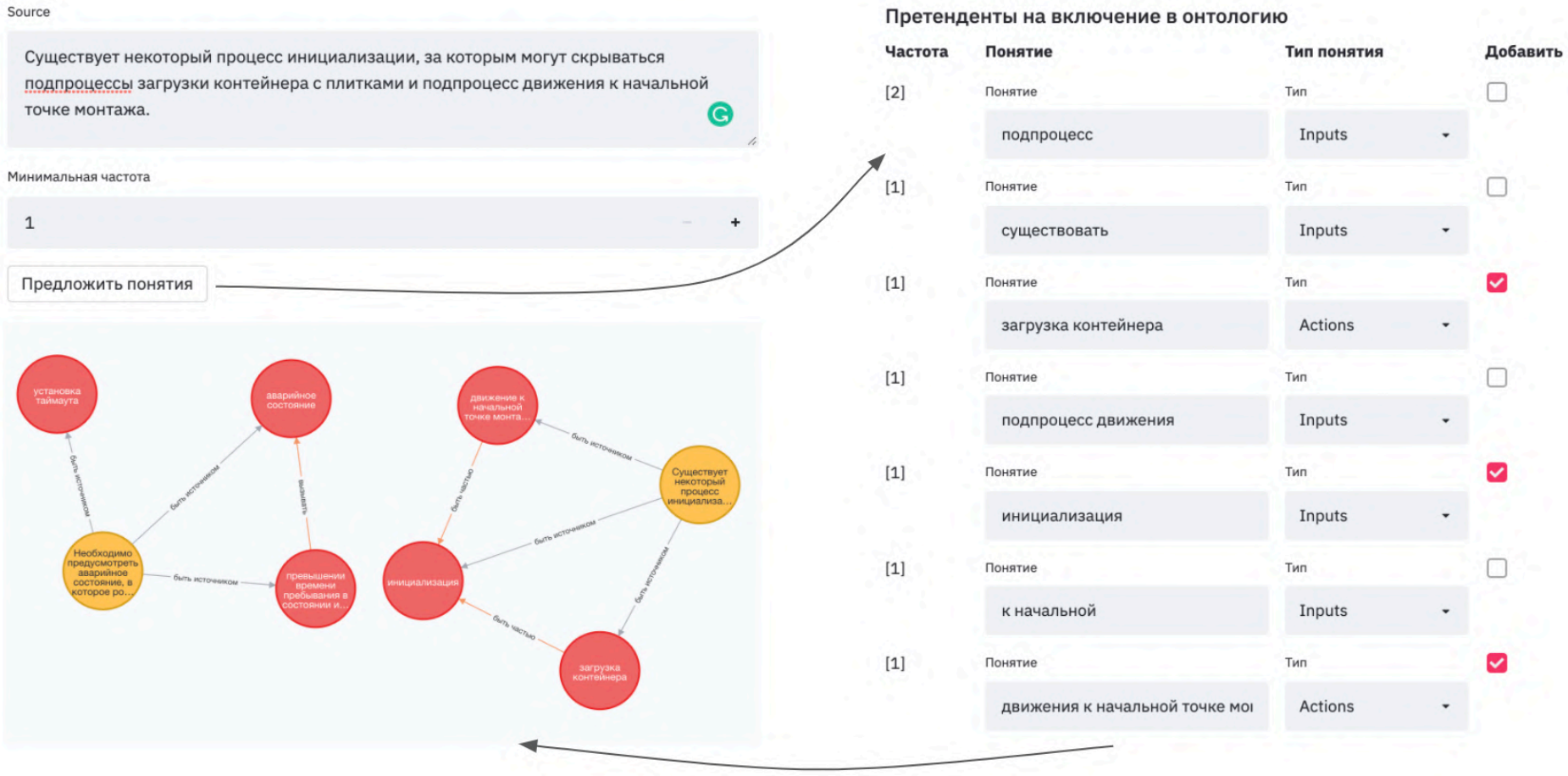


Рисунок 4.1 – Процесс формирования онтологии требований

- исходное состояние;
- результирующее состояние;
- условия перехода из исходного состояние в результирующее;
- действия или процессы, которые необходимо осуществить до перехода;
- действия или процессы, которые необходимо осуществить после перехода.

Процесс генерирования таких формализованных требований с использованием разработанных средств представлен в на рис. 4.2.

Как видно на рисунке, на основании данной формализации модуль формирования онтологии автоматически генерирует онтологию проектирования и онтологию реализации с занесением соответствующей информации в базу данных Neo4j. Всякий раз, когда в данную структуру вносятся изменения, модуль формирования онтологии может быть запущен повторно, в результате чего в онтологии проекта будут ***изменены сущности*** (добавлены новые сущности, удалены неиспользуемые сущности, изменены имена сущностей).

Модуль формирования онтологии генерирует и изменяет ***онтологию проектирования*** в соответствии с правилами логического вывода типа:

- ***если*** существуют два связанных состояния (состояние А и состояние В), из и в которое необходимо осуществить переход соответственно, ***то*** необходимо
    - добавить вспомогательную сущность «Переход из состояния А в состояние В»;
    - связать данную сущность с состоянием А отношением «быть переходом из» и с состоянием В отношением «быть переходом в»;
    - добавить отношение «переходить в» между состоянием А и состоянием В;

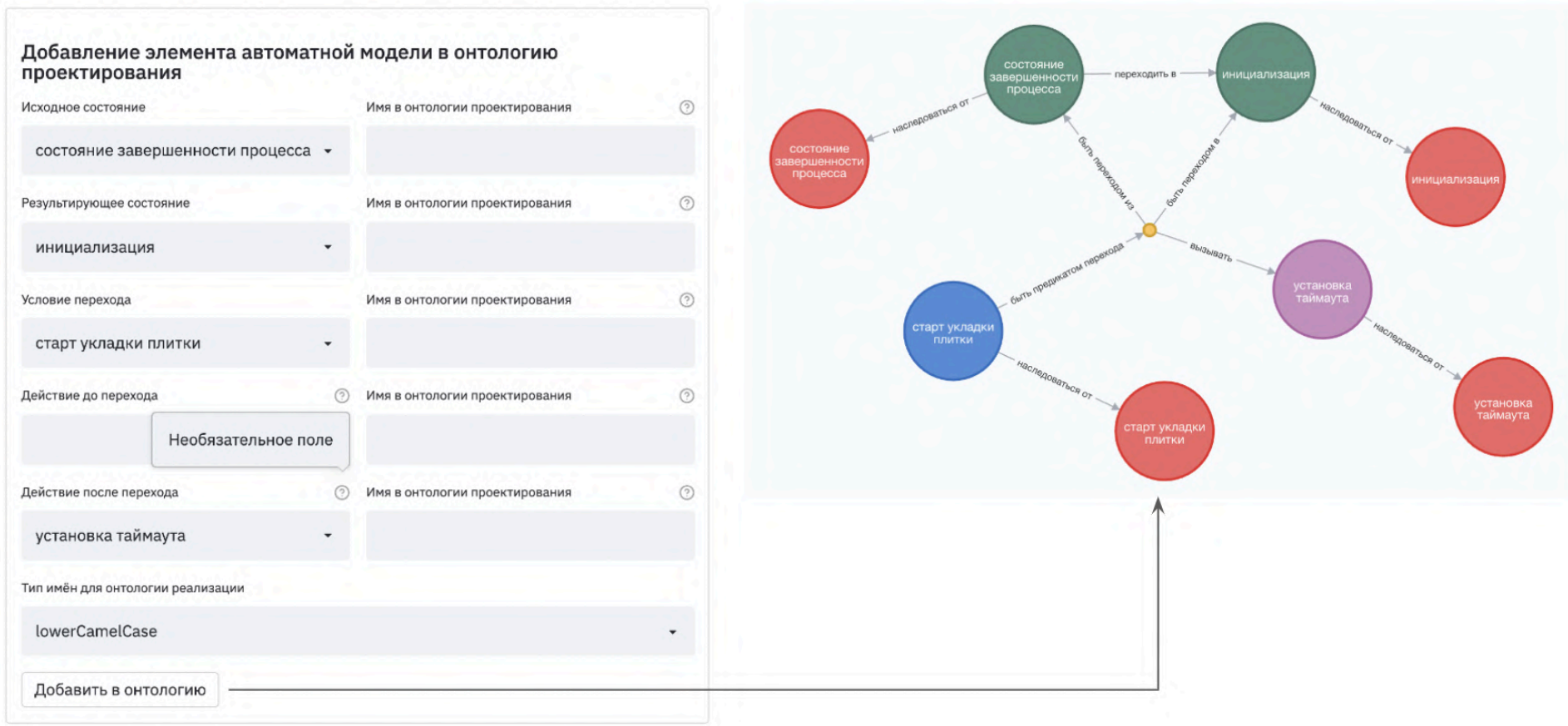


Рисунок 4.2 - Процесс формирования онтологии проектирования

- а также ***если*** существует условие перехода из состояния A в состояние B, ***то*** необходимо
    - добавить отношение «быть предикатом перехода» между условием перехода и новой сущностью «Переход из состояния A в состояние B»;
- а также ***если*** существуют действия A (которые необходимо выполнить до перехода), ***то*** необходимо
    - добавить отношение «предшествовать» между состоянием A и каждым действием A (которое необходимо выполнить до перехода), а также между каждым действием A и новой сущностью «Переход из состояния A в состояние B»;
- а также ***если*** существуют действия B (которые необходимо выполнить после перехода), ***то*** необходимо
    - добавить отношение «вызывать» между новой сущностью «Переход из состояния A в состояние B» и каждым действием B (которое необходимо после перехода), а также отношение «предшествовать» между каждым действием B и состоянием B;
- ***если*** существуют и действия A, и действия B, ***то*** необходимо
    - добавить отношение «предшествовать» между каждым действием A и каждым действием B.

Алгоритм формирования ***онтологии реализации*** заключается в том, что для каждой сущности онтологии проектирования (за исключением вспомогательных) необходимо создать соответствующую сущность в онтологии реализации, связанную с первой отношением «быть реализованным» и обладающую следующими свойствами:

- имя на уровне реализации (необязательный параметр; если не задано, то наследуется от онтологии проектирования);

- имя, используемой в исходных кодах программ (обязательный параметр; может быть сгенерировано автоматически с использованием средств автоматического перевода);
- тип (функция, операция, параметр).

На рисунке 4.3 показан фрагмент онтологии проекта, описывающий переход из состояния «Движение вперед» в состояние «Движение вправо», которая сгенерирована на основе описанных выше правил и алгоритмов. При этом белым отмечены сущности онтологии требований, серым – сущности онтологии проектных решений, а черным – сущности онтологии реализации.

Поскольку онтология реализации используется для генерирования исходного кода программы СЛУ и, как было описано выше, ее сущности генерируются на основе онтологии проектирования, формируемой, в свою очередь, на основе формализованных требований; то **автоматическая модификация исходного кода также возможна при внесении изменений в требования**.

Так, например, в требования может потребоваться внести следующее изменение:

Source^{N}*:* перед переходом из состояния «Движение вперед» в состояние «Движение вправо», необходимо переключить передачу на более высокую, поскольку робот осуществляет движение по наклонной плоскости, и движение вперед (вверх) требует пониженной передачи.

В этом случае в онтологию требований и проектирования добавится сущность типа *Action* «Изменить скорость» с описанием – «Переключить передачу на более высокую», а в онтологию реализации – новая сущность типа *Function* (функция) *changeSpeed*, связанная с сущностью «Изменить скорость» отношением «быть реализованным». После чего можно будет запустить скрипт генерации программного кода на основе онтологии проекта и получить модифицированный код.

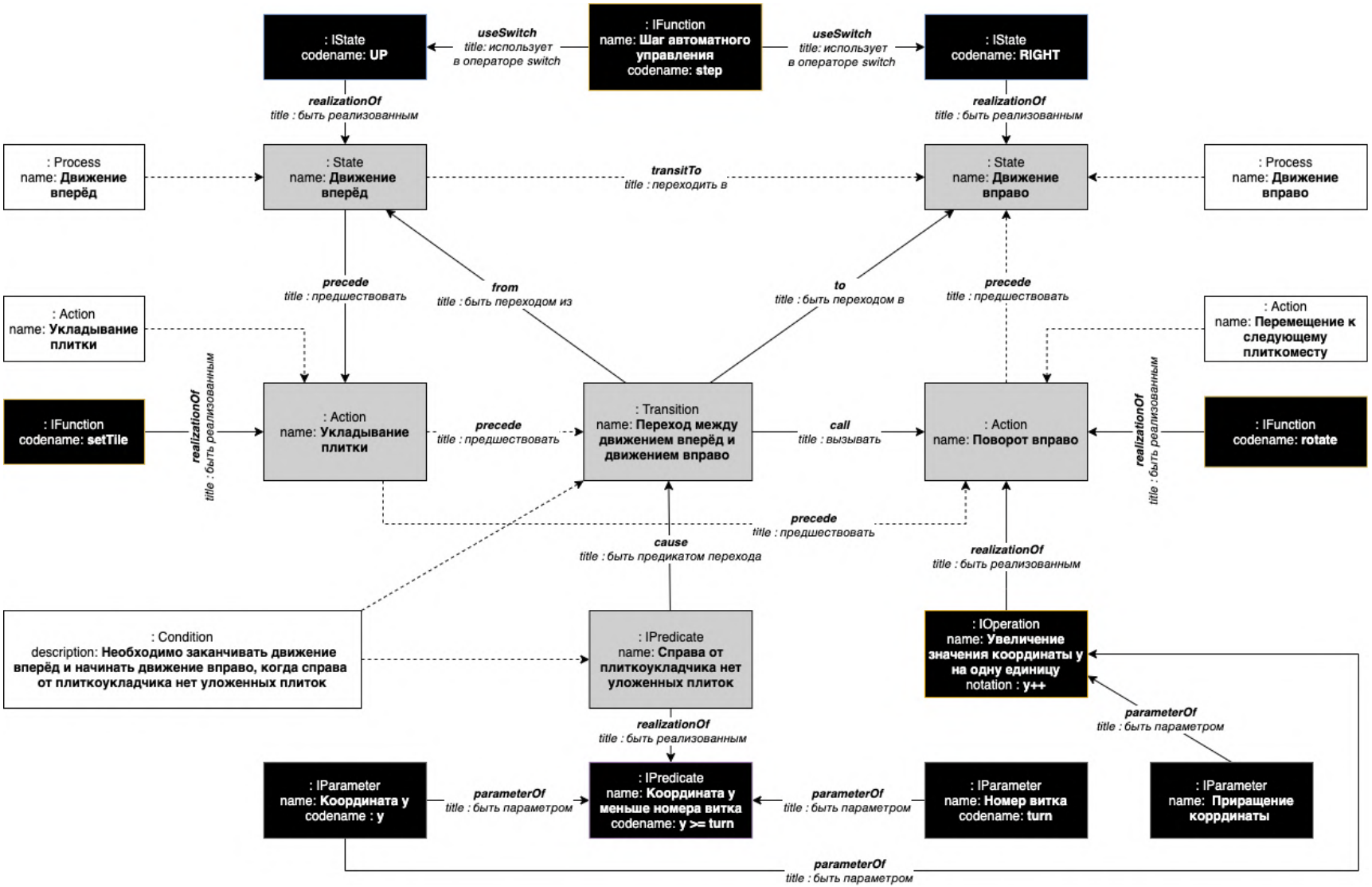


Рисунок 4.3 - Фрагмент онтологии СЛУ пликтоукладчиками

4.1.3 Генерация UML и исходного кода программы

Онтологические спецификации проекта, описанные выше, пригодны для автоматической генерации диаграммы состояний через реализацию следующего алгоритма:

- сгенерировать описание диаграммы состояний в соответствии с синтаксисом PlantUml:
  - обозначить начало диаграммы с помощью конструкции @startuml;
  - извлечь из базы данных онтологии проекта все состояния States;
  - для каждого состояния State:
    - извлечь из базы данных онтологии проекта действия Actions*, которые необходимо выполнить до перехода из текущего состояния;
    - сформировать соответствующую строку описания состояния;
    - извлечь из базы данных онтологии проекта все возможные переходы Transitions, результирующие состояния States* и условия перехода Predicates;
    - для каждого перехода:
    - сформировать соответствующую строку описания перехода;
  - обозначить конец диаграммы с помощью конструкции @enduml;
- сгенерировать изображение с помощью утилиты PlantUml.

Результат работы Python-скрипта по генерированию диаграммы состояний показан на рис. 4.4.

Кроме того, на основе онтологии реализации возможна генерация исходного кода функции на языке C, реализующий шаг автоматного управления (step).

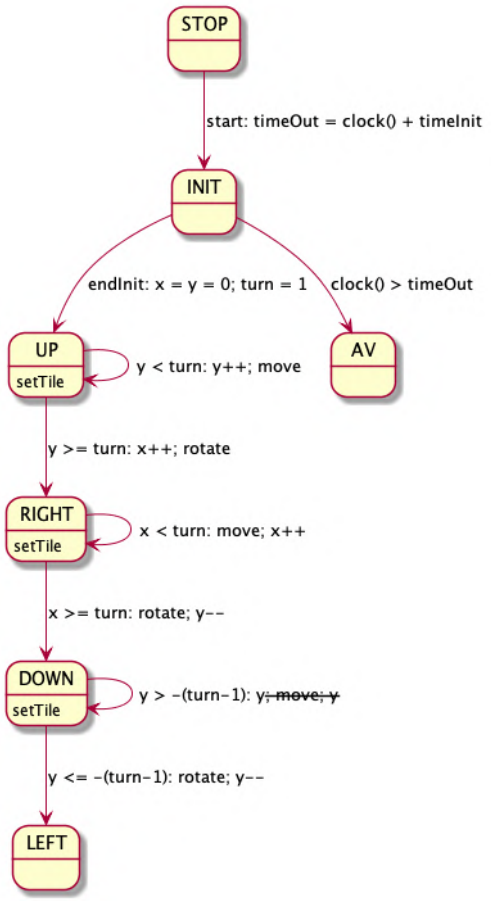


Рисунок 4.4 – Сгенерированная диаграмма состояний

Для обеспечения генерации кода используется набор Cypher-запросов на чтение данных из онтологии реализации (запросы реализованы в формате отдельных функций на языке Python), выполняемых и обрабатываемых согласно следующему алгоритмы:

- объявить функцию;
- получить все состояния уровня реализации States (функция get_states);
- объявить оператор switch;
- для каждого состояния:
  - объявить новую ветвь исполнения;
  - получить операции Actions*, которые необходимо выполнить до осуществления перехода из текущего состояния (функция get_operations_before_transition, которая принимает на вход имя

текущего состояния) и сгенерировать соответствующие строки кода;

- получить состояния States*, в которые может быть осуществлен переход из текущего состояния (функция get_states_to_transit, которая принимает на вход имя текущего состояния); для каждого такого состояния:
  - получить условие перехода (функция get_condition, которая принимает на вход имена состояний);
  - объявить условный оператор;
  - получить список операций, которые необходимо выполнить после осуществления перехода (функция get_operations_before_transition, которая принимает на вход имя предиката перехода) и сгенерировать соответствующие строки кода;

В таблице 4.1 представлен фрагмент кода функции, реализующей шаг автоматного управления на языке С в двух вариантах: сгенерированный модулем автоматической генерации кода на основе ОМ и написанный программистом вручную. Как видно из сравнения, сгенерированный код обладает некоторой избыточностью (в случае условного ветвления и в случае возникновения «петли» – перехода из состояния в себя), а также может нуждаться в небольшой доработке (что локализуется при тестировании).

Дополнительным позитивным эффектом использования разработанных средств автоматизации является возможность автоматического комментирования кода на основе данных из онтологии требований и проектных решений.

Таблица 4.1 - Сравнение сгенерированного кода с написанным вручную

Сгенерированный код	Написанный программистом код
void step(void){ switch (state){ case UP: setTile(); /* Укладывание плитки */ if (y < turn) { /* Справа от плиткоукладчика есть уложенные плитки */ **state = UP;** y++; /* Увеличение значения координаты y на одну единицу */ move(); /* Перемещение к следующему плиткоместу */ } **if (y >= turn) { /* Справа от плиткоукладчика нет уложенных плиток */** state = RIGHT; x++; /* Увеличение значения координаты x на одну единицу */ rotate(); /* Поворот */ } break; case STOP: **if (!start) { /* Нет сигнала на старт укладки плитки */** **state = STOP;** **}** if (start) { /* Получен сигнал на старт укладки плитки */ state = INIT; init(); /* Выполнение инициализации */ timeOut = clock() + timeInit; /* Установка ограничения на пребывание в состоянии инициализации */ } break;	void step() { switch (state){ case UP: setTile(); // установка плитки if (y < turn) { y++; move(); } **else {** state = RIGHT; x++; rotate(); } break; case STOP: // если есть сигнал start, то запустить инициализацию if (start) { state = INIT; timeOut = clock() + timeInit; init(); } break;

Таким образом, при возникновении необходимости в перепроектировании вследствие обнаружения неточностей или изменения требований заказчика, внедрение в проектный процесс ОМ существенно снижает затраты, поскольку обеспечивает повышение автоматизации как на стадии концептуального проектирования (генерирования диаграммы состояний), так и на стадии реализации (частичная генерация кода). Кроме того, использование ОМ обеспечивает концептуальную целостность разрабатываемой системы за счет поддержки многих пространств имен.

4.1.4 Сравнительный анализ производительности труда проектировщика с учетом использования методов и средств онтологического сопровождения проектирования

Для оценивания степени автоматизации процесса онтологического сопровождения проектирования оценим каждую проектную операцию по технике ***«покер планирования»***. Данная техника предложена Джеймсом Греннингом [124] в 2002 году для оценивания сложности задач, решаемых при разработке ПО в рамках гибкой методологии разработки. Техника широко применяется при работе проектных команд разработки сложных систем с программным обеспечением в крупных организациях, таких как Amazon, IBM, Tesla и др. [125] Преимуществом данной техники является то, что с ее помощью можно оценить и сравнить задачи, выполняемые разными специалистами, включая те, которые не предполагают написание кода.

Суть техники заключается в том, что за условную единицу принимается задача ***Z0***, которая содержит в себе минимальное количество неопределенности: оценка ее трудоемкости принимается за ***одну единицу сложности (SP, Story Point)*** с ориентацией на такие составляющие сложности, как оценочное время выполнения задачи, необходимый набор компетенций, требуемое инструментальное оснащение и некоторые другие. Оценка других задач выставляется с ориентацией на их сложность относительно задачи ***Z0***. При оценивании задач чаще всего используются числа из ряда Фибоначчи, дополненная значением 0,5 (0, 0.5, 1, 2, 3, 5, 8, 13).

Для сравнения трудозатрат проектировщика на разработку АС с использованием методов и средств онтологического сопровождения и без их использования примем за одну единицу сложности (1 SP) проектную операцию, соответствующую ***описанию одного требования*** к логике функционирования разрабатываемой системы. В контексте рассматриваемого проекта одна единица сложности примерно соответствует одному человеко-часу работы соответствующего специалиста. В Приложении 4 в форме

таблицы представлены оценочные показатели трудозатрат на выполнение различных проектных процедур, являющихся частью проектного процесса по разработке АС, в которые целесообразно вовлекать онтологическое моделирование, – без использования методов и средств онтологического моделирования и с их использованием. Таблица дополнена пояснениями различий в работе проектировщика в случаях, если он использует методы и средства онтологического моделирования и если он их не использует.

В качестве переменных (множителей) введены ***метрики***, от величины которых зависит трудозатраты на проектную процедуру, частью которой является указанная проектная операция (например, трудозатраты на процедуру описания требований к системе зависит от количества таких требований). В ходе эксперимента данные переменные принимали следующие значения: количество требований – 24; количество задействованных прототипов – 0; количество разработанных UML-диаграмм – 1; количество сущностей ОМ на уровне реализации – 50; количество функций программы – 1.

Тогда сокращение трудозатрат можно представить в виде следующей функции от количества измененных требований, которая выражает долю трудозатрат, сэкономленных за счет онтологического моделирования, от общих (без вовлечения онтологий). Для удобства расчетов будем считать все изменения равнозначными.

$$f(x) = 1 - \frac{E_O^P + E_O^R x}{E^P + E^R x}, \tag{4.1}$$

где E^P и E_O^P– количество трудозатрат на первую итерацию проектного процесса (до необходимости внесения изменений в требования) без использования онтологического моделирования и с его использованием соответственно; E^R и E_O^R– количество трудозатрат на доработки.

При использовании методов и средств онтологического моделирования требуются дополнительные трудозатраты на разработку онтологических моделей, в особенности на этапе формирования и анализа требований,

поэтому в начале проектного процесса трудозатраты специалистов, использующих онтологическое моделирование, могут превышать трудозатраты специалистов, работающих без использования предложенной методологии. Однако, при возникновении необходимости внесения изменений в требования, что всегда происходит на практике, сформированные ***онтологические модели значительно снижают трудозатраты на доработку системы***: и, чем больше требований требуется изменить, тем более заметны позитивные эффекты от онтологического моделирования в части повышения производительности труда (см. рис. 4.5).

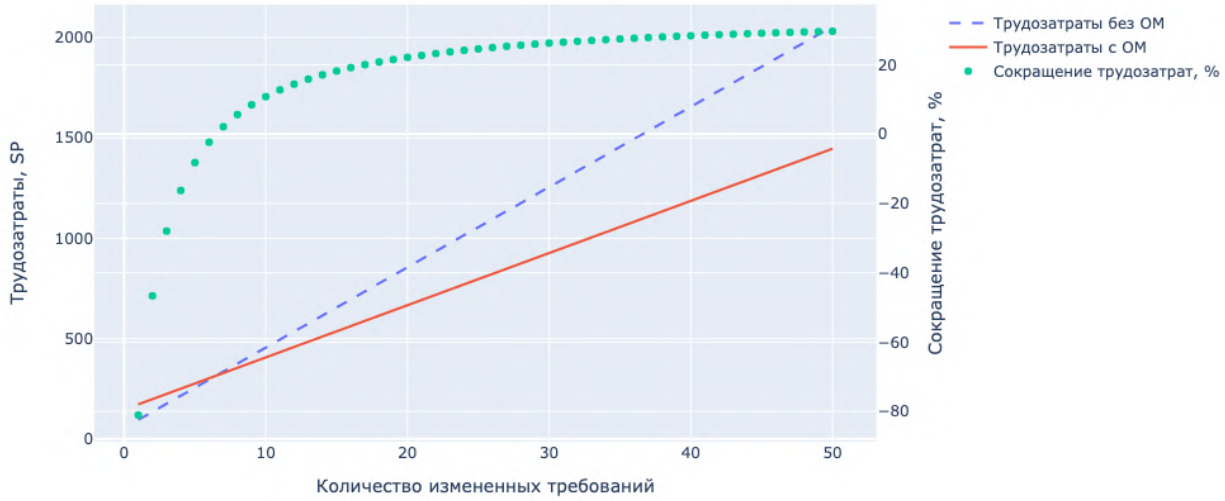


Рисунок 4.5 - Сокращение трудозатрат на проектирование

Так, в рамках эксперимента по проектированию СЛУ плиткоукладчиками продемонстрировано, что при необходимости изменить 7 и более требований трудозатраты на онтологическое моделирование нивелируются средствами автоматизации, а ***при внесении 50 изменений сокращение трудозатрат достигает 30%***.

Для сравнения трудозатрат проектировщика на онтологическое моделирование АС с использованием средств автоматизации и без их использования примем за условную единицу (1 SP) проектную операцию ***выделения одного объекта (понятия) онтологической модели из множества описаний объекта автоматизации***. Тогда оценку остальных

проектных операций, относящихся к проектному процессу онтологического сопровождения проектирования в части разработки онтологических моделей и их модификации, можно представить в виде таблицы с аналогичной структурой (см. Приложение 5).

В эксперименте по проектированию СЛУ плиткоукладчиками указанные в таблице переменные принимали следующие значения: объем документации – 7 000 слов; количество объектов – 200; количество агрегатов – 20; количество связей – 300; количество правил – 30 правил; количество измененных сущностей в ходе модификации требований = 50. Согласно этим данным и таблице, представленной в Приложении 5, трудозатраты без использования средств автоматизации составляют 1826 SP, а с использованием средств автоматизации – 811,8 SP. Таким образом за счет использования разработанных средств достигается сокращение трудозатрат почти в **56%** (см. рис. 4.6).

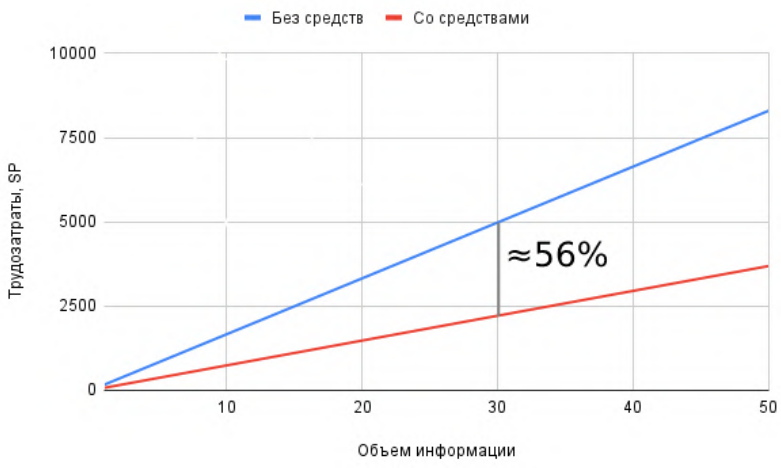


Рисунок 4.6 - Сокращение трудозатрат за счет автоматизации онтологического моделирования

4.2 Создание средств поддержки проектирования прототипов базовых средств программного управления транспортными роботами на основе онтологического моделирования

В данном параграфе на реальном примере (средства управления транспортными роботами) продемонстрировано, как применение технологии онтологического моделирования на ранних фазах прототипирования может привести к сокращению трудозатрат в условиях динамически изменяющихся требований к разрабатываемому продукту.

Задача сформулирована следующим образом:

Z1. Создать средства поддержки проектирования прототипов средств управления транспортными роботами, используемых на этапе анализа требований.

Z2. Оформить концептуально важные требования в таком формате онтологических спецификаций, который может интерпретироваться средствами поддержки быстрого прототипирования, а также разработать средства трансформации данных спецификаций в формат данных исполняющей схемы.

Особенности решения подзадачи ***Z1*** описаны в [118]. Средства прототипирования реализованы средствами веб-технологий и обеспечивают «имитацию движения на основе спецификаций траекторий в двумерном пространстве и организуются как параметрически управляемый процесс анимации групп графических примитивов» [118]. В данном параграфе подробнее остановимся именно на средства онтологической поддержки, речь о которых идет в подзадаче ***Z2***.

На рис. 4.7 представлена структурно-функциональная схема средств прототипирования с использованием средств онтологического моделирования. Базовая структурно-функциональная организация данных средств (без инструментов онтологического моделирования описана) в [119]. Набор средств реализован по клиент-серверной архитектуре. «Все

компоненты, расположенные ниже штриховой линии, относятся к клиентской части, реализованной средствами языка JavaScript. Программы реализации функций логического управления на стороне сервера разрабатываются на любом языке высокого уровня» [119]. Модули, относящиеся к онтологическому сопровождению, реализованы на языке Python; для хранения онтологической модели используется графовая СУБД Neo4j.

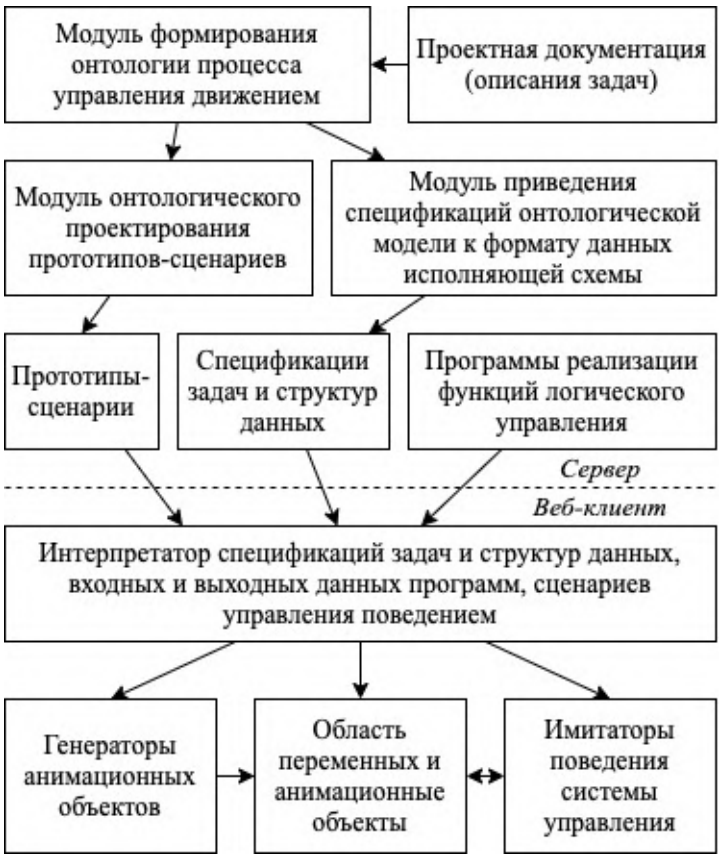


Рисунок 4.7 - Структурно-функциональная организация

4.2.1 Структура онтологической модели

Согласно модели проектного процесса, описанной в параграфе 2.2 и формуле (3.1), на входе мы имеем текстовые описания всех подпроцессов (задач) проекта с указанием входных и выходных параметров для каждого подпроцесса (см. Приложение 4). В данном примере онтологическая модель будет иметь только два уровня – уровень требований и уровень реализации.

Онтологическая модель требований строится на основе указанных текстовых описаний и дополняется сущностями, относящимися к уровню реализации. Понятийная часть объединенной ОМ имеет следующую структуру:

Nodes = <Subprocess, Parameter, Script, Value…>,

где ***Subprocess*** – концепты для описания подпроцессов (описываются на уровне требований); ***Parameter*** – концепты для описания входных и выходных параметров (описываются на уровне требований); ***Script*** – концепты для описания прототипов-сценариев (описываются на уровне требований); ***Value*** – концепты для описания конкретных значений входных параметров, используемые в прототипах-сценариях (относится к ОМ реализации).

Концепт каждого типа имеет собственную структуру. Так, концепт типа ***Subprocess*** обладает следующим набором свойств:

Subprocess = <head, legend, numTask>,

где ***head*** – название подпроцесса; ***legend*** – развернутое описание подпроцесса (по сути часть проектной документации); ***numTask*** – порядковый номер подпроцесса.

Концепты типа ***Parameter*** имеют иную структуру:

Parameter = <comment, n, notMofify, minValue, maxValue…>,

где ***comment*** – имя концепта на естественном языке; ***n*** – имя переменной, используемое для обозначения данного концепта в исполняющей программе; ***notMofify*** – принимает значение True/False и свидетельствует о том, является ли концепт конфигурационным параметром (т.е. константой, в терминах исполняющей программы).

Свойства ***minValue*** и ***maxValue*** обозначают минимальное и максимальное значения, которое может принимать параметр в соответствии с требованиями проекта.

Концепты типа ***Script*** содержат поля:

Script = <comment, numScript>,

где ***comment*** – имя концепта на естественном языке; ***numScript*** – порядковый номер сценария.

И, наконец, концепты типа ***Value*** содержат только одно поле – с указанием значения.

В онтологической модели предусмотрены отношения следующих типов:

Relations = <partwhole, structInput, structOutput, SemRel…>,

где ***partwhole*** – партитивные отношения; ***structInput*** – отношение «быть входным параметром»; ***structOutput*** – отношение «быть выходным параметром»; ***SemRel*** – группа отношений семантических типов, в которую входят такие отношения, как «иметь значение», «реализация» и др.

На рис. 4.8. представлен фрагмент онтологической модели, относящийся к конфигурации подпроцесса №1 («Время равномерного движения по прямой»):

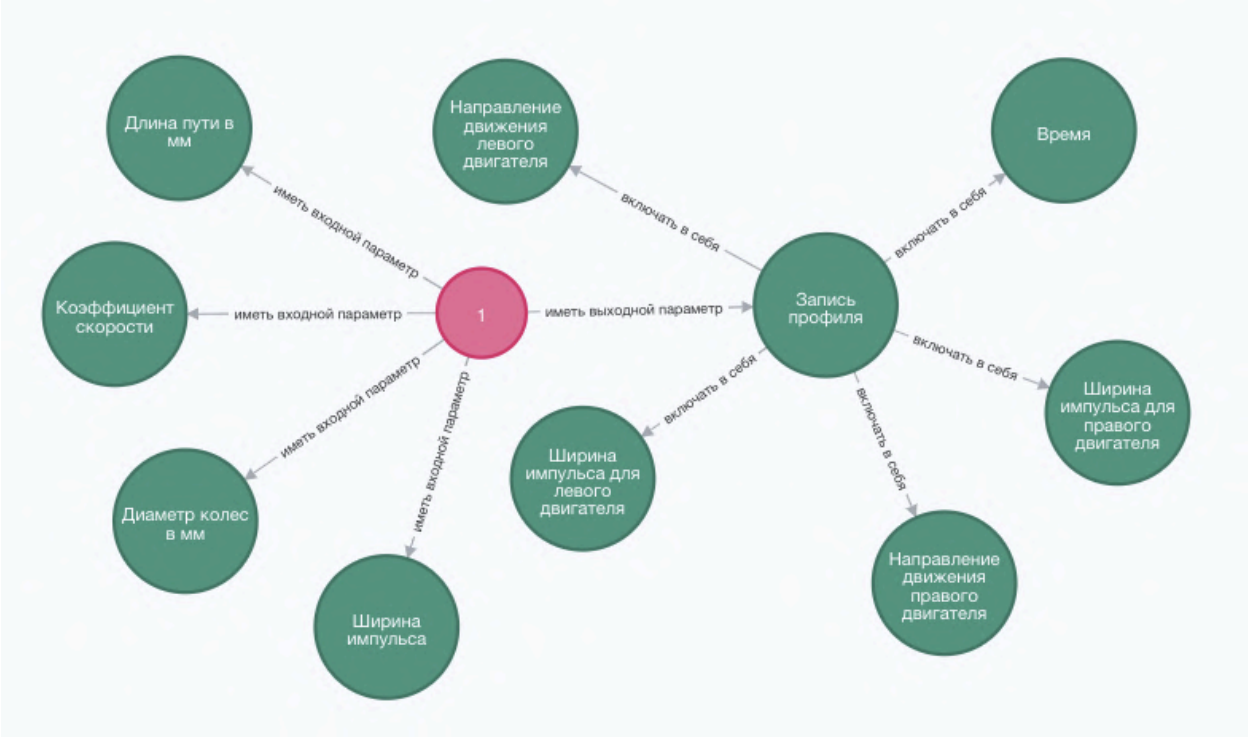


Рисунок 4.8 – Конфигурация подпроцесса №1

После создания для данного подпроцесса прототипа-сценария (“Пример 1”), ее конфигурация в онтологической модели будет выглядеть следующим образом (см. рис. 4.9).

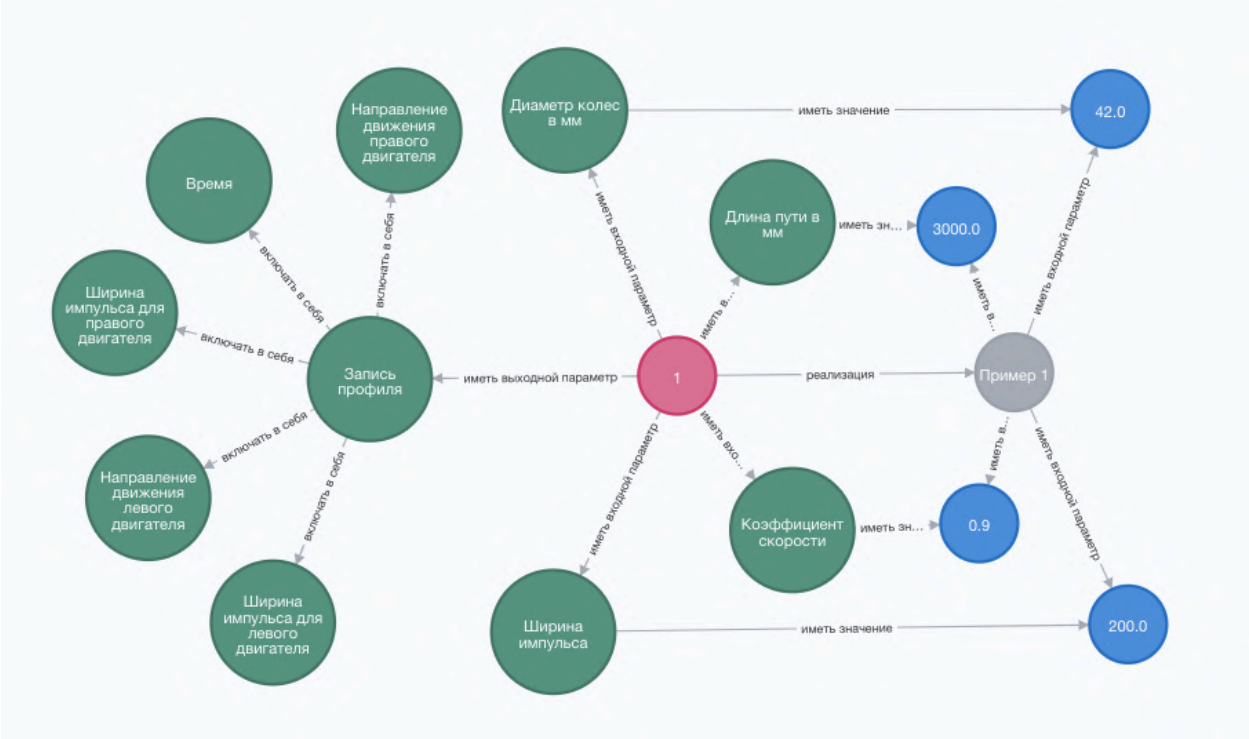


Рисунок 4.9 – Конфигурация задачи 1 и ее прототипа-сценария

4.2.1 Архитектура средств онтологической поддержки прототипирования

Рассмотрим более детально архитектуру той части средств, которая обеспечивает онтологическое сопровождение инструментов прототипирования.

Модуль формирования онтологии процесса управления движением транспортного робота принимает на вход информацию из проектной документации, прошедшей предварительную обработку, а именно 4 таблицы (могут сформированы в любом табличном редакторе; в нашем случае был использован LibreOffice):

- таблицу с html-размеченным описанием каждой задачи и указанием ее порядкового номера (tasks.csv, см. рис. 4.10);
- таблицу концептов, представляющих собой входные и выходные параметры задачи, а также связанные с ними концепты (concepts.csv, см. рис. 4.11);

- таблицу с указанием типа существующих отношений между концептами (relations.csv, см. рис. 4.12);

numTask	head	legend
0	Замысел проекта "Управление транспортным роботом"	<p>Руководители школы робототехники обнаружили у своих учен
1	Время равномерного движения по прямой	<p> В рассматриваемом идеализированном робокаре: име
2	В нужном месте в нужное время	<p> В рассматриваемом идеализированном робокаре: име
3	Определение времени на основе калибровки	<p> В рассматриваемом идеализированном робокаре: име
4	Вперед-разгрузка-возврат	<p> В рассматриваемом идеализированном робокаре: име
5	Движение под углом	<p> В рассматриваемом идеализированном робокаре: име
6	Трапецеидальный профиль	<p> В рассматриваемом идеализированном робокаре: име
7	Разворот одним колесом	<p> В рассматриваемом идеализированном робокаре: име
8	Самый крутой разворот	<p> В рассматриваемом идеализированном робокаре: име
9	Движение по прямой с энкодерами	<p> В рассматриваемом робокаре: имеется два двигателя
10	Коррекция погрешности инерции робокара	<p> В рассматриваемом робокаре: имеется два двигателя

Рисунок 4.10 - Таблица с описанием задач (фрагмент)

comment	n	type	notModify	numTask
Длина пути в мм	mmS	structInput		1,2,3,4,6,9,10
Диаметр колес в мм	mmD	structInput	TRUE	1,2,3,4,5,6,7,9,10
Коэффициент скорости	kSpeed	structInput		1,2,4,5,6,7,8
Ширина импульса	W	structInput		1,3,4,5,7,8,9,10
Время	T			
Направление движения левого двигателя	CL			
Ширина импульса для левого двигателя	WL			
Направление движения правого двигателя	CR			
Ширина импульса для правого двигателя	WR			
Время в мсек	msT	structInput		2
Время в процессе калибровки	Tc	structInput		3
Расстояние в процессе калибровки	Sc	structInput		3
Время разгрузки	Tp	structInput		4
Коэффициент повышения скорости после разгрузки	kLoadUnload	structInput		4
Координата X исходной точки	mmXs	structInput		5
Координата Y исходной точки	mmYs	structInput		5
Координата X финишной точки	mmXf	structInput		5
Координата Y финишной точки	mmYf	structInput		5

Рисунок 4.11 - Таблица с описанием концептов (фрагмент)

c1	rel	c2
Запись профиля	включать в себя	Время
Запись профиля	включать в себя	Направление движения левого двигателя
Запись профиля	включать в себя	Ширина импульса для левого двигателя
Запись профиля	включать в себя	Направление движения правого двигателя
Запись профиля	включать в себя	Ширина импульса для правого двигателя
Пара чисел энкодеров	включать в себя	Счетчик энкодера левого колеса
Пара чисел энкодеров	включать в себя	Счетчик энкодера правого колеса

Рисунок 4.12 - Таблица с описанием отношений (фрагмент)

На выходе работы данного модуля – онтологическая модель, которая хранится в графовой базе данных, сформированной на базе СУБД Neo4j. Для

подключения к базе данных предусмотрено два модуля – ***модуль настроек доступа к онтологической БД*** и ***модуль установления связи с онтологической БД***.

Модуль приведения спецификаций онтологической модели к формату данных исполняющей системы принимает на вход данные из онтологии процессов управления движением транспортного робота и формирует спецификации задач и структур данных в JSON-формате.

Модуль онтологического проектирования прототипов-сценариев также использует онтологические данные для формирования прототипов-сценариев в JSON-формате.

Все основные модули подключены к ***модулю визуализации средств онтологической поддержки прототипирования***, который формирует веб-интерфейс (см. рис. 4.13) приложения с помощью фреймворка Dash для Python.

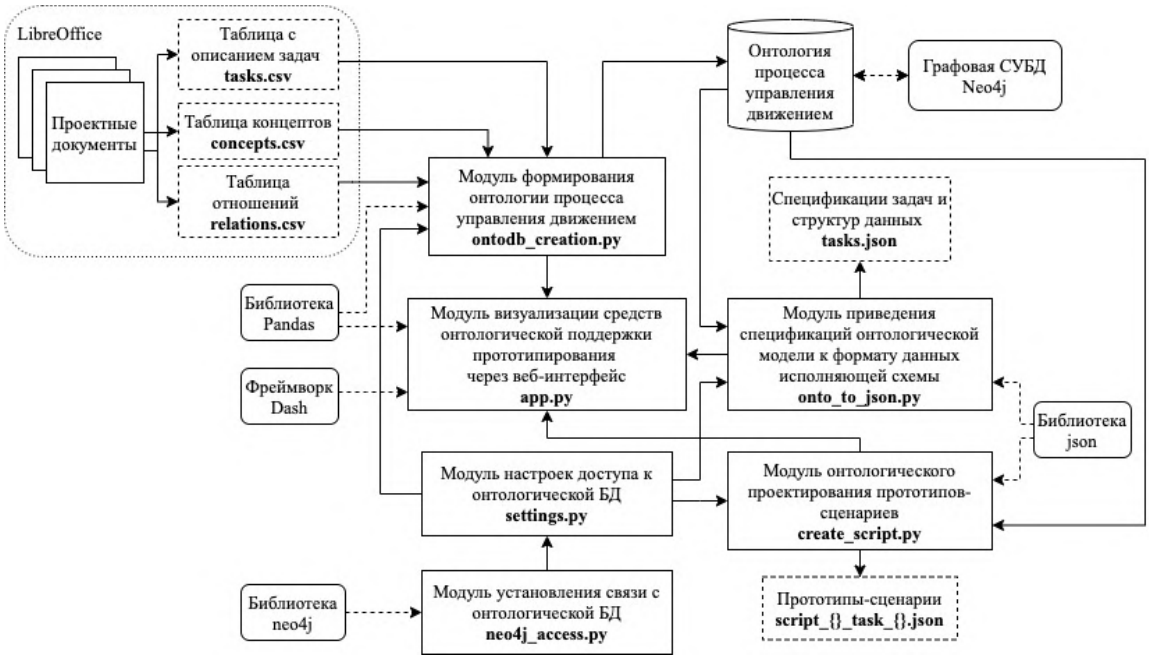


Рисунок 4.13 - Архитектура средств онтологического прототипирования

Интерфейс (см. рис. 4.14) позволяет построить онтологическую модель на основе описанных выше таблиц, генерировать task.json (формат данных

исполняющей программы), а также создавать для указанных подпроцессов новые прототипы-сценарии с заданными входными параметрами. Данные файлы, в свою очередь, используются программами реализации функций логического управления для демонстрации работы транспортного робота согласно различным сценариям.

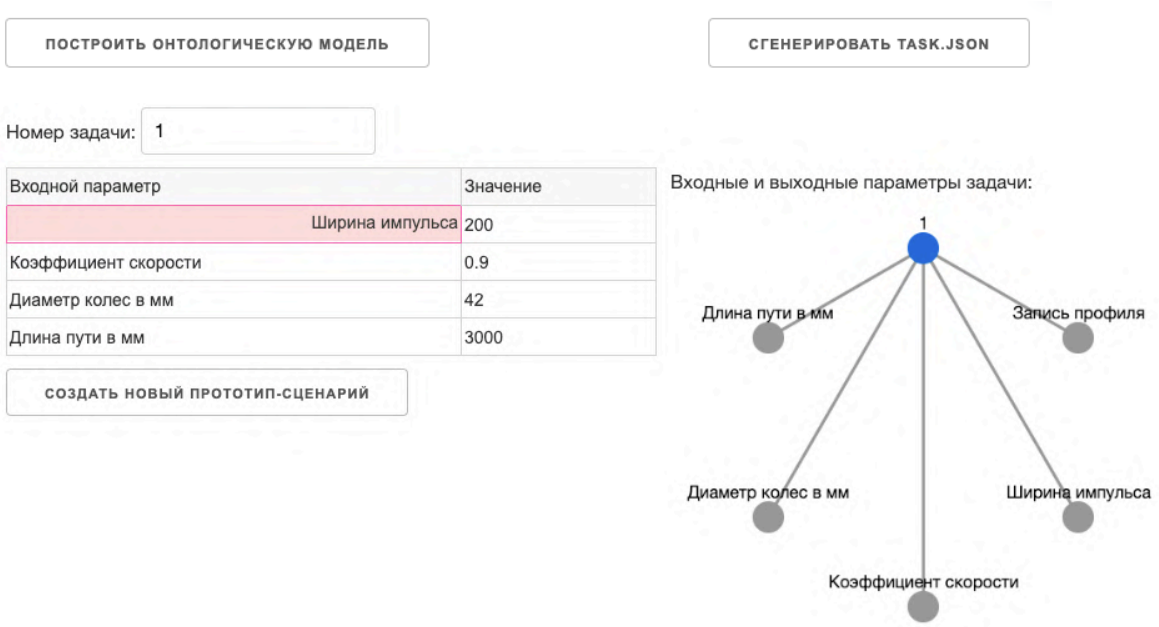


Рисунок 4.14 - Интерфейс средств онтологического прототипирования

Таким образом, онтологическая модель проекта в представленном представляет собой такую формализацию концептуальных требований к проекту, которые ориентированы на реализацию средств прототипирования.

На рис. 4.15 показаны диаграммы деятельности проектировщика: слева – без применения средств онтологической поддержки прототипирования; справа – с их применением. Как видно, использование средств интерпретации онтологических спецификаций на этапе прототипирования, во-первых, повышает гибкость процесса уточнения требований, во-вторых снижает трудоемкость прототипирования и, тем самым, уменьшает следующие риски:

- неодинаковое толкование требований заказчиком и исполнителем;
- ошибка недооценки трудоемкости, приводящая к срывам сроков и нарушениям бюджета.

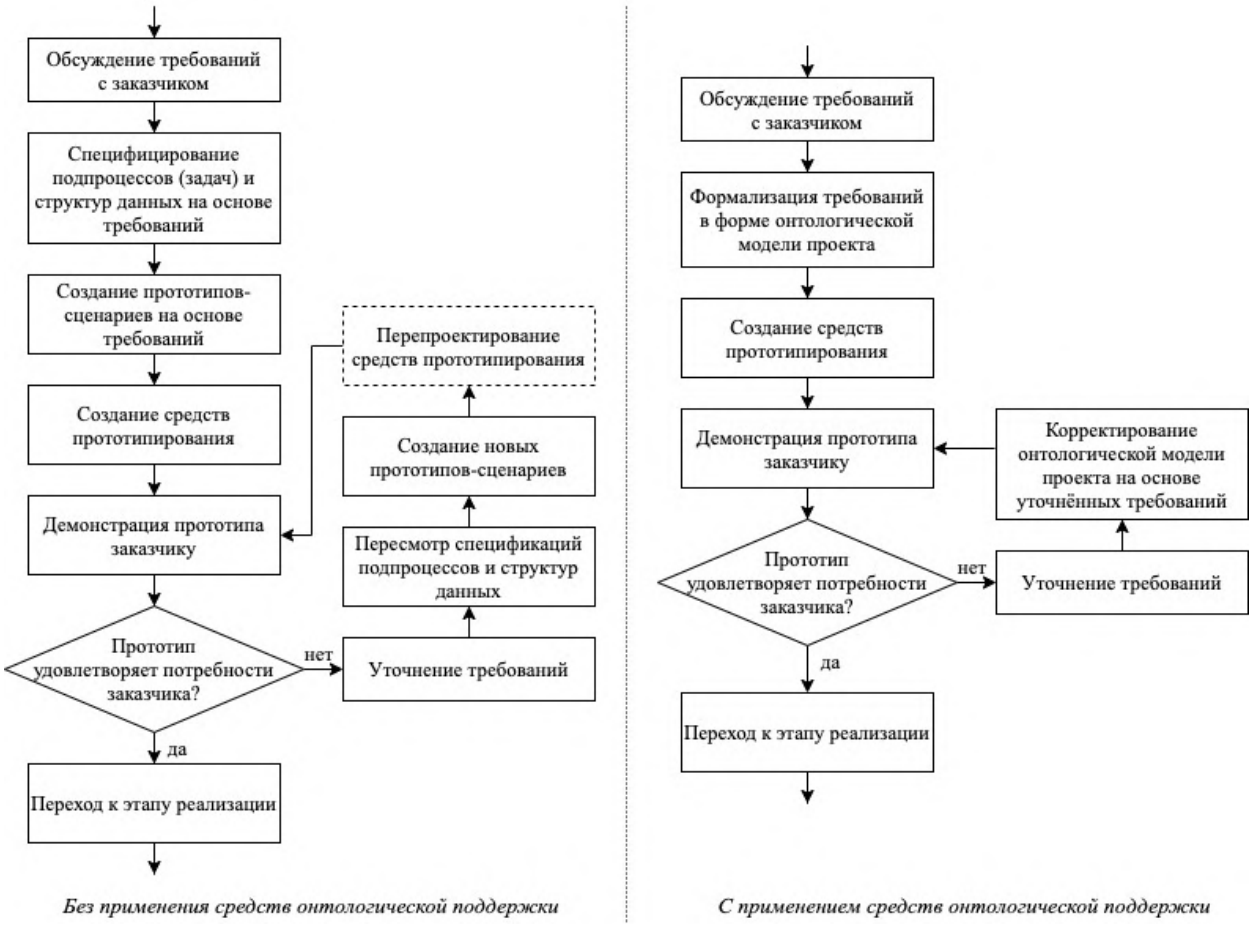


Рисунок 4.15 – Сравнение диаграмм деятельности проектировщика без и с применением средств онтологической поддержки

4.3 Разработка управляющей продвижением товаров народного потребления подсистемы АС с использованием средств онтологического моделирования

Эксперимент связан с разработкой одной из подсистем АС, обслуживающей процесс управления производством товаров народного потребления, а также продвижения данных товаров с помощью электронной торговой системы, на основе пространства прототипов с использованием средств онтологической поддержки. Функциональным назначением разрабатываемой подсистемы является организация хранения данных о

производителях товаров народного потребления, собственно товарах, заказчиках, а также данных, обслуживающих процессы их взаимодействия.

4.3.1 Специфицирование исходных прототипов и формирование ОМ исходных проектных решений

Для реализации данной подсистемы возможно использование прототипа, а именно – фрагмента базы данных, обслуживающей развлекательный сервис с аудио- и текстовым контентом, реализованный в виде мобильного приложения. Данный фрагмент возможно использовать в качестве прототипа, поскольку многие его сущности связаны с сущностями результирующей системы отношением аналогии.

Рассматриваемый фрагмент БД содержит 14 таблиц (описание таблиц представлено в таблице 4.2) и 136 полей, которые формируют пространство имен ***NS*** исходных прототипов ***IPt***:

$$\boldsymbol{IPtNS = (IDBTN, IDBFN)}, \quad (4.2)$$

где ***IDBTN*** *(Input Database Table Names)* – набор имен таблиц исходной базы данных;

IDBFN *(Input Database Field Names)* – набор имен полей исходной базы данных.

Таблица 4.2 - Структура БД развлекательного сервиса

Название таблицы	Краткое описание
_Installation	Содержит информацию об уникальных установках мобильного приложения.
_Session	Содержит информацию об уникальных пользовательских сессиях в мобильном приложении (промежуток между

	открытием приложения и его закрытием или сворачиванием).
_User	Содержит информацию об уникальных пользователях мобильного приложения.
Blacklist	Содержит информацию об авторах историй, отправленных в черный список, которым запрещено присылать новые истории для публикации в приложении.
Category	Содержит информацию о категориях (чаще всего жанрах) истории в приложении.
CoverVariant	Содержит информацию о вариантах обложки историй, используемых для АБ-тестирования.
Episode	Содержит информацию об эпизодах (главах) истории.
InAppPurchase	Содержит информацию об in-app покупках (покупках внутри приложения).
Product	Содержит информацию о различных продуктах, которые можно купить внутри приложения.
Purchase	Также содержит информацию о покупках.
Receipt	Содержит информацию о чеках.
Story	Содержит информацию об историях, опубликованных в приложении.
StoryRegistry	Содержит информацию об историях, отправленных на рассмотрение авторами для возможной последующей публикации (приеме историй).
StoryStats	Содержит информацию об аналитических показателях каждой истории, опубликованной в приложении.

StoryStatusHistory	История статусов проверки историй, отправленных на рассмотрение авторами для возможной последующей публикации.
TargetDatasource	Содержит информацию об источниках таргетированной рекламы.

С помощью соответствующих SQL-запросов была получена информация о наборе сущностей и потенциальных связей между ними. Далее была сформирована онтологическую модель уровня исходных проектных решений ***IO***. В данной ОМ концепты обладают следующими свойствами

- name (имя концепта на естественном языке);
- materialization_name (имена, которые имеют отношение к концепту, встречающиеся в различных материализациях: в нашем случае это названия таблиц и полей);
- materialization_type (тип материализации: в нашем случае это Database Table – таблица БД, Database Field – поле таблицы БД, Database Field Value – значение поля таблицы БД).

На рис. 4.16 показан фрагмент онтологической модели (концепты и семантические отношения между ними без указаний свойств данных объектов).

Данная модель содержит 47 понятий, 48 свойств-связей между понятиями, 88 свойств-атрибутов понятий, 3 агрегата, а также аксиомы (правила вывода).

Агрегаты разбивают онтологическую модель по сферам применения, а именно: Agg A – привлечение пользователей и продажи; Agg B – содержательная часть системы (контент, как продукт потребления); Agg C – организация поставок контента. Как видно из описания данных агрегатов, они легко соотносятся с результирующей подсистемой АС с той разницей, что

продуктом потребления будет являться товар, а организация поставок контента будет замене на организацию поставок товаров.

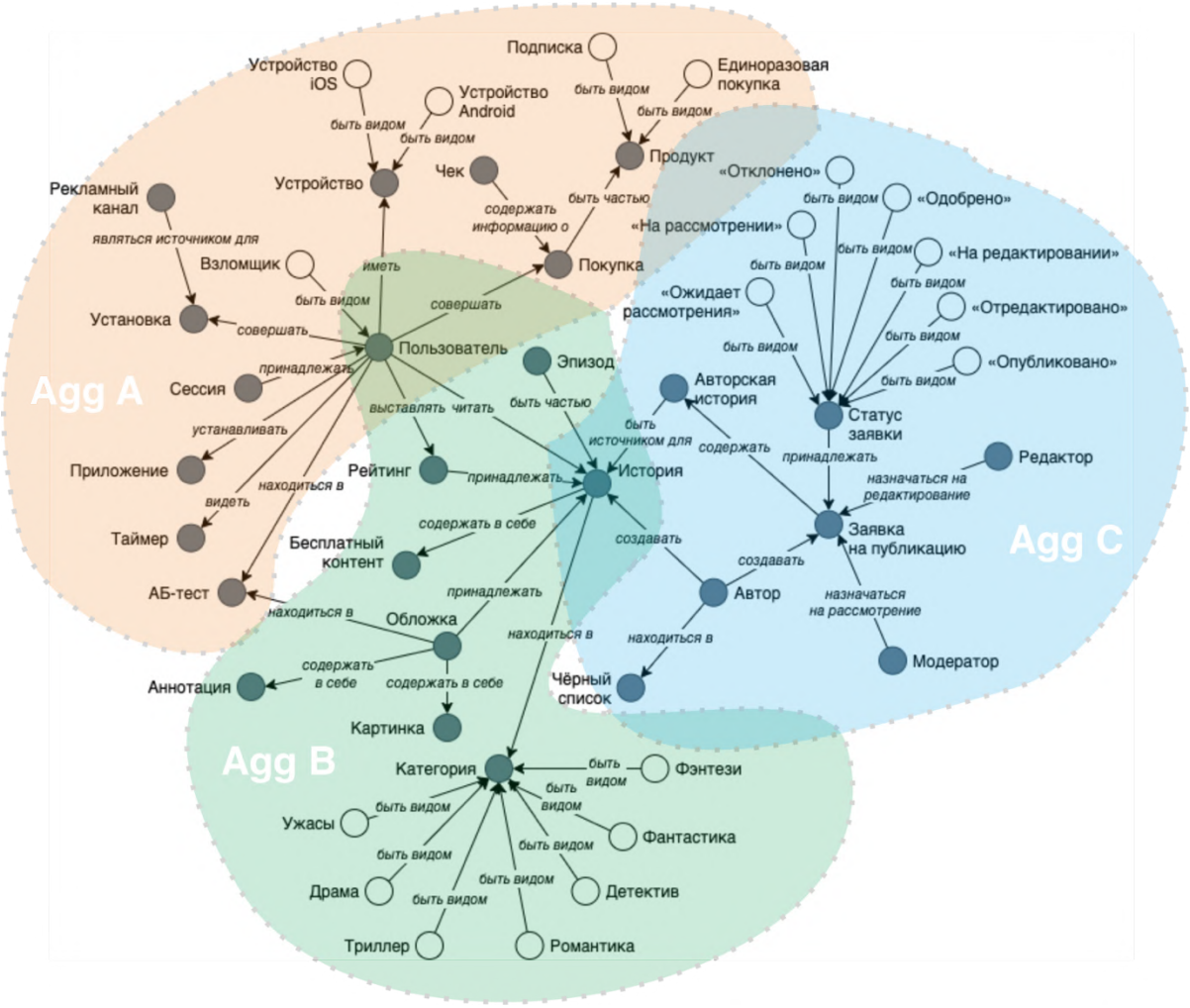


Рисунок 4.16 - Онтологическая модель исходных проектных решений

Примеры правил, применимых к данной онтологической модели:

- ЕСЛИ автор создает заявку, ТО на ее рассмотрение назначается модератор:

  Автор (?a) ∧ Заявка на публикацию (?b) ∧ создавать (?a, ?b) → Модератор (?c), назначаться на рассмотрение (?c, ?b)

- ЕСЛИ заявка на публикацию имеет статус «Одобрено», ТО она назначается на редактирование редактору:

«Одобрено» (?a) ∧ Заявка на публикацию (?b) ∧ принадлежать (?a, ?b) → Редактор (?c), назначаться на редактирование (?c, ?b)

- ЕСЛИ заявка на публикацию имеет статус «Отредактировано», ТО авторская история, содержащаяся в ней становится источником для публикации истории:

  «Отредактировано» (?a) ∧ Заявка на публикацию (?b) ∧ принадлежать (?a, ?b) ∧ Авторская история (?c) ∧ содержать (?b, ?c) → История (?d), быть источником для (?c, ?d)

Понятия и их свойства-атрибуты формирует пространство имен онтологической модели исходных проектных решений ***IONS*** над пространством имен ***IPtNS***:

$$IONS = (IOCN, IOAN), \quad (4.3)$$

где ***IOCN*** *(Input Ontology Concept Names)* – имена концептов исходной онтологической модели;

IOAN *(Input Ontology Attribute Names)* – имена атрибутов концептов исходной онтологической модели.

4.3.2 Трансформирование ОМ исходных проектных решений формирование ОМ целевой подсистемы

Трансформирование онтологической модели происходит в несколько этапов:

1) исключение из исходной модели тех понятий, которые не могут быть материализованы в целевых проектных решениях;
2) модификация имен агрегатов, концептов, атрибутов и связей – там, где меняется их семантическая нагрузка;
3) добавление в модель новых сущностей.

Построим связи между онтологическими моделями:

- modified (связывает модифицированные элементы онтологии целевых проектных решений с элементами ОМ исходных ПР – по сути является отношением аналогии);
- reused (связывает заимствованные элементы онтологии целевых проектных решений с элементами ОМ исходных ПР).

В дальнейшем данная информация может быть использована для расчета различных метрик, демонстрирующих эффективность применения онтологического моделирования.

Обозначим синим концепты и отношения, которые необходимо исключить из исходной онтологической модели исходных ПР:

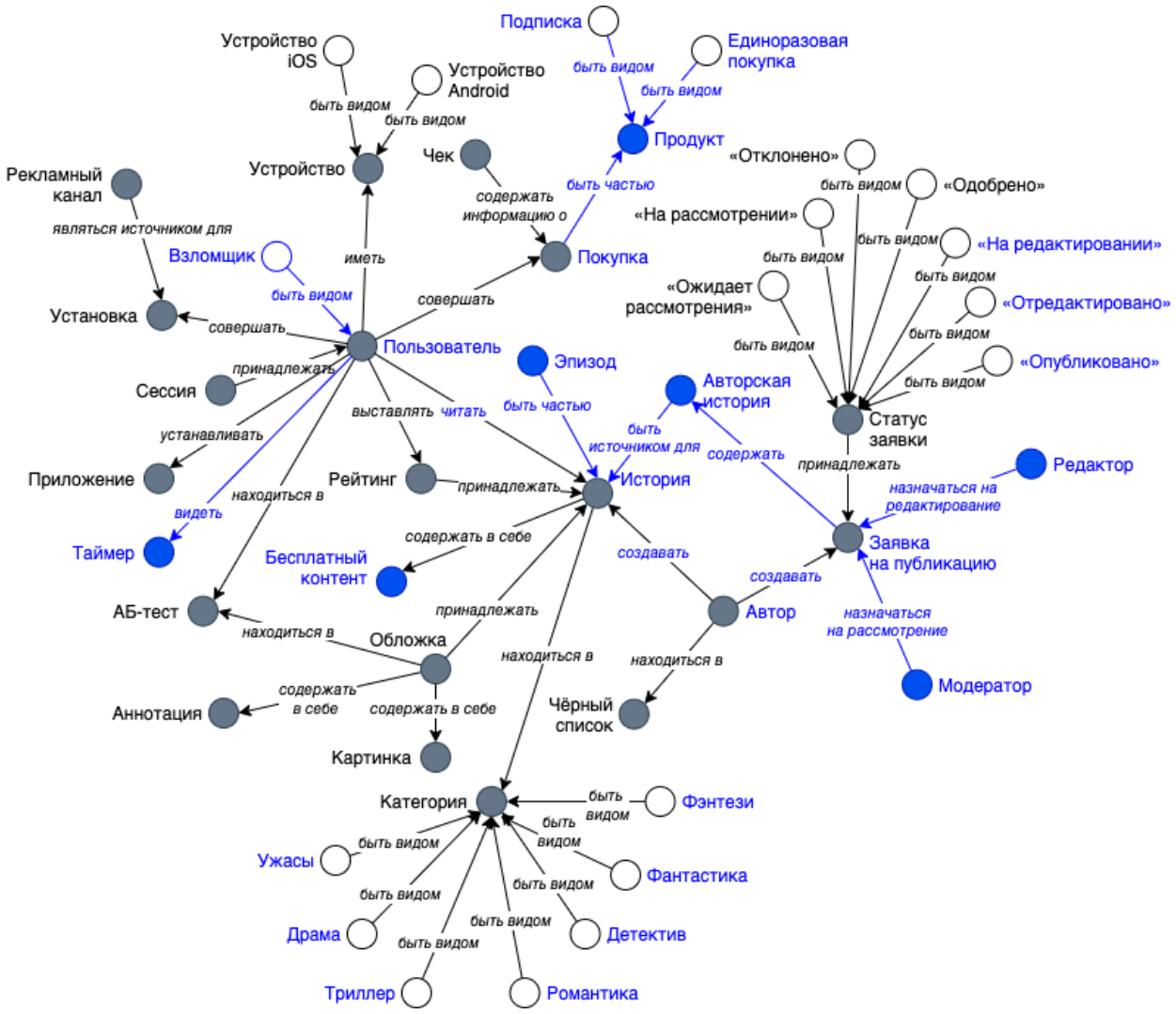


Рисунок 4.17 - Трансформирование онтологической модели. Фаза 1

Обозначим красным концепты и отношения, которые необходимо включить в новую онтологическую модель целевых ПР:

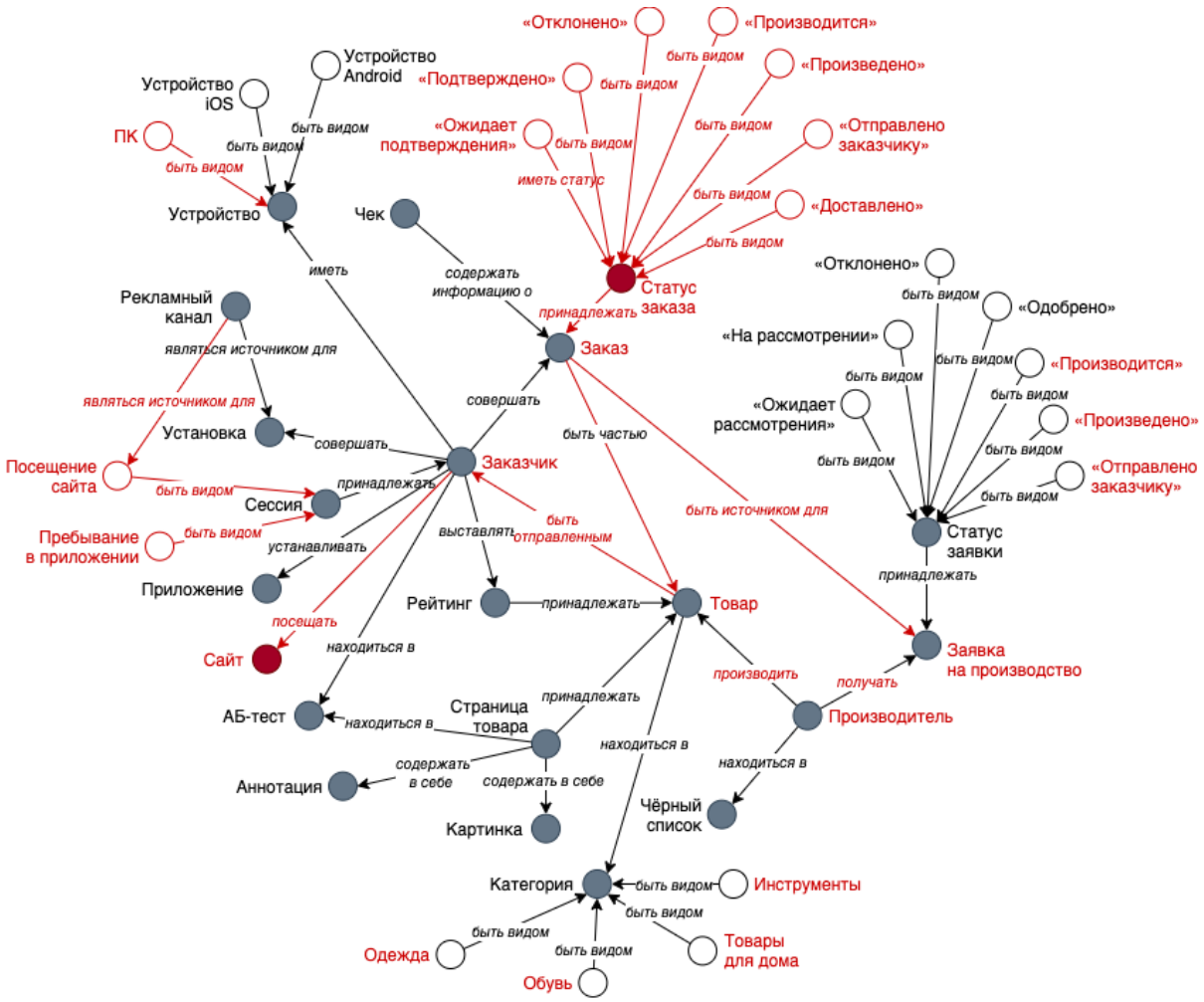


Рисунок 4.18 – Трансформирование онтологической модели. Фаза 2

Модифицированные и новые объекты онтологии формируют пространство имен онтологической модели целевых проектных решений ***ONS***:

$$\boldsymbol{ONS = (NOCN, NOAN),} \qquad (4.4)$$

где ***NOCN*** *(New Ontology Concept Names)* – имена концептов новой онтологической модели; ***NOAN*** *(Input Ontology Attribute Names)* – имена атрибутов концептов новой онтологической модели.

Онтологическая модель целевых проектных решений может служить основой для формирования структуры БД подсистемы АС, обслуживающей процесс управления производством товаров народного потребления и

продвижения данных товаров с помощью электронной торговой системы. Новая БД содержит 5 таблиц и 117 полей, большая часть из которых наследуется из исходной. Новые же единицы БД формируют пространство имен целевых проектных решений ***PANS***:

$$PANS = (NDBTN, NDBFN), \quad (4.5)$$

где ***NDBTN*** *(New Database Table Names)* – набор имен таблиц целевой базы данных; ***NDBFN*** *(New Database Field Names)* – набор имен полей целевой базы данных.

4.3.3 Оценка сокращения семантического разрыва

Статистика по количеству единиц в каждом подпространстве имен на различных этапах проектирования показана на рис 4.19.

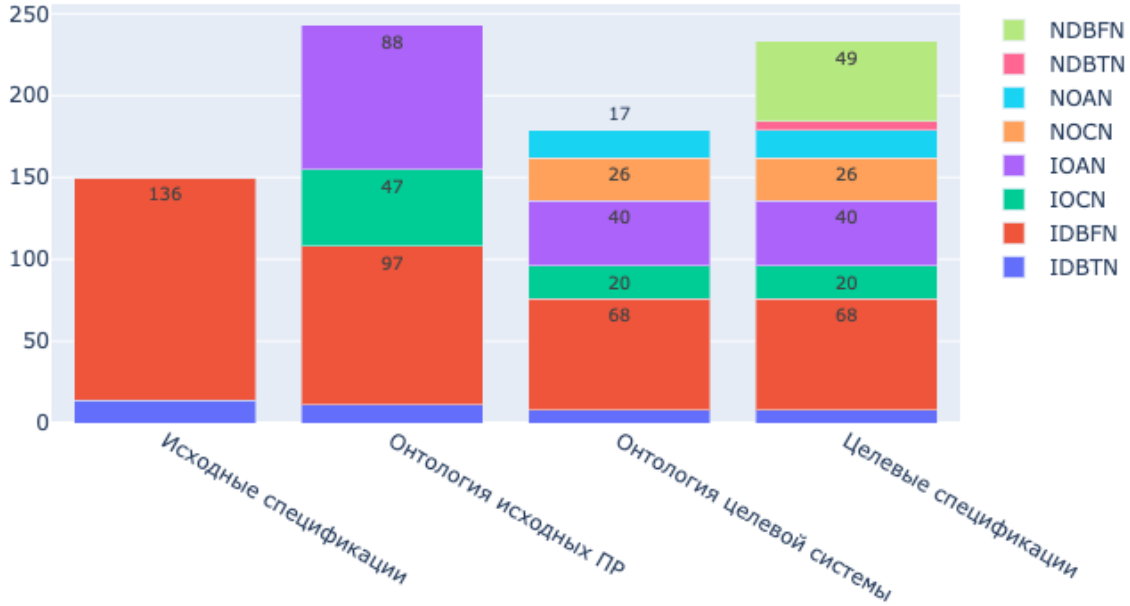


Рисунок 4.19 - Пересечение пространств имен на различных фазах проектирования

Поскольку одной из целей нашего исследования является повышение степени концептуального единства проектных решений и снижение трудозатрат на когнитивные усилия для осознания того, что различные имена

обозначают одно понятия, то логично использовать в качестве метрики оценки эффективности ***меру пересечения пространств имен*** на различных фазах проектирования, которую предлагается рассчитывать с помощью коэффициента близости Жаккара, который широко используется при оценке семантической близости лингвистических данных в тех случаях, когда отсутствует необходимость принимать в расчет такие характеристики лингвистических единиц, как, например, частотность:

$$K_J = \frac{n(A \cap B)}{n(A)+n(B)-n(A \cap B)}, \quad (4.6)$$

Как видно на рис. 4.18, коэффициенты близости пространств имен, полученных в результате их обобщения, трансформации и конкретизации, выполненных с помощью онтологий, в среднем ***в 3 раза превосходят*** коэффициент близости пространств имен исходных и целевых проектных решений ***IPtNS*** и ***PANS***. А, поскольку величина семантического разрыва между представлением информации на различных стадиях проектирования коррелирует с трудозатратами, то данные показатели, полученные в результате эксперимента, указывают на эффективность включения в процесс проектирования онтологического сопровождения.

Рисунок 4.20 - Коэффициенты близости

Выводы по главе 4

В данной главе были представлены результаты ряда экспериментов, подтверждающие работоспособность и эффективность предлагаемых методов онтологического сопровождения проектирования и разработанных инструментальных средств.

Эксперимент с проектированием СЛУ роботами-плиткоукладчиками показал, что, при возникновении необходимости в доработках вследствие обнаружения неточностей или изменения требований заказчика, внедрение в проектный процесс ОМ существенно снижает затраты, поскольку обеспечивает повышение автоматизации проектного процесса как на стадии концептуального проектирования (генерирования диаграммы состояний), так и на стадии реализации (частичная генерация кода).

Использование средств интерпретации онтологических спецификаций системой быстрого прототипирования базовых средств программного управления транспортными роботами, во-первых, показало повышение гибкости процесса уточнения требований, во-вторых, снижение трудоемкости прототипирования и, тем самым, уменьшение рисков неодинакового толкования требований заказчиком и исполнителем, а также риски недооценки трудоемкости, приводящие к срывам сроков и нарушениям бюджета.

Эксперимент по разработке управляющей продвижением товаров народного потребления подсистемы АС показал, что средства онтологического моделирования позволяют уменьшить величину семантического разрыва между представлением информации на различных стадиях проектирования. А, поскольку величина семантического разрыва коррелирует с трудозатратами, то показатели, полученные в результате эксперимента, указывают на эффективность включения в процесс проектирования онтологического сопровождения.

Заключение

В результате выполнения работы получены следующие практические и научные результаты:

1. Проведено исследование подходов к онтологическому моделированию проектного процесса, а также анализ существующих проблем в области проектирования АС, связанных с семантическими разрывами.

2. Разработан подход к онтологическому сопровождению проектного процесса разработки АС в условиях оперативного взаимодействия проектировщиков с доступным опытом, обеспечивающий доминирование сущностей объекта автоматизации на всех его стадиях, вплоть до реализации.

3. Разработана модель проектного процесса разработки АС, базирующаяся на принципах DT-подхода с применением онтологического моделирования, направленная на обеспечение концептуальной целостности, непротиворечивости и полноты разрабатываемых проектных решений.

4. Разработаны аналитические модели системы связанных онтологий требований, проектных решений и реализации, включающие в себя набор ограничений и функций интерпретации, обеспечивающих возможность автоматизированной трансформации ОМ, а также генерирования исходных кодов программ.

5. Разработаны средства, обслуживающие онтологическую поддержку проектного процесса и повышающие производительность труда в части обработки исходных спецификаций, которые служат основой для формирования проектных онтологий.

6. Разработаны средства, обслуживающие формирование UML-диаграмм и исходных кодов программ автоматизации на основе ОМ проекта.

7. Проведена апробация разработанного подхода и инструментальных средств в ходе решения трех задач по автоматизации. Внедрение подхода и инструментальных средств позволило сократить трудозатраты на доработку системы в условиях изменения требований

(сокращение составило 30% при внесении 50 изменений в требования к логике функционирования системы) и трудозатраты непосредственно на онтологическое моделирование на 56%, а также снизить информационный разрыв между этапами проектного процесса в 3 раза.

Разработанная технология является универсальной и может быть применена для автоматизации проектного процесса разработки различных сложных систем с ПО за счет предложенной структуры метаданных онтологии проекта, которая организована таким образом, что предусматривает возможность изменения набора онтологических моделей и их внутренней структуры в зависимости от поставленных задач. Возможности разработанных инструментальных средств могут быть расширены и адаптированы для их применения в различных предметных областях.

Список сокращений и условных обозначений

CALS-технологии (англ. Continuous Acquisition and Life cycle Support) – непрерывная информационная поддержка поставок и жизненного цикла изделий

DT – Design Thinking

LPG – Labeled Property Graph

OBSE-ODSE (Ontology-Based (-Driven) Software Engineering) – программная инженерия под управлением онтологий

UML – Unified Modeling Language

W3C (World Wide Web Consortium) – Консорциум Всемирной паутины

XML – eXtensible Markup Language

АС – автоматизированная система

АП – автоматизированное проектирование

БД – база данных

ЕЯ – естественный язык

ЖЦ – жизненный цикл

ИПИ – информационная поддержка процессов жизненного цикла изделий

ООП – объектно-ориентированное программирование

ОМ – онтологическая модель

ОП – онтология проекта

ПО – программное обеспечение

ПР – проектное решение

ПрО – предметная область

САПР – система автоматизированного проектирования

СКФ – социо-киберфизическая реальность

ЯП – язык программирования

Список литературы

1. Массель, Л.В., Ворожцова, Т.Н., Пяткова, Н.И. Онтологический инжиниринг для поддержки принятия стратегических решений в энергетике / Л.В. Массель // Онтология проектирования. – 2017. – N 1(7). – С. 66 -76.

2. Игруша, В.А., Сосинская, С.С. Формализация описания технологических процессов изготовления деталей машиностроения на основе онтологии и объектно-ориентированных баз данных / В.А. Игруша // Онтология проектирования. – 2017. – N 1(7). – С. 77 -88.

3. Golenkov, V.V., Taberko, V.V., Ivanyuk, D.S. et. al. Designing batch-manufacturing enterprises using ontologies / V.V. Golenkov // Онтология проектирования. – 2017. – N 2(7). – P. 123-144.

4. Павлов, С.В., Ефремова, О.А. Онтологическая модель интеграции разнородных по структуре и тематике пространственных баз данных в единую региональную базу данных / С.В. Павлов, О.А. Ефремова // Онтология проектирования. – 2017. – N 3(7). – С. 323 -333.

5. Микони, С.В. О качестве онтологических моделей / С.В. Микони // Онтология проектирования. – 2017. – N 3(7). – С. 347 -360.

6. Кучуганов, В.Н. Онтология и анимация прецедентов / В.Н. Кучуганов // Онтология проектирования. – 2016. – N 3(6). – С. 287 -296.

7. Олейник, А.Г., Ломов, П.А. Разработка онтологии интегрированного пространства знаний / А.Г. Олейник // Онтология проектирования. – 2016. – N 4(6). – С. 456-474.

8. Gavrilova, T., Leshcheva, I., Bolotnikova, E., Blagov, E., Yanson, A. Measuring Psychological Impact on Group Ontology Design and Development: Empirical Approach / T. Gavrilova // Book series “Communications in Computer and Information Science” / P. Klinov, D. Mouromtsev. – Springer, 2013. – P. 29-43, 2013.

9. Gavrilova, T., Gladkova, M. Big Data Structuring: The Role of Visual Models and Ontologies / T. Gavrilova // Procedia Computer Science. – Elseiver, 2014. – T. 31. – P. 336-343.

10. Zagorulko, Y.A., Borovikova, O., Zagorulko, G. Methodology for the development of ontologies for thematic intelligent scientific internet resources / Y.A. Zagorulko // Proceedings of the Second Russian-Pacific Conference on Computer Technology and Applications (RPC). – 2017. – P. 194.

11. Zagorulko, Y.A., Zagorulko, G. Technology for Building Subject-Based Intelligent Scientific Internet Resources Based on Ontology / Y.A. Zagorulko // International Conference on Intelligent Software Methodologies, Tools, and Techniques. – 2016. – P. 51-60.

12. Хорошевский, В.Ф. Проектирование систем программного обеспечения под управлением онтологий: модели, методы, реализации / В.Ф. Хорошевский // Онтология проектирования. – 2019. – N 4 (34). – С. 429-445.

13. Hein, Andreas M. Identification and Bridging of Semantic Gaps in the Context of Multi-Domain Engineering / Andreas M. Hein // Proceedings of 2010 Forum on Philosophy, Engineering & Technology. – 2010. – P.57-58.

14. Minagawaa, M., Kusayanagi Sh. Study on BIM Utilization for Design Improvement of Infrastructure Project / M. Minagawaa // Proceedings of the 5th International Conference of Euro Asia Civil Engineering Forum (EACEF-5). Procedia Engineering. – 2015. – P. 431-437.

15. Sikos, Leslie F. The Semantic Gap / Leslie F. Sikos // Description Logics in Multimedia Reasoning / Leslie F. Sikos. – Springer, 2017. – P. 51-66.

16. Regev, S., Shtub, A., Ben-Haim, Y. Managing Project Risks as Knowledge Gaps / S. Regev // Project Management Journal. – 2006. – N 37. – P. 17-25.

17. Liu, Y. Critical Factors for Managing Project Team Communication at the construction stage: PhD Thesis / Y. Liu. – Hong Kong, 2009.

18. Xie, X, Thorpe, A., Baldwin, A. A survey of communication issues in construction design / X. Xie // Proceedings of the 16th Annual ARCOM Conference. – 2000. – Vol. 2. – P. 771-80.

19. Biscaya, S., Tah, J.H.M., A Literature Review on Information Coordination in Construction / S. Biscaya // Proceedings of the Seventh International Postgraduate Research Conference in the Built Environment. – 2007. – P. 192-201.

20. Winch, G., Usmani, A., Edkins, A. Towards total project quality: A gap analysis approach / G. Winch // Construction Management & Economics. 1998. – N 16. – P. 193-207.

21. Razavian, M., Tang A., Capilla R., Lago P. Reflective Approach for Software Design Decision Making / M. Razavian // Proc. of the first symposium “Qualitative Reasoning about Software Architectures”. – 2016. – P. 19-26.

22. Häger F., Kowark T., Krüger J., Vetterli Ch., Übernickel F., Uflacker M. DT@Scrum: Integrating Design Thinking with Software Development Processes / F. Häger // Design Thinking. Understanding Innovation. – 2014. – P. 263-289.

23. Brown, T. Change by Design: How Design Thinking Transforms Organizations and Inspires Innovation / T. Brown. – USA : HarperBusiness, 2009. – 272 P.

24. Leifer, L, Meinel, C. Manifesto: Design thinking becomes foundational / L. Leifer // Design Thinking Research: Making Design Thinking Foundational. – 2015. – P. 1-4.

25. Razavian, M., Tang, A., Capilla, R., Lago, P. Reflective Approach for Software Design Decision Making / M. Razavian // Proc. of Qualitative Reasoning about Software Architectures. – 2016. – P. 19-26.

26. Соловьев, В.В. Система автоматизированного проектирования. САПР – вопросы и ответы. Учебное пособие для бакалавров инженерного факультета по направлению 55900 «Технология, оборудование и

автоматизация машиностроительных производств» / В.В. Соловьев. – М. – 2004. – 135 с.

27. Лофицкий, И.В., Матюнин, С.А. САПР автоматизации технологических процессов [Электронный ресурс] : комплекс тестовых материалов для интерактив. обучения в системе MOODLE / Минобрнауки России, Самар. гос. аэрокосм. ун-т им. С. П. Королева (нац. исслед. ун-т); авт.-сост. И. В. Лофицкий, С.А. Матюнин. - Электрон. текстовые и граф. дан. (797 Кбайт). - Самара, 2012. - 1 эл. опт. диск (CD-ROM).

28. Fuge, M, Yumer, M.E., Orbay, G., Kara L.B. Conceptual design and modification of freeform surfaces using dual shape representations in augmented reality environments / M. Fuge // Computer-Aided Design. – 2012. – N 44 (10). – P. 1-13.

29. Vuletic, T., Duffy, A., Hay, L., Mcteague, Ch., Lyall, L., Grealy, M. The challenges in computer supported conceptual engineering design / T. Vuletic // Computers in Industry. – 2018. – N 95. – P. 22-37.

30. Суханов, В.О., Кукарцев В.В.. Актуальность Применения Cals-технологий на машиностроительных предприятиях России / В.О. Суханов // Актуальные проблемы авиации и космонавтики. – 2011. – Т. 1, вып. 7. – С. 466-467

31. Shangina, E. The Introduction of CALS-Technologies in Russia / E. Shangina // Advances in Economics, Business and Management Research. – 2020. – Vol. 128. – P. 1258-1263.

32. Iskanius, P. An Agile Supply Chain for a Project-Oriented Steel Product Network: Academic Dissertation / P. Iskanius. – Linnanmaa, 2006. – 214 p.

33. Wang, Ch., Bi, Z., Xu, Li. IoT and Cloud Computing in Automation of Assembly Modeling Systems / Ch. Wang // IEEE Transactions on Industrial Informatics. – 2014. – N 10(2). – P. 1426-1434.

34. Норенков, И.П. Основы автоматизированного проектирования: Учеб. для вузов. 2-е изд., перераб. и доп. / И.П. Норенков – М.: Изд-во МГТУ им. Н. Э. Баумана, 2002. – 336 с.

35. Bjarnason, E., Wnuk, K., Regnell, B. Requirements are slipping through the gaps — A case study on causes & effects of communication gaps in large-scale software development / E. Bjarnason // Proceedings of the 2011 IEEE 19th International Requirements Engineering Conference. – 2011. – P. 37-46.

36. Gruber, T.R. The role of common ontology in achieving sharable, reusable knowledge bases / T.R. Gruber // Principles of Knowledge Representation and Reasoning. Proceedings of the Second International Conference / J.A. Allen, R. Fikes, E. Sandewell – eds. Morgan Kaufmann. – 1991. – P. 601-602.

37. Gruber, T.R. A Translation Approach to Portable Ontologies / T.R. Gruber // Knowledge Acquisition. – 1993. – N 5(2). – P. 199–220.

38. Borst, W. Construction of Engineering Ontologies: PhD thesis / W. Borst. – Enschede, 1997.

39. Studer, R., Benjamins, R., Fensel, D. Knowledge engineering: Principles and Methods / R. Studer // Data & Knowledge Engineering. – 1998. – N 25(1–2). – P. 161–198.

40. Лапшин, В.А. Онтологии в компьютерных системах / В.А. Лапшин // RSDN Magazineю – 2009. – N 4. – С. 1110.

41. Gasevic, D., Kaviani, N., Milanovic, M. Ontologies and Software Engineering / D. Gasevic // Handbook on Ontologies. – 2009. – P. 593-615.

42. Горшков, С. Введение в онтологическое моделирование [Электронный ресурс] / С. Горшков – ООО «ТриниДата», 2016. – Режим доступа: https://trinidata.ru/files/SemanticIntro.pdf.

43. Bhatia, M.P.S., Kumar A., Beniwal K.. Ontologies for Software Engineering: Past, Present and Future / / M.P.S. Bhatia, A. Kumar, R. Beniwal // Indian Journal of Science and Technology. – 2016. – 9(9). – P. 8-17.

44. Fitsilis, P., Gerogiannis, V., Anthopoulos, L. Ontologies for Software Project Management: A Review / P. Fitsilis, V. Gerogiannis, L. Anthopoulos // Journal of Software Engineering and Applications. – 2014. – 7. – P. 1096-1110.

45. Happel, H.-J., Seedorf, S. Applications of Ontologies in Software Engineering / H.-J. Happel, S. Seedorf // Proceedings of international workshop on semantic Web enabled software engineering. – 2006.

46. Боргест, Н.М. Прикладные онтологии проектирования / Н.М. Боргест // Онтология проектирования. – 2017. – N 1(7). – С. 7-33.

47. Griffo, C., Almeida, J.P., Guizzardi, G., Nardi, J.C.. From an Ontology of Service Contracts to Contract Modeling in Enterprise Architecture / C. Griffo // 9th International Workshop on Vocabularies, Ontologies and Rules for the Enterprise. – 2017.

48. Sales, T.P., Guarino, N., Guizzardi, G., Mylopoulos, J. An Ontological Analysis of Value Propositions / T.P. Sales // 9th International Workshop on Vocabularies, Ontologies and Rules for the Enterprise. – 2017. – P. 84-193

49. Gero, J.S., Kannengiesser, U. The Function-Behaviour-Structure ontology of design, in Amaresh Chakrabarti and Lucienne Blessing / J.S. Gero // An Anthology of Theories and Models of Design. – Springer, 2014. – P. 263-283.

50. Green, St., Southee, D., Boult J. Towards a Design Process Ontology / St. Green // An International Journal for All Aspects of Design. – 2014. – Volume 17. – Issue 4. – P. 515-537.

51. Hasan, B., Wikander, J., Onori, M. Ontological Approach to Share Product Design Semantics for an Assembly / B. Hasan // Proceedings of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2016). – 2016. – Volume 2: KEOD.– P. 104-111.

52. Chen, J.-W., Yang, H.-J., Cui, J.-J., Zhang, J.-Sh. Concept semantics driven computer aided product innovation design / J.-W. Chen // Journal of

Computational Methods in Sciences and Engineering. – 2016& – Vol. 16. – N 3. – P. 575-590.

53. Trumbach, C. C., McKesson, Ch., Ghandehari, P., DeCan, L., Eslinger, O. Innovation and Design Process Ontology / C. Trumbach // Anticipating Future Innovation Pathways Through Large Data Analysis. – 2016. – P. 133-151.

54. Miah, Sh. J., Islam, H., Samsudin, A.Z.H. Ontology Techniques for Representing the Problem of Discourse: Design of Solution Application Perspective / Sh. J. Miah // IEEE International Conference on Computer and Information Technology (CIT). – 2016. – P. 148-152.

55. Наместников, А. М. Разработка инструмента инженерии онтологии в интеллектуальном проектном репозитории / А. М. Наместников, Р. А. Субхангулов // Автоматизация процессов управления. – 2012. – N 2. – С. 38-43.

56. Наместников, А. М. Применение тезаурусов и онтологий в интеллектуальных архивах проектной документации / А. М. Наместников, Н. Г. Ярушкина // Наукоемкие технологии. – 2013. – N 5, Т. 14. – С. 79-86.

57. Наместников, А. М. Онтологическая модель контекстного поиска электронных документов в архиве проектной организации / А. М. Наместников, Р. А. Субхангулов // Радиотехника. – 2015. – N 6.– С. 73-78.

58. Наместников, А. М. Онтологический подход к структурированию знаний проектной организации / А. М. Наместников // Радиотехника. – 2016. – N 9. – С. 77-83.

59. Наместников, А. М. Интеграция проектных диаграмм и онтологий в задаче балансировки мощностей авиастроительного предприятия / А. М. Наместников, Г. Ю. Гуськов, Н. Г. Ярушкина, Т. В. Афанасьева, В. Н. Негода, М. К. Самохвалов, А. А. Романов // Автоматизация процессов управления. – 2017. – N 4. – С. 85-93.

60. Namestnikov, A.M., Filippov, A.A., Avvakumova, V.S. An Ontology-Based Model of Technical Documentation Fuzzy Structuring / A.M. Namestnikov // Proceedings of the 2nd International Workshop on Soft Computing Applications and Knowledge Discovery (SCAKD 2016). – 2016. –P. 63-74.

61. Namestnikov, A.M., Guskov, G.U., Yarushkina, N.G. Approach to the search for similar software projects based on the UML ontology / A.M. Namestnikov // 2nd International Scientific Conference proceedings «Intelligent Information Technologies for Industry» (IITI-2017). – 2017. – P. 3-10.

62. Pan, J.Z., Staab, St., Aßmann, U., Ebert, J., Zhao Y. Case Studies for Marrying Ontology and Software Technologies / Jeff Z. Pan, Steffen Staab, Uwe Aßmann, Jürgen Ebert, Yuting Zhao // Ontology-Driven Software Development. – Springer, 2013.

63. Happel, H.J. KOntoR: An Ontology-enabled Approach to Software Reuse / H.J. Happel, A. Korthaus, S. Seedorf, P. Tomczyk // Proc. SEKE 2006: the 18th International Conference on Software Engineering & Knowledge Engineering (July 5-7, 2006, California, USA, 2006). – 2006. – P. 349-354.

64. Olszewska, J.I. ODYSSEY: Software Development Life Cycle Ontology / J.I. Olszewska, I.K. Allison // Proc. International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (Seville, Spain 18 Sep 2018-20 Sep 2018). – 2018. – P. 303-311.

65. Rochaa, R. DKDOnto: An Ontology to Support Software Development with Distributed Teams / R. Rochaa, A. Ara´ujoa, D. Cordeiroa, A. Ximenesa, J. Teixeiraa, G. Silvaa, D. da Silvaa, D. Espinharaa, R. Fernandesa, J. Ambrosiob, M. Duartec, R. Azevedo // Proc. 22nd International Conference on Knowledge-Based and Intelligent Information & Engineering Systems. – Procedia Computer Science 126, 2018. – P. 373-382.

66. Alobaid, A. Automating ontology engineering support activities with OnToology [Электронный ресурс] / A. Alobaid, D. Garijo, M. Poveda-Villalón, I.

Santana-Perez, A. Fernandez-Izquierdo, O. Corcho // Journal of Web Semantics. – N 57. – 2019. – Режим доступа: https://ssrn.com/abstract=3260516.

67. Головков, В., Портнов А., Чернов В. RDF — инструмент для неструктурированных данных [Электронный ресурс] / В. Головков // Открытые системы. СУБД – 2012. – N 09. – Режим доступа: https://www.osp.ru/os/2012/09/13032513.

68. Musen, M.A. The Protégé project: A look back and a look forward / M.A. Musen // AI Matters. Association of Computing Machinery Specific Interest Group in Artificial Intelligence. – 2015. – N 1(4). – P. 4-12.

69. Steinmetz, Ch., Schroeder, G., Roque, A., Pereira, C., Wagner, C., Saalmann, Ph., Hellingrath, B. Ontology-driven IoT code generation for FIWARE / Ch. Steinmetz // 2017 IEEE 15th International Conference on Industrial Informatics (INDIN). – 2017.

70. Lal, M. Neo4j Graph Data Modeling / M. Lal. – Packt Publishing, 2015. – 119 P.

71. Bechhofer, S., Horrocks, I., Goble, C., Stevens, R. OilEd: a Reasonable Ontology Editor for the Semantic Web / S. Bechhofer, I. Horrocks, C. Goble, R. Stevens // Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence, September 19-21, Vienna. Springer-Verlag LNAI. – 2001. – Vol. 2174. – P. 396-408.

72. Bārzdiņš, J., Bārzdiņš, G., Čerāns, K., Liepiņš, D., Sproģis, A. UML Style Graphical Notation and Editor for OWL 2 / J. Bārzdiņš, G. Bārzdiņš, K. Čerāns, D. Liepiņš, A. Sproģis // Perspectives in Business Informatics Research, Lecture Notes in Business Information Processing. – 2010. – Volume 64. – Part 2. – P. 102-114.

73. Weichbroth, P. Fluent Editor and Controlled Natural Language in Ontology Development / P. Weichbroth // International Journal on Artificial Intelligence Tools. – 2019. – N 28(04). – P. 243.

74. Lamy, J.-B.. Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies / J.-B. Lamy // Artificial Intelligence in Medicine. – 2017.

75. Соснин, П.И. Персональная онтология профессионального опыта / П.И. Соснин // Conference: Open Semantic Technologies for Intelligent Systems (OSTIS-2014). 2014 – Vol. 1– P. 147-154.

76. Овдей, О.М., Проскудина, Г.Ю. Обзор инструментов инженерии онтологий [Электронный ресурс] / О.М. Овдей // Труды 6-ой Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции». – 2004. – Т. 7. – Вып. 4. – Режим доступа: https://elbib.ru/article/download/254/253.

77. Чистякова, И.С. Инженерия онтологий / И.С. Чистякова // Инжерения программного обеспечения. – 2014. – №4 (20). – С. 53-68.

78. Kapoor, B., Savita, Sh. A Comparative Study Ontology Building Tools for Semantic Web Applications [Электронный ресурс] / B. Kapoor, Sh. Savita // International Journal of Web & Semantic Technology. – 2010. – Vol.1. – Num. 3. – Режим доступа: https://core.ac.uk/download/pdf/205969664.pdf.

79. Cardoso, J., Lisete, A., Escórcio, N. Editing Tools for Ontology Construction [Электронный ресурс] / J. Cardoso, A. Lisete, N. Escórcio. // Semantic Web Services: Theory, Tools and Applications. – 2007. – Режим доступа: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.119.8835&rep=rep1&type=pdf.

80. Alatrish, E.. Comparison Some of Ontology Editors / E. Alatrish // Management Information Systems. – 2013. – Vol. 8. – No. 2. – P. 18-24.

81. Rastogi, N., Verma, P., Kumar, P.. Analyzing Ontology Editing Tools For Effective Semantic Technology / N. Rastogi, P. Verma, P. Kumar // International Journal of Engineering Sciences & Research Technology. 2017. – N 6(5). – P. 40-47.

82. Barrasa, J. RDF Triple Stores vs. Labeled Property Graphs: What's the Difference? [Электронный ресурс] / J. Barrasa // Neo4j Blog. – 2017. – Режим доступа: https://neo4j.com/blog/rdf-triple-store-vs-labeled-property-graph-difference/.

83. Neosemantics(n10s) User Guide [Электронный ресурс] / Neo4j Labs Team. – Режим доступа: https://neo4j.com/labs/neosemantics/4.0/.

84. Dorst, K. The Nature of Design Thinking / K. Dorst // DTRS8 Interpreting Design Thinking: Design Thinking Research Symposium Proceedings. – 2009. – P. 131–139.

85. Gibbons, S. Design Thinking [Электронный ресурс] / Sarah Gibbons // Nielsen Norman Group. – 2016. – Режим доступа: https://www.nngroup.com/articles/design-thinking/.

86. Горшков, С.В. Онтологическое моделирование предприятий: методы и технологии : монография ; [отв. ред. С. В. Горшков] ; предисл. С. В. Горшкова. – Екатеринбург : Изд-во Урал. ун-та, 2019.– 236 с.

87. Brambilla, M., Cabot, J., Wimmer, M. Model-Driven Software Engineering in Practice. Second Edition / M. Brambilla. – Morgan & Claypool Publishers, 2012. – 187 p.

88. Frankel, D. S. Model Driven Architecture: Applying MDA to Enterprise Computing / David S. Frankel. – John Wiley & Sons, 2003.

89. Ward M. Language Oriented Programming / Martin Ward // Software Concepts and Tools. – 1994. – N 15(4). – P. 147-161.

90. Schäfer, Ph. OnToCode: Template-based code-generation from ontologies / Ph. Schäfer // Journal of Open Source Software. – 2019. – N 4(40).

91. Knublauch, H. Ontology-driven software development in the context of the semantic web: An example scenario with protege/owl / H. Knublauch // 1st International workshop on the model-driven semantic web (MDSW2004). – 2004. – P. 381-401.

92. Соснин, П.И., Пушкарева, А.А. Средства онтологического сопровождения проектного мышления / П.И. Соснин, А.А. Пушкарева // Труды Конгресса по интеллектуальным системам и информационным технологиям «IS&IT'17». Научное издание в 3-х томах. – Таганрог: Изд-во Ступина С.А., 2017. – Т. 2. – С. 366-374

93. Соснин, П.И. Вопросно-ответное программирование человеко-компьютерной деятельности / П.И. Соснин. – Ульяновск : УлГТУ, 2010. – 240 с.

94. Назаров, С.В. Архитектуры и проектирование программных систем: монография / С.В. Назаров. — М.: ИНФРА-М, 2013. — 412 с.

95. Брукс, Ф. Проектирование процесса проектирования: записки компьютерного эксперта. : Пер. с англ. — М. : ООО "И.Д. Вильямс", 2013. — 464 с.: ил. — Парал. тит. англ.

96. ГОСТ 34.601-90. Информационная технология. Комплекс стандартов на Автоматизированные системы. Автоматизированные системы. Стадии создания. М.: Издательство стандартов. – 1991.

97. Davis, A. M. Software Prototyping / Alan M. Davis // Advances in Computers. – 1995. – Vol. 40. – P. 39-63.

98. Вивек К. Продажи и дистрибуция (SD) [Электронный ресурс] / К. Вивек // Внедрение SAP R/3: Руководство для менеджеров и инженеров. – Режим доступа: https://it.wikireading.ru/48336.

99. Ермакович, М. В. Автоматическое определение границ слова в русском тексте с помощью комплекса лингвистических правил [Электронный ресурс] / М.В. Ермакович // Компьютерная лингвистика и интеллектуальные технологии: по материалам международной конференции «Диалог 2017». – 2017. – Режим доступа: http://dialog-21.ru/media/3981/yermakovichmv.pdf.

100. Bird, S., Loper, E., Klein, E. Natural Language Processing with Python / Steven Bird, Edward Loper, Ewan Klein. – Sebastopol : O'Reilly Media Inc., 2009. – 479 P.

101. Korobov, M. Morphological Analyzer and Generator for Russian and Ukrainian Languages / M. Korobov // Analysis of Images, Social Networks and Texts. – 2015. – P. 320-332.

102. Bouge, K.. Download stop words [Электронный ресурс] / Bouge K. – Режим доступа: https://sites.google.com/site/kevinbouge/stopwords-lists.

103. Шарафутдинова, Н. С. Теория и история лингвистической науки: Учебное пособие / Н. С. Шарафутдинова. – Ульяновск: УлГТУ, 2006. – 284 с.

104. Кручинкина, Н.Д. Лексико-семантический и функционально-семантический потенциал существительного / Кручинкина Н.Д. // Научный результат. Вопросы теоретической и прикладной лингвистики. – 2017. – Т 3. – №3. – С. 22-31.

105. Валгина, Н.С. Современный русский язык: Синтаксис: Учебник / Н.С. Валгина. — 4-е изд., испр. — М.: Высш. шк., 2003 — 416 с.

106. Winston, M.E., Chaffin, R., Herrmann, D. A Taxonomy of Part-Whole Relations / M.E. Winston, R. Chaffin, D. Herrmann // Cognitive Science. – 1987. – N 11. – P. 417 – 444.

107. Рахилина, Е. В. Когнитивный анализ предметных имен: семантика и сочетаемость / Е.В. Рахилина – М.: Русские словари, 2008. — 416 с.

108. Лукашевич, Н.В. Тезаурусы в задачах информационного поиска / Н.В. Лукашевич. – М., 2010. – 396 с.

109. Лукашевич, Н.В., Лашевич, Г., Герасимова, А.А., Иванов, В.В., Добров, Б.В. Порождение тезауруса типа WordNet для русского языка / Лукашевич Н.В. // Труды конференции по искусственному интеллекту КИИ-2016. – 2016. – Т. 2. – С. 89-97.

110. Loukachevitch, N.V., Lashevich, G., Gerasimova, A.A., Ivanov, V.V., Dobrov, B.V. Creating Russian WordNet by Conversion / Loukachevitch N.V. // Proceedings of Conference on Computatilnal linguistics and Intellectual technologies Dialog-2016. – 2016. – P. 405-415.

111. Loukachevitch, N., Lashevich, G. Multiword expressions in Russian Thesauri RuThes and RuWordNet / Loukachevitch N.V. // Proceedings of the AINL FRUCT 2016. – 2016. – P. 66-71.

112. Sorensen, M.N.. Logical connectors [Электронный ресурс] // University of Washington. – 1997. – Режим доступа: https://staff.washington.edu/marynell/grammar/logicalconnectors.html.

113. Шапиро, Э.Д. Выражение причинно-следственных отношений в сложно-подчиненных предложениях различных типов / Шапиро Э.Д. // Вестник ЧГУ. – 2008. – №4.

114. Бессалов, А.Ю. Каузативные глаголы как средство выражения причинно-следственных отношений в английском и французском языках // Вестник МГОУ. Сер.: Лингвистика. - М.: Изд-во МГОУ, 2010. - № 6. - С. 8590.

115. PlantUML in a nutshell [Электронный ресурс] // PlantUML. – Режим доступа: https://plantuml.com/.

116. Поташников, Н.М. PlantUML – все, что нужно бизнес-аналитику для создания диаграмм в программной документации [Электронный ресурс] / Поташников Н.М. // Хабр. – 2018. – Режим доступа: https://habr.com/ru/post/416077/.

117. Radović, A. Csnake. Developer Interface (API) [Электронный ресурс] / A. Radović // Gitlab. – 2018. – Режим доступа: https://andrejr.gitlab.io/csnake/api.html.

118. Негода, В.Н. Реализация прототипов поведения робокара средствами веб-технологий / В.Н. Негода // Информационно-измерительные и управляющие системы. – 2020 – Т.18. – №18. – С. 57-62.

119. Негода В.Н. Прототипирование поведения объектов систем логического управления в Web-базированной САПР / В.Н. Негода // Радиотехника. – 2019. – Т. 83. – №9(14). – С. 90-94.

120. Филиппов А. А., Мошкин В. С., Гуськов Г. Ю., Ярушкина Н. Г. Применение нечеткой базы знаний проблемной области в задаче поиска архитектурно подобных программных проектов / А. А. Филиппов, В. С. Мошкин, Г. Ю. Гуськов, Н. Г. Ярушкина // Нечеткие системы и мягкие вычисления. – 2017. – Т. 12. – Выпуск 2. – С. 107–120.

121. Тушканова, О.Н., Самойлов, В.В. Knowledge Net: модель и система накопления, представления и использования знаний и данных. // Онтология проектирования. – Том 9. – №1(31). – 2019. – С. 117–131.

122. Городецкий, В.И. Искусственный интеллект – наука и информационная технология: настоящее и будущее. // Материалы конференции «Информационные технологии в управлении». – 2020. – С. 3-14.

123. Соснин, П.И., Маклаев, В.А. Формирование и использование вопросно-ответных сетей в проектировании автоматизированных систем. // Автоматизация процессов управления. – 2017. – N 4. – С. 75-84.

124. Grenning J. Planning Poker or How to avoid analysis paralysis while release planning. – [Электронный ресурс] – Режим доступа: https://wingman-sw.com/articles/planning-poker.

125. Planning Poker. – [Электронный ресурс] / 2021. – Режим доступа: https://www.planningpoker.com/.

126. Негода, В.Н. Автоматное программирование логического управления. – [Электронный ресурс] / 2021. – Режим доступа: http://ofap.ulstu.ru/resources/7005/list.

Приложение 1. Вопросно-ответный протокол анализа дискурсов

Проектные рассуждения	**Теоретическое обоснование исследования**	**Тип конструкта**
D1. Для повышения эффективности концептуальной активности проектировщика в ситуациях, возникающих при решении назначенных ему задач целесообразно разработать средства, обеспечивающие достижение необходимого и достаточного понимания, регистрируемого в вербально-графической форме в рассуждениях и проектных документах.	***U1.*** Необходимое и достаточное понимание, достигаемое проектировщиком в процессе концептуальной активности в отношении проектируемой системы, должно регистрироваться в вербально-графической форме в рассуждениях и проектных документах.	Установка
Q'1.1. Что собой представляет и как проявляется феномен понимания проектного задания? ***A'1.1.*** Понимание – это природно-искусственный феномен, нацеленный на контроль условно-деятельностных рефлексов по реализации опыта формирования проектных решений.	***C1.*** Понимание – это природно-искусственный феномен, нацеленный на контроль условно-деятельностных рефлексов по реализации опыта формирования проектных решений.	Понятие
Q'1.1.1. Что есть условно-деятельностный рефлекс? ***A'1.1.1.*** Условно-деятельностный рефлекс есть реакция субъекта на сложившуюся ситуацию в определенных условиях, возникающая в процессе осуществления деятельности, продукты которой рассматриваются как прототипы.	***C2.*** Условно-деятельностный рефлекс есть реакция субъекта на сложившуюся ситуацию в определенных условиях, возникающая в процессе осуществления деятельности, продукты которой рассматриваются как прототипы.	Понятие
Q'1.1.1.1. В чем и как проявляются условия, при которых возникает условно-деятельностный рефлекс?	***S2.*** Архитектурное понимание ситуации складывается в результате вербального или графического выражения определенных условий, сложившихся в процессе осуществления	Утверждение

A'1.1.1.1. Условия могут быть выражены словесно (вербально) или графически (с помощью рисунков и схем). Таким образом, складывается архитектурное понимание ситуации.	деятельности, продукты которой рассматриваются как прототипы.	
Q'1.1.1.1.1. Что есть результат архитектурного понимания? ***A'1.1.1.1.1.*** Результатом архитектурного понимания является изложение планируемого решения задачи в виде рисунка, сформированного в ходе модификации прототипа.	***U5.*** Для достижения необходимого и достаточного архитектурного понимания задачи необходимо отражать концептуальное решение в онтологии проекта, последовательно формируя онтологическую модель проекта, а также итеративно представлять проектные решения данной задачи, формируемые на этапе объектно-ориентированного моделирования, пока все они не придут к устоявшемуся состоянию, при этом согласовывая их с онтологией проекта.	Установка
Q'1.1.1.1.2. Что представляет собой процесс достижения архитектурного понимания? ***A'1.1.1.1.2.*** Процесс достижения архитектурного понимания может представлять собой итеративное формирование рисунка до тех пор, пока он не придет к устоявшемуся состоянию, т.е. он будет включать в себя все необходимые элементы, а также на нем будут отсутствовать элементы, затрудняющие обработку данных.		
Q'1.1.1.1.3. Каким образом контролируется процесс достижения архитектурного понимания? ***A'1.1.1.1.3.*** Контроль архитектурного понимания происходит путем отражения концептуальных составляющих рисунков и схем, а также рассуждений, являющихся продуктами концептуального моделирования в проектной онтологии в ее текущем состоянии, а также согласования таких проектных решений, как, например, диаграммы классов, ER-диаграммы,		

схемы организации баз данных и др., с проектной онтологией на этапах объектно-ориентированного моделирования.		
Q'1.1.1.2. В чем и как проявляется реакция условно-деятельностного рефлекса? ***A'1.1.1.2.*** Реакция представляет собой следствие условий, в которых проявляется рефлекс. Таким образом, прогнозирование реакций позволяет субъекту достичь причинно-следственного понимания сложившейся ситуации.	***S3.*** Причинно-следственное понимание возникает в результате прогнозирования реакций, т.е. следствий тех условий, которые сложились в процессе осуществления деятельности, продукты которой рассматриваются как прототипы.	Утверждение
Q'1.1.1.2.1. Что есть результат причинно-следственного понимания? ***A'1.1.1.2.1.*** Результатом причинно-следственного понимания является единица опыта, выраженная в логической формуле вида: ЕСЛИ (причина), ТО (следствие).	***U6.*** Для достижения причинно-следственного понимания необходимо сформировать проектное решение задачи, выраженное в наборе логических формул, согласованных между собой и не противоречащим онтологии проекта.	Установка
Q'1.1.1.2.2. Что представляет собой процесс достижения причинно-следственного понимания? ***A'1.1.1.2.2.*** Процесс достижения причинно-следственного понимания представляет собой создание проектного решения, которое формируется по ходу появления логических формул и их проверки.		
Q'1.1.1.2.3. Каким образом контролируется процесс достижения причинно-следственного понимания?		

A'1.1.1.2.3. Создание проектного решения контролируется путем согласования его с текущим состоянием проектной онтологии: в частности, извлечения из нее причинно-следственных связей и корректировки решения с учетом этих связей.		
Q'1.1.2. В чем заключается контроль условно-деятельностных рефлексов в рамках концептуального проектирования? ***A'1.1.2.*** Контроль условно-деятельностных рефлексов заключается, с одной стороны, в проверке корректности архитектурного (соответствие условий сложившейся ситуации) и, с другой стороны, причинно-следственного (соответствие возможной реакции ожидаемому результату) понимания. Понимание корректно, когда оно является одновременно необходимым и достаточным.	***S3.*** Понимание корректно, когда оно является одновременно необходимым и достаточным.	Утверждение
Q'1.1.2.1. Что включает в себя необходимое и достаточное понимание? ***A'1.1.2.1.*** Необходимое понимание предполагает отсутствие ошибок, неоднозначностей и упущенных единиц понимания (т.е. неполноты), в то время как достаточное понимание предполагает отсутствие лишней информации, затрудняющих ее обработку.	***Ds1.*** Необходимое понимание предполагает отсутствие ошибок, неоднозначностей и упущенных единиц понимания (т.е. неполноты), в то время как достаточное понимание предполагает отсутствие лишней информации, затрудняющих ее обработку.	Описание
Q'1.1.2.1.1. Что есть единица понимания?	***C3.*** Понятие есть единица понимания, которая обеспечивает детализацию объекта понимания.	Понятие

A'1.1.2.1.1. Единица понимания есть понятие, которое обеспечивает детализацию объекта понимания (например, задачи).		
Q'1.1.3. В чем выражается природная составляющая феномена понимания? ***A'1.1.3.*** Природная составляющая феномена понимания выражается в том, что достижение понимания есть естественный процесс, свойственный человеческому мозгу от природы.	–	–
Q'1.1.4. В чем выражается искусственная составляющая феномена понимания? ***A'1.1.4.*** Искусственная составляющая феномена понимания выражается в том, что понимание может быть зарегистрирована в искусственно созданных формах, например, в виде рисунков, схем, логических формул и т.п.	–	–
D2. В целях избегания незапланированных трат на доработку системы, которая не отвечает требованиям ее потребителя, необходимо разработать средства, обеспечивающие сокращение семантического разрыва между представлением информации на различных стадиях проектирования.	***Pr1.*** Использование семантических технологий для преодоления семантического разрыва между представлением проекта на различных стадиях проектирования.	Принцип
Q'2.1. Каким образом выражаются требования потребителя к системе? ***A'2.1.*** Требования потребителя к системе выражаются при помощи различных спецификаций проекта – таких, как,		

например, вопросно-ответные спецификации, формируемые в ходе взаимодействия заказчика (потребителя) и исполнителя по проекту; на более поздних стадиях анализа требований они материализуются в форме технического задания к проекту.		
Q'2.2. Что есть семантический разрыв между представлением информации на различных стадиях проектирования? ***A'2.2.*** Семантический разрыв есть различие между значением понятий, материализованных в проекте, возникающее вследствие изменения способа представления информации на различных стадиях проектирования.	***C4.*** Семантический разрыв есть различие между значением понятий, материализованных в проекте, возникающее вследствие изменения способа представления информации на различных стадиях проектирования.	Понятие
Q'2.3. На каких стадиях проектирования может возникать семантический разрыв? ***A'2.3.*** Семантический разрыв может возникнуть на всех стадиях проектирования АС, а именно на стадиях формирования требований, концептуального проектирования, технического проектирования и рабочего проектирования.		
Q'2.3.1. Какие способы представления информации используются на стадии формирования требований? ***A'2.3.1.*** На стадии формирования требований, главным образом, используется текстовое представление информации (вопросно-ответные протоколы взаимодействия заказчика с исполнителем, тексты рассуждений и описания требований, первичная проектная документация, в том числе техническое задание).		

Q'2.3.2. Какие способы представления информации используются на стадии концептуального проектирования? ***A'2.3.2.*** На стадии концептуального проектирования может использоваться как текстовое представление информации (описание концепции проекта, описание системы и др.), так и формализованное представление информации (например, для формализации может использоваться онтологическая модель или другие средства моделирования).		
Q'2.3.3. Какие способы представления информации используются на стадии технического проектирования? ***A'2.3.3.*** На стадии технического проектирования, главным образом, используется формализованное представление информации (нотации UML, BPMN, ER, IDEF и другие), также используется текстовое представление информации (описания разработанных моделей и алгоритмов).		
Q'2.3.4. Какие способы представления информации используются на стадии рабочего проектирования? ***A'2.3.4.*** На стадии рабочего проектирования, главным образом, используется алгоритмическое и реляционное представление информации (исходные тексты программ, спецификации таблиц баз данных, тесты, программы испытаний).		
Q'2.4. Каким образом или с помощью каких средств можно преодолеть семантический разрыв?		

A'2.4. Семантический разрыв можно преодолеть за счет использования семантических технологий на всех стадиях проектирования – в частности технологии онтологического моделирования, позволяющей воплотить в электронном виде информацию с учетом ее семантики и обрабатывать ее, что способствует обеспечению концептуальной целостности проекта и согласованности на всех этапах работы над ним.		
D3. Поскольку формирование концептуальной модели базируется на прототипах, являющихся продуктами опыта по созданию проектных решений, что приводит к возникновению многих пространств имен в проекте, целесообразно включить в средства поддержки концептуального проектирования такие инструменты, которые обеспечивали бы концептуальную целостность в рамках этих пространств имен.	***S1.*** Создание концептуальной модели проекта всегда базируется на продуктах опыта по созданию проектных решений, которые становятся прототипами по отношению к концептуальной модели; данные прототипы носят лоскутный характер и практически всегда приводят к появлению многих пространств имен в проекте.	Утверждение
	U2. Для обеспечения концептуальной целостности проекта, необходимо осуществлять инструментальную поддержку существования многих пространств имен на всех стадиях работы над проектом.	Установка
Q'3.1. Что есть концептуальная целостность и как она проявляется в проектировании? ***A'3.1.*** Концептуальная целостность есть такое свойство автоматизированной системы, которое позволяет охарактеризовать все ее компоненты, а также проектные	***C5.*** Концептуальная целостность есть такое свойство автоматизированной системы, которое позволяет охарактеризовать все ее компоненты, а также проектные решения, формируемые в ходе	Понятие

решения, формируемые в ходе ее разработки как согласованные, полные и непротиворечивые.	ее разработки как согласованные, полные и непротиворечивые.	
Q'3.1.2. За счет чего достигается концептуальная целостность в проектировании? ***A'3.1.2.*** Т.к. концептуальное единство определяется информационным обеспечением систем автоматизированного проектирования, к которым относится онтология проектирования, применение средств онтологической поддержки проекта может стать эффективным средством достижения концептуальной целостности.	***G1.*** Использование онтологии проекта в качестве продукта формализации его концептуальной модели может являться эффективным средством обеспечение концептуальной целостности проекта.	Гипотеза
Q'3.2. Какие пространства имен могут возникать в проекте? ***A'3.2.*** В процессе проектирования АС могут возникать многие пространства имен, обозначающих одни и те же понятия разными способами: такие пространства имен могут быть обусловлены как конкретными языками моделирования и программирования, так и уникальным профессиональным опытом проектировщиков.		
Q'3.2.1. Какие последствия могут быть вызваны возникновением в процессе проектирования многих пространств имен? ***A'3.2.1.*** Наличие несогласованных пространств имен приводит к значительному повышению трудозатрат на достижение понимания проектного задания; в то же время, наличие многих согласованных пространств имен, напротив, снижает трудозатраты на создание конкретных проектных решений,		

поскольку позволяет специалисту осуществлять свою деятельность с использованием привычных ему терминов.		
Q'3.2.1.1. С помощью чего может быть достигнута согласованность многих пространств имен? ***A'3.2.1.1.*** Согласованность многих пространств имен в проектировании может быть достигнута с помощью использования онтологии проекта.		
Q'3.3. Какие продукты опыта могут быть использованы в проектировании в качестве прототипов? ***A'3.3.*** В качестве прототипов могут быть использованы такие продукты опыта, как проектные документы, спецификации, стандарты, проектные решения, относящихся к смежным проектам, а также части уже разработанных автоматизированных систем.		
D4. С целью предотвращения и обнаружения ошибок в используемых рассуждениях и проектных документах, порождаемых в процессе разработки систем с программным обеспечением и их семейств, целесообразно сформировать и использовать словарь контролируемой лексики.	***Pr2.*** Контролируемое формирование и использование языка проекта, которое заключается в том, что любой его конструкт должен найти овеществление в разрабатываемом проекте.	Принцип
	U3. Концептуальная активность проектировщиков на протяжении всех ее стадий должна управляться языком проекта, что способствует предотвращению и обнаружению	Установка

	ошибок в порождаемых ими рассуждениях и проектных документах.	
Q'4.1. Какие ошибки могут возникать в текстовых единицах, возникающих в процессе разработки систем с ПО? ***A'3.1.*** Среди возможных ошибок, встречающихся в текстовых единицах, – использование неконтролируемой, неоднозначной и неопределенной лексики.	–	–
Q'4.1.1. За счет чего словарь контролируемой лексики может способствовать предотвращению и обнаружению ошибок? ***A'4.1.1.*** Разрабатываемый словарь контролируемой лексики способствует предотвращению и обнаружению ошибок за счет использования зрелых механизмов логико-лингвистической обработки в процессе извлечения из текстовых документов лексем и групп лексем. При этом, из документов извлекаются все лексические единицы, среди которых могут быть такие, которые уже содержатся в разрабатываемом словаре и позволяют расширять его за счет добавления нового варианта использования уже имеющейся словарной единицы, и такие, которых еще нет в разрабатываемом словаре, что позволяет его наполнять и обеспечивает его открытость.	–	–
Q'4.1.1.1. Что такое лексическая единица? ***A'4.1.1.1.*** Лексическая единица есть лексема или группа лексем, которая потенциально может рассматриваться в качестве концепта.	–	–

Q'4.1.1.2. Какие механизмы логико-лингвистической обработки используются в процессе извлечения из текстовых документов лексических единиц? ***A'4.1.1.2.*** Среди механизмов логико-лингвистической обработки: автоматизированный морфологический анализ, синтаксический анализ, фильтрация единиц контролируемой лексики с применением специальных словарей.	–	–
Q'4.1.1.2.1. Каким образом осуществляется морфологический анализ? ***A'4.1.1.2.1.*** Автоматизация процесса морфологического анализа текстовых единиц достигается посредством использования специальных средств прикладной лингвистики, а именно – морфологического анализатора, который позволяет определить морфологические характеристики, а также частеречную принадлежность каждой лексической единицы.	–	–
Q'4.1.1.2.2. Каким образом осуществляется синтаксический анализ? ***A'4.1.1.2.2.*** Синтаксический анализ осуществляется как автоматически – за счет использования внешней программы (синтаксического анализатора), так и автоматизировано – за счет применения синтаксических моделей, которые используются для извлечения из текстовых документов групп лексем, связанных синтаксическими отношениями, поддающиеся жесткой классификации и позволяющие сформировать совокупность синтаксических правил обработки.	–	–

Q'4.1.1.2.3. Какие специальные словари могут быть использованы в процессе фильтрации лексических единиц? ***A'4.1.1.2.3.*** В процессе фильтрации используются словари стоп-слов, словари сокращений, терминологические словари, а также в некоторых случаях – словарь контролируемой лексики.	–	–
Q'4.2. Каким образом формируется открытый словарь контролируемой лексики? ***A'4.2.*** Открытый словарь контролируемой лексики формируется путем извлечения лексем и групп лексем из текстовых единиц, накапливаемых в процессе проектирования и разработки систем с программным обеспечением и их семейств, таким образом, чтобы результаты обработки данных документов в виде единиц контролируемой лексики могли управляемо расширять словарь.	***S5.*** Необходимо формировать открытый словарь контролируемой лексики проекта путем извлечения лексем и группы лексем из текстовых единиц, накапливаемых в процессе проектирования и разработки систем с программным обеспечением и их семейств.	Утверждение
Q'4.2.1. Что представляет собой единица контролируемой лексики? ***A'4.2.1.*** Единица контролируемой лексики – это основа для словаря контролируемой лексики, унифицированная, однозначная лексическая единица, которая может быть использована в проектных документах и рассуждениях. Совокупность единиц контролируемой лексики формирует язык проекта.	***Ds2.*** Язык проекта состоит из совокупности единиц контролируемой лексики, каждая из которых является унифицированной, однозначной лексической единицей, которая может быть использована в проектных документах и рассуждениях.	Описание
D5. С целью организации хранения информации о проектах систем с программным обеспечением и их семейств, а также оперативного использования данной информации	***Pr3.*** Материализация языка проекта в форме онтологии; при этом лексические единицы, входящие в нее, могут быть заимствованы из	Принцип

проектировщиком в процессе решения стоящих перед ним задач, целесообразно использовать словари контролируемой лексики, построенные по образцу словарей семантических типов, в которых была бы отражена семантика понятий и связей между ними.	языков родственных проектов, а также из специальной лексики той предметной области, которая соответствует разрабатываемому проекту.	
Q'5.1. Какие словари семантических типов можно использовать в качестве образца для построения словаря контролируемой лексики? ***A'5.1.*** Среди словарей семантических типов можно выделить толковые словари и тезаурусы.	–	–
Q'5.1.1. В чем заключаются особенности построения толковых словарей? ***A'5.1.1.*** Толковые словари построены таким образом, что содержат понятие и одно или несколько определений к нему.	–	–
Q'5.1.2. В чем заключаются особенности построения тезаурусов? ***A'5.1.2.*** Тезаурусы построены таким образом, что содержат понятия и связи между ними различных типов.	–	–
Q'5.1.1.2. В какой форме целесообразно построить словарь контролируемой лексики, чтобы была возможность хранить и извлекать информацию о понятиях, их определениях и связях?	***U7.*** В проектную онтологию необходимо включать понятия, которые используются в ходе разработки проекта, характеризуют его и служат	Установка

A'5.1.1.2. Словарь контролируемой лексики целесообразно организовать в форме проектной онтологии.	для его описания – иными словами составляют ядро проекта.	
Q'5.1.1.2.1. Какие понятия необходимо включать в проектную онтологию? ***A'5.1.1.2.1.*** В проектную онтологию необходимо включать понятия, которые составляют понятийное ядро проекта.		
Q'5.1.1.2.1.1. Что есть понятийное ядро проекта? ***A'5.1.1.2.1.1.*** Понятийное ядро включает в себя понятия, которые используются в ходе разработки конкретного проекта, характеризуют его и служат для описания его отличительных особенностей, архитектуры, способов реализации и т.д.		
D6. В потенциально возможных действиях проектировщика целесообразно передать компьютеру ту часть работ, которая может выполняться автоматически.	***U4.*** В ходе решения задач проектировщик должен принимать решение о включении в контролируемый язык проекта тех или иных понятий исходя из того, найдут ли они свое овеществление в проекте, опираясь при этом на предлагаемые решения используемых программных средств логико-лингвистической обработки.	Установка
Q'6.1. Какие работы могут выполняться автоматически? ***A'6.1.*** Поскольку действия проектировщика в ходе разработки автоматизированных систем предполагают большой объем работы по анализу текстовых единиц (проектных документов,	–	–

текстов рассуждений и т.д.), целесообразно выполнять часть таких работ автоматически.		
Q'6.1.1. За счет чего могут быть автоматизированы работы по анализу текстовых единиц? ***A'6.1.1.*** Работы могут быть автоматизированы за счет использования специальных программ логико-лингвистической обработки в процессе обработки текстовых единиц, порождаемых в ходе проектирования автоматизированных систем.	–	–
Q'6.1.1.1. Какие программы логико-лингвистической обработки могут быть использованы для автоматизации? ***A'6.1.1.1.*** Для автоматизации могут быть использованы программы лексического и синтаксического анализа, а также специальные словари.	–	–
Q'6.2. Какие работы не могут выполняться автоматически, т.е. какие работы проектировщику необходимо осуществлять самостоятельно? ***A'6.2.*** За проектировщикам остается работа по непосредственному принятию решений, а также контроль над корректностью выполнения автоматических работ, поскольку механизмы логико-лингвистической обработки текстов всегда совершаются с небольшой погрешностью.	–	–

Q'6.3. Какие механизмы целесообразно использовать для автоматизации и каким образом должны быть организованы разрабатываемые средства? ***A'6.3.*** С учетом того, что автоматизировать возможно лишь часть работ, при этом некоторые из них могут выполняться параллельно и независимо друг от друга, целесообразно разработать несколько агентов, каждый из которых выполнял бы одну из необходимых работ. Таким образом, в разрабатываемых средствах целесообразно использовать механизм многоагентной поддержки.	–	–

Приложение 2. Вопросно-ответный протокол анализа задачи исследования в промежуточном состоянии

Z1. Для повышения качества концептуальной активности проектировщиков, разрабатывающих системы с программным обеспечением, создать комплекс средств, позволяющий оперативно формировать и использовать язык проекта, употребление которого должно способствовать сокращению семантического разрыва между представлением информации на различных стадиях проектирования, а также предотвращению и обнаружению семантических ошибок, в том числе за счет достижения необходимого и достаточного понимания и его регистрации для повторного использования.
Q1. Что есть концептуальная активность проектировщика? ***A1.*** Концептуальная активность проектировщика есть работа проектировщика по подготовке концептуального решения стоящей перед ним задачи.
Q1.1. Что есть концептуальное решение задачи? ***A1.1.*** Концептуальное решение задачи есть описание на естественно-профессиональном языке (в виде текстов, рисунков, логических и алгоритмических формул) совокупности действий для реагирования, способствующего полезному (задуманному, целевому, ожидаемому) выходу из задачной ситуации, созданное на базе доступного опыта (доступных прецедентов).
Q1.1.1. Что есть прецедент? ***A1.1.1.*** Прецедент есть базовая единица человеческого опыта, являющаяся результатом интеллектуальной обработки условного-деятельностного рефлекса или их совокупности.
Q1.1.2. Что есть база опыта? ***A1.1.2.*** База опыта есть упорядоченное хранилище для прецедентов, предназначенная для накопления активов повторного использования.
Q2. За счет чего можно повысить качество концептуальной активности проектировщика в ходе разработки систем с программным обеспечением? ***A2.*** Качество концептуальной активности проектировщика в ходе разработки систем с программным обеспечением можно повысить за счет использования средств, способствующих достижению необходимого и достаточного понимания задачи, стоящей перед проектировщиком.
Q3. Каким образом разрабатываемый комплекс средств формирует язык проекта? ***A3.*** Разрабатываемый комплекс средств формирует язык проекта по ходу возникновения текстовых единиц (проектных документов, рассуждений), имеющих отношение к проекту, путем извлечения понятий и связей между ними из данных текстовых единиц.
Q3.1. Каким образом происходит извлечение понятий?

A3.1. Извлечение понятий происходит путем разделения текстовых единиц на словоформы, их нормализации, фильтрации нормализованных словоформ, а также извлечения словосочетаний из текстовых единиц. Извлеченные понятия (отфильтрованные словоформы и словосочетания) добавляются в словарь контролируемой лексики по решению проектировщика и впоследствии включаются в понятийное ядро проекта.

Q3.1.1. Что есть словоформа и каким образом текстовая единица разделяется на словоформы?

A3.1.1. Словоформа – это слово в той форме, в которой оно представлено в тексте. Текстовая единица разделения на словоформы путем разбиения текста по пробелам и знакам пунктуации.

Q3.1.2. Что есть нормализация и каким образом она осуществляется?

A3.1.2. Нормализация – это приведение текста к его исходной (нормальной, начальной) форме. Нормализация осуществляется с помощью морфологического анализатора.

Q3.1.3. Каким образом происходит фильтрация нормализованных словоформ?

A3.1.3. Нормализованные словоформы могут фильтроваться с помощью словарей стоп-слов, терминологических словарей, словарей сокращений, а также алгоритмов, ориентированных на частотность слов в тексте. Также для фильтрации могут быть использованы тематические словари.

Q3.1.4. Каким образом извлекаются слловосочетания из текстовых единиц?

A3.1.4. Словосочетания извлекаются из текстовых единиц на основе правил, проверяющих каждое сочетание из 2-3 слов в анализируемом тексте на соответствие частеречной модели.

Q3.2. Каким образом происходит извлечение связей?

A3.2. Извлечение связей может быть осуществлено посредством поиска тегов.

Q3.2.1. Что есть тег и как он используется в процессе извлечения связей?

A3.2.1. Тег – это маркер определенного типа связи в тексте, представляющий собой слово. Существуют теги различных типов, в зависимости от которых формируется алгоритм поиска тегов.

Q3.2.3.1. Какие типы тегов существуют?

A3.2.3.1. В общем случае можно выделить два типа тегов: тег, маркирующий главное понятие в паре связанных понятий в паре, и тег, маркирующий зависимое понятие.

Q4. Каким образом разрабатываемый комплекс средств использует язык проекта?

A4. Разрабатываемый комплекс средств позволяет использовать язык проекта для контроля понимания стоящей перед проектировщиком задачи, контроля лексики, используемой проектировщиком в процессе создания текстовых единиц, относящихся к

проекту, а также качестве вспомогательного механизма для осуществления вопросно-ответного анализа задачи (в частности – для извлечения вопросов из текстовых единиц).
Q4.1. Что в себя включает контроль понимания и как он осуществляется в рамках разрабатываемого комплекса средств? ***A4.1.*** Контроль понимания есть процесс проверки достигнутого проектировщиком понимания стоящей перед ним задачи, зарегистрированного в определенной форме, на предмет соответствия его текущему состоянию проектной онтологии.
Q4.2. Что в себя включает контроль лексики и как он осуществляется в рамках разрабатываемого комплекса средств? ***A4.2.*** Контроль лексики есть процесс проверки соответствия лексики, используемой в текстовых единицах, порождаемых в процессе разработки систем с программным обеспечением, словарю контролируемой лексики. Данный процесс также включает в себя непосредственно процесс формирования словаря контролируемой лексики.
Q5. Каким образом употребление языка проекта способствует обнаружению и предотвращению семантических ошибок? ***A5.*** Употребление языка проекта способствует обнаружению и предотвращению семантических ошибок за счет сопоставления текстов рассуждений, формируемых на этапе концептуального проектирования, с текущем состоянием онтологии проекта.
Q6. Какие семантические ошибки можно обнаружить и предотвратить? ***A6.*** Среди возможных ошибок, встречающихся в текстовых единицах, – использование неконтролируемой, неоднозначной и неопределенной лексики, а также неполнота рассуждений.
Q7. Что есть необходимое и достаточное понимание? ***A7.*** См. Приложение 1.
Q8. Каким образом регистрируется понимание для повторного использования? ***A8.*** Понимание может быть зарегистрировано в виде текстов, рисунков, логических формул.
Q9. Что есть семантический разрыв между представлением информации на различных стадиях проектирования? ***A9.*** См. Приложение 1.
Q10. Каким образом или с помощью каких средств можно преодолеть семантический разрыв? ***A10.*** См. Приложение 1.

Z2. В разработке комплекса средств ориентироваться на представление понятийного ядра проекта в виде проектной онтологии, а в его создании – на повышение степени автоматизации оперативных действий проектировщика.

Q1. Что есть понятийное ядро?

A2. Понятийное ядро есть набор понятий, которые наиболее точно характеризуют разрабатываемый проект.

Q2. Какие оперативные действия проектировщика могут быть автоматизированы?

A2. Автоматизированы могут быть оперативные действия проектировщика, связанные с извлечением необходимых понятий из текстовых единиц (проектных документов, текстов рассуждений и т.д.); а также трансформация понятий в сущности, используемые на стадиях технического и рабочего проектирования.

Q2.1. За счет чего могут быть автоматизированы работы по анализу текстовых единиц?

A2.1. Работы могут быть автоматизированы за счет использования специальных программ логико-лингвистической обработки в процессе обработки текстовых единиц, порождаемых в ходе проектирования автоматизированных систем.

Q2.1.1. Какие программы логико-лингвистической обработки могут быть использованы для автоматизации?

A2.1.1. Для автоматизации могут быть использованы программы лексического и синтаксического анализа (морфологический и синтаксический анализаторы), а также специальные словари.

Q2.2. Какие действия включает в себя обработка текстовых единиц в процессе разработки систем с программным обеспечением?

A2.2. Работа с текстовыми единицами включает в себя следующее:

- проведение вопросно-ответного анализа, в том числе формулировку вопросов и ответов;
- выявление понятий, относящихся к языку проекта;
- выявление связей между понятиями, относящимися к языку проекта;
- определение типов выявленных связей;
- анализ текстовых единиц с целью достижения архитектурного и причинно-следственного понимания.

Приложение 3. Мотивационно-целевой анализ задачи исследования в промежуточном состоянии

Базовый мотив, которым мы руководствуемся при решении нашей задачи – повысить степени успешности разработок систем АС. Однако этот мотив – слишком общий, поскольку в своей работе мы касаемся взаимодействия с моделями знаний, базой опыта.

Нам необходимо оперативно использовать комплексирование естественного опыта и его моделей, интегрированных в базе опыта. При этом, эффективность взаимодействия с базой опыта в процессе разработки систем с программным обеспечением можно существенно повысить за счет систематизации ее содержимого с помощью прикладной онтологии.

Таким образом, сформулируем основной мотив:

M0. **Включение прикладных онтологий в состав баз опыта для систематизации их содержания должно способствовать повышению эффективности оперативного взаимодействия проектировщиков с базами опыта в процессах разработки систем с ПО.**

Эффективность взаимодействия с базами опыта повышается в том случае, когда сосредоточенные в них знания являются полезными, – следовательно, находят практическое отражение в проекте, обеспечивая тем самым сокращение семантического разрыва, о котором мы говорили в предыдущих подразделах, а также снижение количества семантических ошибок на стадии концептуального проектирования, которые впоследствии сказываются на других этапах работы над проектом.

Таким образом, следующая группа мотивов выглядит так:

M1. Сократить семантический разрыв между представлением информации на различных стадиях проектирования.

M2. Снизить количество семантических ошибок на стадии концептуального проектирования.

Говоря о семантическом разрыве, следует учитывать то, за счет чего он может сократиться. Если переводить рассуждение в более прикладное русло, то можно отметить, что семантический разрыв может сокращаться, во-первых, за счет повышения степени автоматизации при формировании онтологии, поскольку в этом случае исключаются ошибки интерпретации, обусловленные человеческим фактором (или хотя снижается их количество); а во вторых – за счет возможности использовать онтологическую модель проекта в качестве основы для создания спецификаций проектных решений и даже, в

некоторых случаях, реализации самих проектных решений. Соответственно группа дочерних мотивов по отношению к мотиву ***M1*** выглядит следующим образом:

M1.1. Повысить степень автоматизации при формировании онтологии.

M1.2. Обеспечить возможность трансформации единиц онтологии, полученных на стадии концептуального проектирования, в сущности, используемые на стадиях технического и рабочего проектирования.

Степень автоматизации при формировании онтологии может быть повышена за счет эффективного применения средств автоматического анализа текста – в частности для извлечения из текстовой информации понятий, которые впоследствии могут стать основой для формирования онтологии. Поскольку задача автоматизированного формирования онтологии является предметом исследования многих ученых-лингвистов и специалистов в области компьютерных технологий и до сих пор не решена, мы будем стремиться к тому, чтобы отсеять 80-90% лексем и групп лексем, которые не являются понятиями предметной области, чтобы облегчить задачу формирования онтологии проекта и частично ее автоматизировать.

Перейдем к мотиву ***M2***. Семантические ошибки чаще всего встречаются в проектных спецификациях, к которым относится проектная документация и другие текстовая и графическая информация о проекте. Соответственно, необходимо сконцентрироваться на поиске и исправлении, а также предотвращении ошибко именно там. Кроме того, источником ошибок служит ненормированное и неопределенное использование языка, а также недостаточное понимание проектировщиками стоящих перед ними задач. Таким образом, следующая группа мотивов выглядит так:

M2.1. Предотвратить ошибки в проектных спецификациях.

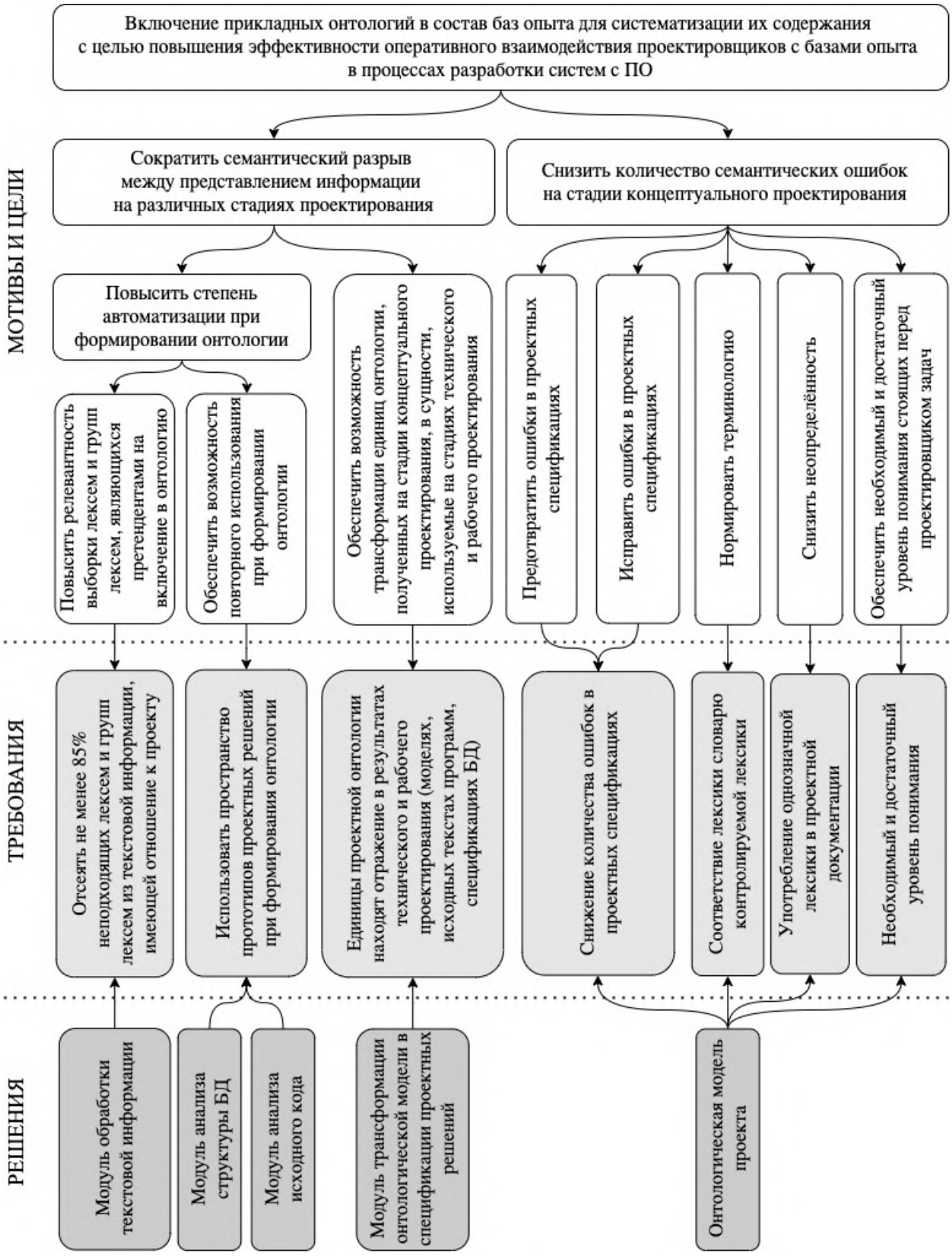
M2.2. Исправить ошибки в проектных спецификациях.

M2.3. Нормировать терминологию.

M2.4. Снизить неопределенность.

M2.5. Обеспечить необходимый и достаточный уровень понимания стоящих перед проектировщиком задач.

Обобщим наши рассуждения в виде схемы, также обозначив на ней основные требования, порождаемые указанными мотивами и проектные решения, которые могут способствовать удовлетворению данных требований.


Включение прикладных онтологий в состав баз опыта для систематизации их содержания с целью повышения эффективности оперативного взаимодействия проектировщиков с базами опыта в процессах разработки систем с ПО
МОТИВЫ И ЦЕЛИ
Сократить семантический разрыв между представлением информации на различных стадиях проектирования
Снизить количество семантических ошибок на стадии концептуального проектирования
Повысить степень автоматизации при формировании онтологии
Обеспечить возможность трансформации единиц онтологии, полученных на стадии концептуального проектирования, в сущности, используемые на стадиях технического и рабочего проектирования
Предотвратить ошибки в проектных спецификациях
Исправить ошибки в проектных спецификациях
Нормировать терминологию
Снизить неопределённость
Обеспечить необходимый и достаточный уровень понимания стоящих перед проектировщиком задач
Повысить релевантность выборки лексем и групп лексем, являющихся претендентами на включение в онтологию
Обеспечить возможность повторного использования при формировании онтологии
ТРЕБОВАНИЯ
Отсеять не менее 85% неподходящих лексем и групп лексем из текстовой информации, имеющей отношение к проекту
Использовать пространство прототипов проектных решений при формировании онтологии
Единицы проектной онтологии находят отражение в результатах технического и рабочего проектирования (моделях, исходных текстах программ, спецификациях БД)
Снижение количества ошибок в проектных спецификациях
Соответствие лексики словарю контролируемой лексики
Употребление однозначной лексики в проектной документации
Необходимый и достаточный уровень понимания
РЕШЕНИЯ
Модуль обработки текстовой информации
Модуль анализа структуры БД
Модуль анализа исходного кода
Модуль трансформации онтологической модели в спецификации проектных решений
Онтологическая модель проекта

Приложение 4. Анализ трудозатрат при разработке АС с учетом онтологического моделирования

Множество постановок задач *sPDef* будем формировать на основе множества изменений требований *sChReq*. Каждая постановка строится на основе некоторого подмножества *ssChReq* ⊂ *sChReq*. Благодаря этому оценивание может охватывать задачи перепроектирования с различным объемом изменений требований. Каждая такая задача заключается в создании целевого прототипа программы логического управления с измененной функциональностью относительно базового прототипа.

Порождение достаточно объемного множества постановок задач проще всего строить на расширении функциональных возможностей СЛУ. Нацелим это расширение на поддержку настройки платформы плиткоукладчика для повышения его производительности. Программа логического управления в соответствии с логикой функционирования робота запускает те или иные операции платформы.

В базовом прототипе в качестве основных операций выступают: инициализация, монтаж плитки, выполнение шага вперед, выполнение поворота. Предполагается, что для каждой из этих операций возможны различные задержки времени. Выполним классификацию значений этих задержек, например так:

- T ≤ Tvalid – допустимая задержка, не требующая никаких дополнительных действий по настройке;
- Tvalid < T ≤ Tshort – короткая задержка, требующая простой операции настройки ;
- Tshort < T ≤ Tmedium – средняя задержка, требующая операции настройки средней сложности;
- Tmedium < T ≤ Thight – короткая задержка, требующая сложной операции настройки;
- T > Thight – недопустимая задержка, требующая перехода в состояние аварии.

Поскольку инерционность различных операций платформы различна, для каждой такой операции имеет место свой массив граничных значений. На языке C++ пространство граничных значений для генератора параметров постановок задач можно описать следующим образом:

```
// индексы границ интервалов
enum  {VALID, SHORT, MEDIUM, HIGHT, INTERVALS_NUM};
// индексы операций платформы
// OP_INIT - инициализация, OP_SETTILE - монтаж плитки;
// OP_ROTATE - поворот;
// OP_MOVE - шаг вперед к следующему плиткоместу
enum {OP_INIT, OP_SETTILE, OP_ROTATE, OP_MOVE, OP_NUM};
// массив всех границ для всех операций
int BT[OP_NUM][INTERVALS_NUM];
```

Механизм формирования множества постановок задач *sPDef* предполагает использовать различные сочетания возможных интервалов и операций. В общем случае одна постановка задачи может охватывать от одной до OP_NUM операций и от 2 до INTERVALS_NUM интервалов. Это означает, что из всех 2^{OP_NUM} подмножеств множества операций мы не можем использовать только пустое подмножество, а из всех $2^{INTERVALS\_NUM}$ подмножеств интервалов нет используется пустое подмножество и INTERVALS_NUM подмножеств с одним элементом. Таким образом мы можем иметь множество постановок задач с мощностью, исчисляемой выражением:

$$\text{card}(sPDef) = (2^{OP\_NUM} - 1) * (2^{INTERVALS\_NUM} - INTERVALS\_NUM - 1)$$

Для приведенного выше примера деклараций данных OP_NUM = 4 и INTERVALS_NUM = 4, что дает card(*sPDef)* = 15 * 11 = 165.

Базовый прототип предполагает укладку плитки на наклонной поверхности, что предопределяет неодинаковую нагрузку на ходовую часть платформы и механизмы монтажа плитки в различных состояниях – движение вверх, вниз, влево, вправо. Это означает, что операция OP_MOVE может декомпозироваться на 4 операции: OP_UP_MOVE, OP_DOWN_MOVE, OP_LEFT_MOVE, OP_RIGHT_MOVE. Учитывая различные направления вектора скатывающей силы в разных состояниях платформы, можно подвергнуть декомпозиции также операции OP_SETTILE и OP_ROTATE. Все три рассмотренные декомпозиции породят 12 операций и с учетом 13-й операции OP_INIT получаем:

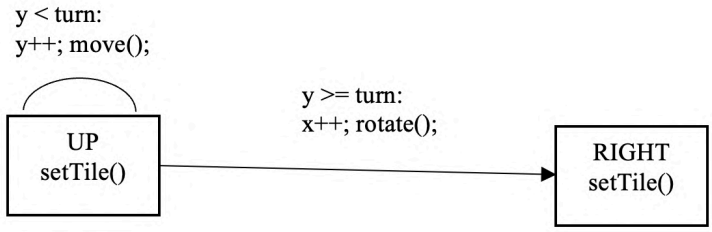
$$\text{card}(sPDef) = (2^{13} - 1) * (2^4 - 5) = 8191 * 11 = 90\,101.$$

В потенциально возможном множестве постановок задач существует много подмножеств постановок, эквивалентных по числу требуемых изменений. Увеличение разнообразия по числу изменений возможно за счет определения различных наборов правил реагирования на попадание задержки времени операции платформы в тот или иной интервал для разных состояний СЛУ базового прототипа. Можно также увеличить количество интервалов граничных значений времени выполнения операций.

Многообразие вариантов постановок задач весьма полезно при организации учебных занятий по дисциплинам, где онтологическое моделирование является инструментом в учебно-исследовательских проектах. Большое множество *sPDef* позволяет выдавать различающиеся по содержанию и сложности индивидуальные задания по модификации преподавательского прототипа студентам нескольких потоков и даже разных лет обучения.

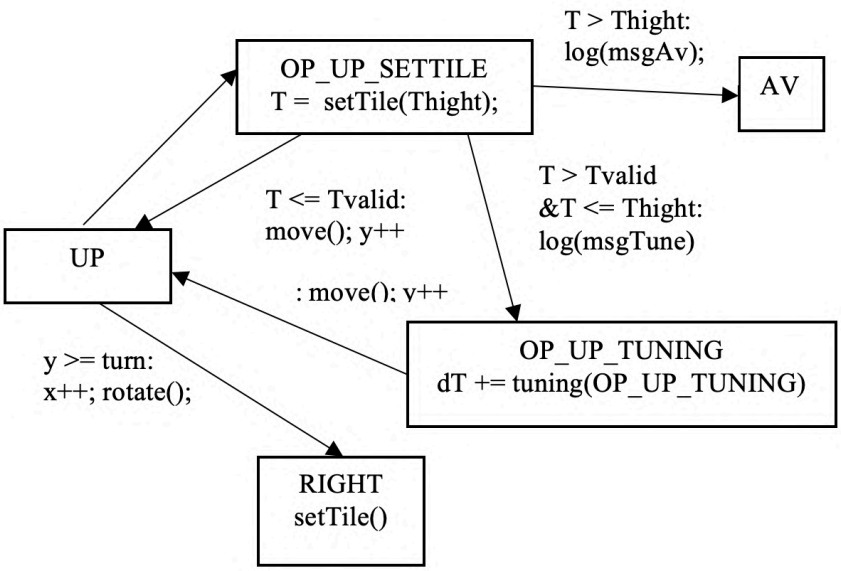
Для оценки сложности изменения проектных решений необходимо иметь представление о характере этих изменений. Поскольку в разработке базового прототипа используется автоматное программирование, характер и сложность изменений легко оценить через анализ изменений, вносимых в диаграмму состояний. Рассмотрим один из возможных вариантов изменений для случая, когда в состоянии UP требуется выполнить операцию OP_UP_SETTILE.

В базовом прототипе UP в автоматной диаграмме представляется так:



В целевом прототипе вводятся новые состояния, позволяющие следить за задержками времени и организовывать переход в соответствующие состояния, где совершается настройка. Кроме того, в протокол дописываются сообщения, характеризующие переходы к операциям настройки, а также организуется учет чистого времени работы плиткоукладчика, не учитывающего время, потраченное на наладку. При этом формируется так называемое «композитное состояние» UP, которое включает в себя несколько подсостояний, обслуживающих установку одной плитки при движении вверх.

Для простоты рассмотрим вариант, когда в постановке задачи фигурируют только изменения для операции OP_UP_SETTILE предусмотрены две границы задержки: Tvalid и Thight. В результате получаем фрагмент диаграммы одним состоянием настройки:



В приведенной на рисунке диаграмме функция запуска монтажа плитки setTile снабжается значением тайм-аута, в качестве которого выступает значение предельно допустимой задержки процесса исполнения операции OP_UP_SETTILE. Эта же функция возвращает реально потраченное время на монтаж, что позволяет СЛУ принимать решений о переходе в аварийное состояние, в состояние наладки или состояния продолжения движения вверх. Функция инициирования настройки tuning возвращает затраты времени, которые подсуммируются к общим затратам на тюнинг, чтобы можно было это время не учитывать в протоколе.

Если изменение требований охватит операции OP_UP_MOVE и OP_UP_ROTATE, то композитное состояние UP при охвате нескольких границ задержек времени может иметь более десятка подсостоиний.

Проектный процесс (подпроцесс)	**Проектная операция без применения онтологического сопровождения**	**Проектная операция с применением онтологического сопровождения на основе DT-подхода**	**Трудо-затраты без ОМ (в SP)**	**Трудо-затраты с ОМ (в SP)**	**Множитель**	**Комментарий**
Сбор фактов. Исследование объекта автоматизации	Разработка способа организации результатов исследования и выбор способа их представления	Результаты исследования представляются в форме вопросно-ответных протоколов	21	21		
Формирование требований пользователя АС	Описание требования к логике функционирования системы	Представление требования к системе в онтологической модели требований, строящейся с учетом результатов исследования объекта автоматизации	1	2	Количество требований	Требуются дополнительные трудозатраты на описание сущностей онтологической модели
Разработка концепции АС	–	Специфицирование прототипа N с ориентацией на онтологическое моделирование	0	1	Количество прототипов	

	Исследование возможности задействования прототипа N в разработке АС	Сопоставление спецификаций прототипа N с онтологической моделью требований для определения возможности его задействования в разработке АС	2	0,5	Количество прототипов	Снижаются трудозатраты на анализ особенностей прототипа, поскольку его сущности формализованы в онтологических спецификациях; сопоставление спецификаций может быть автоматизировано.
	Описание элемента концепции АС в соответствии с требованием N	Представление элемента концепции АС в онтологии проектных решений	1	2	Количество требований	Требуются дополнительные трудозатраты на описание сущностей онтологической модели
Разработка предварительных проектных решений	Разработка UML-диаграммы	Генерирование UML-диаграммы на основе онтологии проектных решений	13	0	Количество UML-диаграмм	UML генерируется автоматически
Разработка программ	–	Представление сущности онтологии проектной решении на уровне реализации (формирование онтологии реализации)	0	1	Количество сущностей ОМ на уровне реализации	

	Модификация прототипа N	Автоматическая модификация прототипа N за счет онтологических спецификаций с единым пространством имен; доработка прототипа (при необходимости)	8	2	Количество прототипов	В ряде случаев требуется только замена имен, что может быть полностью автоматизировано за счет ОМ; в ряде случае требуются дополнительные модификации
	Разработка программы, реализующей модуль или функцию АС	Генерирование программы на основе онтологии реализации; доработка сгенерированного кода (при необходимости)	13	5	Количество функций программы	Процент генерируемого кода варьируется в зависимости от особенностей системы
Внесение изменений в требования	Внесение изменений в требование N	Внесение изменений в онтологию требований	1	1	Количество измененных или новых требований	
Доработка системы при возникновении необходимости изменить требования	Отрисовка модифицированной UML-диаграммы	Генерирование новой UML-диаграммы на основе модифицированной онтологии проектных решений	8	0	Количество измененных или новых UML-диаграмм	При условии, что сгенерированные в первой итерации проектные решения уже были доработаны, доработки минимальны

	Внесение изменений в исходный код программы, реализующей модуль или функцию АС	Генерирование новой программы на основе онтологии реализации; доработка сгенерированного кода (при необходимости)	8	2	Количество измененных или новых функций программы	При условии, что сгенерированный в первой итерации код уже были доработан и встроен в систему, доработки минимальны

Приложение 5. Анализ трудозатрат на онтологическое моделирование АС

Проектная процедура	**Автоматизация**	**Трудозатраты (без средств автоматизации), SP**	**Трудозатраты (со средствами автоматизации), SP**	**Множитель**
Изучение всех проектных документов и других описаний объекта автоматизации	Частично подвергнуто автоматизации: формирование списка претендентов на объекты (автоматически), после которого объем текстовой информации сокращается в 10 раз. Одно понятие встречается в тексте на каждые 70 слов.	1	0,1	Количество слов / 70
Выделение объектов, участвующих в онтологической модели	–	1	1	Количество объектов
Идентификация объектов (выбор подходящих идентификаторов или имен)	Полностью автоматизировано	0,5	0	Количество объектов
Описание классов и агрегатов	Классы и агрегаты заданы разработанной метамоделью онтологии	1	0	Количество классов или агрегатов
Классификация или агрегация объектов	–	0,5	0,5	Количество объектов

Описание свойств объектов	–	1	1	Количество объектов
Описание связей между объектами	Частично подвергнуто автоматизации: - построение предположений о наличии связей между объектами с указанием возможного типа связи на основании их текстовых описаний; - на уровне метамодели задан факт наличия связи между объектами определенных типов, что снижает трудозатраты по определению связей примерно на 50% (усилия складываются из выявления факта наличия связи и выявления его семантического типа).	1	0,5	Количество связей
Описание правил логического вывода, обеспечивающих непротиворечивость онтологической модели и автоматизацию модификаций	Большая часть правил логического вывода (около 80%) заданы на уровне метамодели.	8	1,6	Количество правил
Наполнение хранилища онтологии проекта выявленными сущностями	Полностью автоматизировано.	0,5	0	Количество объектов + Количество связей
Отслеживание изменений в проекте для внесения изменений в онтологическую модель	Частично автоматизировано за счет реализации правил логического вывода. Автоматически модифицируется примерно 40% сущностей.	1	0,6	Количество новых или измененных сущностей

Приложение 6. Свидетельства о регистрации программ для ЭВМ и программно-информационных продуктов

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2017664036

Программа семантического контроля текстов постановок проектных задач

Правообладатель: ***федеральное государственное бюджетное образовательное учреждение высшего образования «Ульяновский государственный технический университет» (RU)***

Авторы: ***Соснин Петр Иванович (RU),***
Куликова Анна Александровна (RU)

Заявка № **2017660939**
Дата поступления **27 октября 2017 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***14 декабря 2017 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2018661247

Программа извлечения из текста связанных концептов типа «часть-целое»

Правообладатель: ***федеральное государственное бюджетное образовательное учреждение высшего образования «Ульяновский государственный технический университет» (RU)***

Авторы: ***Соснин Петр Иванович (RU),***
Куликова Анна Александровна (RU)

Заявка № **2018618215**
Дата поступления **02 августа 2018 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***04 сентября 2018 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2021665410

Программная система онтологического сопровождения проектного процесса по разработке систем логического управления

Правообладатель: ***федеральное государственное бюджетное образовательное учреждение высшего образования «Ульяновский государственный технический университет» (RU)***

Авторы: ***Негода Виктор Николаевич (RU), Куликова Анна Александровна (RU)***

Заявка № **2021664648**

Дата поступления **20 сентября 2021 г.**

Дата государственной регистрации в Реестре программ для ЭВМ ***24 сентября 2021 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

ОФАП УОЦ НИТ

Ульяновский государственный технический университет
Ульяновский областной центр новых информационных технологий
Областной фонд алгоритмов и программ

Свидетельство № 1491

о регистрации программно-информационного продукта

Наименование:	***Онтологическое моделирование в проектировании автоматизированных систем***
Тип:	*Методическое пособие*
URI:	***http://ofap.ulstu.ru/1491***
Авторы:	*Куликова Анна Александровна, Негода Виктор Николаевич – Кафедра «Вычислительная техника» (УлГТУ/ФИСТ)*
Дата регистрации:	*23 сентября 2021 г.*

Директор Ульяновского областного центра НИТ — *К.В. Святов*
Администратор базы данных ОФАП — *Ю.А. Лапшов*

Приложение 7. Акты внедрения

УТВЕРЖДАЮ
Генеральный директор
ФНПЦ АО «НПО «Марс», к.т.н.

В.А. Маклаев
23.09.2021

АКТ

об использовании результатов кандидатской диссертации А.А. Куликовой «Методы и средства формирования и использования онтологий проектов в процессе проектирования автоматизированных систем»

Научно-техническая комиссия в составе:

Председатель комиссии:

Главного специалиста, к.т.н. Э.Д. Павлыгина,

Члены комиссии:

Заместителя начальника комплексного научно-исследовательского отделения 2, к.т.н, А.И. Моисеев;

Начальника отдела развития и поддержания интегрированной автоматизированной системы управления предприятием, к.т.н. А.А. Перцева.

Настоящим актом подтверждается использование следующих научных и практических результатов диссертационной работы А.А. Куликовой «Методы и средства формирования и использования онтологий проектов в процессе проектирования автоматизированных систем», а именно:

1. Средства формирования системы онтологий проекта, охватывающей требования к автоматизированной системе, предварительные проектные решения и ее реализацию, поддерживающей единство пространств имен, задействованных на различных этапах проектного процесса и способствующие сокращению семантического разрыва между стадиями проектирования, а также достижению концептуальной целостности разрабатываемых артефактов проектирования.

2. Средства трансформации онтологических спецификаций проекта за счет реализации правил логического вывода, связывающих онтологию требований с онтологиями проектирования и реализации.

3. Средства использования системы онтологий проекта для генерации проектных решений, в том числе UML-диаграмм и исходного кода программ, способствующие повышению производительности труда проектировщика, в том числе в условиях изменения требований и при возникновении необходимости в перепроектировании объекта автоматизации.

Указанные научно-технические результаты использованы в рамках работ по НИР «Осознание», ОКР «Формирование системы обращения со знаниями», ОКР «Разработка и формирование базы опыта проектной организации в составе интегрированной автоматизированной системы управления предприятием».

Работа обладает научно-технической новизной и может быть использована в проектных организациях, разрабатывающих сложные автоматизированные системы, для рационализации работ по управлению изменениями, облегчения взаимодействия между проектными командами, задействованными на различных стадиях проектного процесса.

Председатель комиссии:
Главный специалист, к.т.н. Э.Д. Павлыгин

Члены комиссии:
Заместитель начальника комплексного
научно-исследовательского отделения 2, к.т.н. А.И. Моисеев

Начальник отдела развития и
поддержания интегрированной
автоматизированной системы
управления предприятием, к.т.н. А.А. Перцев

УТВЕРЖДАЮ

Первый проректор, проректор
по учебной работе УлГТУ

_____ Е.В.Суркова

27 09 2021 г.

АКТ

о внедрении в учебный процесс

результатов диссертационной работы Куликовой А.А.

Результаты диссертационной работы А.А. Куликовой «Методы и средства формирования и использования онтологий проектов в процессе проектирования автоматизированных систем», представленной на соискание степени кандидата технических наук, внедрены в учебный процесс Ульяновского государственного технического университета при обучении студентов бакалавриата направления 09.03.01 «Информатика и вычислительная техника» и студентов магистратуры направления 09.04.01 «Информатика и вычислительная техника.

Методика автоматизированного проектирования, разработанная А.А.Куликовой, использована в 2020/2021 учебном году в двух учебных курсах магистратуры «Логическое управление в автоматизированных системах» и «Формализация в проектировании автоматизированных систем». В настоящее время методика используется в дисциплине магистратуры «Аналитическое моделирование в проектировании автоматизированных систем» и дисциплине бакалавриата «Основы теории систем». Методические материалы, разработанные А.А.Куликовой по результатам своих диссертационных исследований, используются в лекционных и практических занятиях. Эти методические материалы зарегистрированы в областном фонде алгоритмов и программ Ульяновского областного центра новых информационных технологий (свидетельство № 1491) и опубликованы в электронном виде на сайте фонда.

Заведующий кафедрой «Вычислительная техника»

ФГБОУ ВО «Ульяновский государственный технический университет»

к.т.н., доцент К.В.Святов