Федеральное государственное бюджетное учреждение науки
Институт динамики систем и теории управления имени В.М. Матросова
Сибирского отделения Российской академии наук

На правах рукописи

Дородных Никита Олегович

Метод и программное средство разработки баз знаний на основе трансформации концептуальных моделей

05.13.11 – Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

ДИССЕРТАЦИЯ
на соискание ученой степени
кандидата технических наук

Научный руководитель
к. т. н., доцент
Юрин Александр Юрьевич

Иркутск – 2017

Оглавление

Введение

Актуальность темы. В настоящее время знание является стратегическим ресурсом, при этом «оцифровка» знаний и их представление в виде концептуальных моделей, декларативных программ и кодов баз знаний (БЗ) обеспечивает их эффективное использование. БЗ является основным компонентом систем искусственного интеллекта и сдерживающим фактором их широкого применения, так как разработка БЗ является одним из самых сложных и трудоемких этапов при создании интеллектуальных систем – «узким местом» проектирования систем подобного вида [1-11]. На данном этапе решаются задачи моделирования предметной области, получения, концептуализации и формализации знаний с их описанием на определенном языке представления знаний (ЯПЗ) [3, 4, 7]. Актуальность разработки новых методов и средств, повышающих эффективность процессов обработки знаний, в том числе при решении практических слабоформализованных задач в различных предметных областях, остается высокой [1-4].

Одним из способов повышения эффективности процесса разработки БЗ является применение методов получения знаний из различных источников, в том числе концептуальных моделей, под которыми понимаются модели, представленные множеством понятий и связей между ними определяющих смысловую структуру рассматриваемой предметной области вместе со свойствами и характеристиками, классификацией этих понятий по типам, ситуациям, признакам в данной области и законов протекания процессов в ней [12]. При этом особый интерес представляет использование моделей, построенных с использованием программных средств концептуального, когнитивного, онтологического моделирования и CASE-средств путем их трансформации в программные коды.

Значительный вклад в разработку и исследование моделей, методов и средств создания интеллектуальных систем (включая онтологии, БЗ и

программные средства для их проектирования и синтеза программных кодов) внесли Аверкин А.Н., Баадер Ф., Берман А.Ф., Вагин В.Н., Ван Хармелен Ф., Варшавский П.Р., Гаврилова Т.А., Голенков В.В., Грау Б., Грибова В.В., Грубер Т., Гуарино Н., Джарратано Дж., Джексон П., Еремеев А.П., Загорулько Ю.А., Клещев А.С., Колесников А.В., Кудрявцев Д.В., Ленат Д., Люгер Г., Массель Л.В., МакГиннесс Д., Мотик Б., Норвиг П., Осипов Г.С., Осуга С., Патель-Шнайдер П., Попов Э.В., Поспелов Д.А., Райли Г., Рассел С., Рыбина Г.В., Саэки Ю., Сова Дж., Стааб С., Финн В.К., Фоминых И.Б., Хорошевский В.Ф., Хоррокс Я., Шрайбер Г., Штудер Р., Частиков А.П. и др. В области автоматизации создания программных систем и их компонентов, разработки трансляторов, а также подходов трансформации моделей и программ можно отметить работы исследователей Ахо А., Гасевика Д., Горбунова-Посадова М.М., Гринфилда Дж., Джоолта Ф., Ершова А.П., Клеппе А., Кука С., Менса Т., Опарина Г.А., Сабельфельда В.К., Ульмана Дж., Фаулера М., Франкеля Д., Чарнецки К. и др.

Однако, существующие методы и системы автоматизированного создания БЗ на основе концептуальных моделей обладают рядом недостатков, в частности сложностью описания самих моделей для генерации кода; высокими квалификационными требованиями к пользователю; отсутствием возможности совместной распределенной и одновременной работы пользователей; отсутствием или ограниченностью генерации программного кода БЗ на различных ЯПЗ (частичное преобразование, скелетные коды). Это определяет актуальность создания новых моделей, методов и средств, обеспечивающих разработку БЗ, в том числе на основе трансформации концептуальных моделей. В свою очередь, существование множества форматов концептуальных моделей требует создания средства, которое будет обладать свойством расширяемости в части создания дополнительных модулей в форме программных компонентов трансформации моделей.

Целью диссертационного исследования является разработка метода автоматизации проектирования и синтеза программных кодов БЗ в форме декларативных программ на основе трансформации концептуальных моделей и

его программная реализация в виде инструментального средства для повышения эффективности обработки знаний.

Задачи исследования:

1. Выполнить анализ существующих подходов, методов и программных средств автоматизации создания интеллектуальных систем и систем разработки БЗ, в том числе в форме онтологий, обеспечивающих трансформацию концептуальных моделей в программные коды БЗ.
2. Разработать метод автоматизации процесса проектирования и создания программных компонентов интеллектуальных систем, обеспечивающих синтез кода БЗ на основе трансформации концептуальных моделей.
3. Создать предметно-ориентированный декларативный язык для описания трансформаций.
4. Разработать и апробировать инструментальное программное средство, реализующее предлагаемые метод и язык.
5. Разработать методику создания БЗ на основе трансформации концептуальных моделей с использованием разработанного инструментального средства и оценить ее эффективность.

Объектом исследования являются алгоритмическое и программное обеспечение создания программных средств обработки знаний в вычислительных машинах, комплексах и компьютерных сетях.

Предметом исследования являются модели, методы и алгоритмы проектирования и синтеза программных кодов БЗ на основе трансформации концептуальных моделей.

Методы исследования. В работе использовались методы объектно-ориентированного программирования, трансформации моделей, построения трансляторов и предметно-ориентированных языков, а также методы и средства искусственного интеллекта и онтологического моделирования.

Научная новизна:

1. Впервые предложен специализированный метод автоматизации процесса создания программных компонентов интеллектуальных систем для

проектирования БЗ и синтеза их кода на основе трансформации концептуальных моделей, отличием которого от известных является использование языка описания трансформаций и оригинальной модели типового программного компонента.

2. Разработан новый предметно-ориентированный декларативный язык описания трансформаций (TMRL), включающий конструкции для описания не только преобразуемых структур и связей между ними, но и механизма взаимодействия с внешними программными компонентами трансформаций. Это позволяет абстрагироваться от конкретики специализированных языков трансформации моделей общего назначения и использовать созданные ранее компоненты трансформации.
3. На основе предложенного метода разработано инструментальное программное средство, позволяющее интерактивно создавать компоненты трансформации концептуальных моделей, а также проектировать с их помощью БЗ.
4. Создана оригинальная методика автоматизированной разработки БЗ, отличием которой от известных является использование концептуальных моделей в качестве исходных данных и специализированных программных компонентов и языков (TMRL и RVML) в качестве инструментальных средств.

В целом в диссертации предложены новые модели, методы и средства разработки БЗ на основе трансформации концептуальных моделей, позволяющие значительно сократить сроки и стоимость разработки, а также снизить требования к квалификации разработчика в части знания языков программирования и модельных трансформаций за счет применения интерактивного инструментального средства создания компонентов трансформаций.

Практическая значимость результатов. Основные научные результаты по теме диссертации получены в рамках проекта СО РАН IV.36.1.2, проектов РФФИ 15-37-20655, 15-07-03088, 15-07-05641, 16-37-00122 (рук.), а также соглашения №

8770 ФЦП «Научные и научно-педагогические кадры инновационной России» на 2009-2013 гг.

Предложенные в рамках диссертационной работы метод и веб-ориентированное инструментальное средство позволяют снизить трудозатраты и сократить сроки разработки программных компонентов создания БЗ интеллектуальных систем. Практическая значимость результатов подтверждена полученными актами внедрения, а также их использованием в учебном процессе ИрНИТУ в рамках курсов «CASE-средства» и «Инструментальные средства информационных систем» при проектировании БЗ, и при выполнении работ по хозяйственному договору с АО «ИркутскНИИхиммаш».

Достоверность результатов проведенных исследований подтверждается обоснованным использованием методов и технологий трансформации моделей, опубликованных в открытой печати, публикацией полученных результатов, работоспособностью разработанного инструментария, решением тестовых и прикладных задач.

Соответствие диссертации паспорту научной специальности. В соответствие с паспортом специальности 05.13.11, диссертация охватывает решение задач создания программных средств различного назначения (в частности, систем проектирования и синтеза БЗ), включает исследование языков и систем программирования (построение предметно-ориентированного языка трансформации концептуальных моделей в код БЗ и поддерживающего его программной системы), а также моделей, методов, алгоритмов и программных инструментов для организации взаимодействия программ и программных систем. Отражённые в диссертационной работе положения соответствуют пунктам 1, 2 и 3 области исследования специальности 05.13.11.

Научное и народнохозяйственное значение диссертации заключается в повышении эффективности процессов обработки и передачи знаний (автоматизированная разработка БЗ) в вычислительных машинах, комплексах и компьютерных сетях.

Апробация работы. Основные результаты диссертационной работы, её отдельные положения, а также результаты конкретных прикладных исследований и разработок обсуждались на научных семинарах ИДСТУ СО РАН и на следующих международных, всероссийских, региональных научных и научно-практических конференциях: XII Международный форум управления знаниями «International Forum on Knowledge Asset Dynamics. IFKAD-2017» (г. Санкт-Петербург, 2017 г.); II, III Российско-монгольские конференции молодых ученых по математическому моделированию, вычислительно-информационным технологиям и управлению (Россия, г. Иркутск – Монголия, п. Ханх, 2013, 2015 гг.); Международная научно-практическая конференция «Фундаментальная информатика, информационные технологии и системы управления: реалии и перспективы. FIITM-2014» (г. Красноярск, 2014 г.); XLIV Международная конференция «Информационные технологии в науке, образовании и управлении. IT + S&E`15» (г. Гурзуф, 2015 г.); Пятнадцатая национальная конференция по искусственному интеллекту с международным участием. КИИ-2016 (г. Смоленск, 2016 г.); VI, VII Международные конференции «Системный анализ и информационные технологии. САИТ» (г. Светлогорск, 2015, 2017 гг.); VI, VII Международные научно-технические конференции «Открытые семантические технологии проектирования интеллектуальных систем. OSTIS» (Беларусь, г. Минск, 2016, 2017 гг.); XV, XVI Всероссийские конференции молодых ученых по математическому моделированию и информационным технологиям (г. Тюмень, г. Красноярск, 2014, 2015 гг.); XXI, XXII Байкальские Всероссийские конференции с международным участием и Школы-семинары научной молодежи «Информационные и математические технологии в науке и управлении» (г. Иркутск, 2016, 2017 гг.); Конференции «Ляпуновские чтения» (г. Иркутск, 2014, 2015, 2016 гг.).

Публикации и личный вклад автора. Результаты диссертационного исследования опубликованы в 30 печатных работах, в том числе 4 статьи в рецензируемых журналах, рекомендованных ВАК для опубликования результатов диссертаций, 1 статья в рецензируемом журнале, индексируемом в Web of Science

и Scopus, 1 коллективная монография, 22 публикации в трудах международных и всероссийских конференций, 2 свидетельства о государственной регистрации программ для ЭВМ. Результаты главы 1 и 2 опубликованы в работах [14-22, 24-26, 28, 30, 32, 33, 35, 36], результаты главы 3 и 4 опубликованы в работах [23, 27, 29, 31, 34, 37-43].

Все выносимые на защиту научные положения получены соискателем лично. В основных научных работах по теме диссертации, опубликованных в соавторстве, лично соискателем разработаны: в [14-20] – модели и метод создания программных компонентов трансформаций; [21-38] – методическое и программное обеспечение автоматизированного создания БЗ на основе трансформации концептуальных моделей. В [39-43] соискателем проведена апробация разработанного метода и программного средства, включая результаты оценки их эффективности.

Структура и объем диссертации. Диссертационная работа состоит из введения, четырех глав, заключения, списка литературы, включающего 270 наименований, и 11 приложений. Объем составляет 150 страниц основного текста, включая 58 рисунков, 9 таблиц.

Содержание работы. Во *введении* обосновывается актуальность диссертационной работы, формулируется цель и задачи исследования, определяется научная новизна и практическая значимость результатов.

В *первой главе* дается анализ современных подходов к созданию программных компонентов интеллектуальных систем и БЗ, а также подходов к трансформации моделей.

Во *второй главе* приводится описание моделей и метода создания программных компонентов трансформации, предметно-ориентированного языка представления и хранения моделей трансформаций и методики автоматизированной разработки БЗ на основе трансформации концептуальных моделей.

В *третьей главе* приводится описание инструментального программного средства, реализующего предлагаемый метод и язык.

В *четвертой главе* показаны особенности применения разработанного программного средства и компонентов трансформации концептуальных моделей в БЗ, а также приведена оценка их эффективности.

В *заключении* сформулированы основные научные результаты диссертационной работы.

Глава 1. Аналитический обзор

В настоящее время разработка новых методов и подходов к созданию интеллектуальных систем и их программных компонентов остается перспективной областью научных исследований. Чаще всего интеллектуальные системы применяются для решения сложных задач, основная сложность решения которых связана с использованием слабоформализованных знаний специалистов-практиков, где смысловая обработка информации превалирует над вычислительной [3]. Например, поддержка принятия решения в сложных ситуациях, прогнозирование возможных и определение фактических причин повреждения и разрушения машин и конструкций, диагностика механических систем, анализ рисков, постановка диагноза и рекомендации по методам лечения, анализ визуальной информации, управление диспетчерскими пультами и т.д.

Сложность и трудоемкость процесса разработки интеллектуальных систем, в частности, экспертных систем (ЭС), обусловлена, главным образом, сложностью и трудоемкостью разработки БЗ, включая задачи по формализации предметных знаний и их описание на определенном ЯПЗ [1-11]. Именно эти вопросы исследует инженерия знаний (knowledge engineering).

Инженерия знаний – это одно из направлений искусственного интеллекта, изучающая модели и методы получения (извлечения), структурирования (концептуализации) и формализации (представления) знаний для их обработки в интеллектуальных и информационных системах [2, 3, 11]. При этом под знаниями понимаются закономерности предметной области (принципы, связи, законы), полученные в результате практической деятельности и профессионального опыта, позволяющие специалистам ставить и решать задачи в этой области [3].

Традиционно разработку и прототипирование ЭС и систем, основанных на знаниях, разделяют на шесть основных стадий [3, 7]. При этом выделяют три фазы создания БЗ [3]:

1. Получение знаний – процесс добычи (формирования) знаний из различных источников: экспертов, баз данных, Интернет-ресурсов, концептуальных

моделей, документации, специальной литературы и др. Этот процесс в специальной литературе получил несколько названий:

- извлечение (elicitation);
- приобретение (acquisition);
- добыча (knowledge capture);
- выявление (discovery) знаний и др.

Результатом данной стадии является огромное количество гетерогенных (разнохарактерных) противоречивых фрагментов знаний. Средняя продолжительность этапа составляет 1-3 месяца.

2. Структурирование или концептуализация знаний – это разработка неформального описания знаний о предметной области в виде графа, таблицы, диаграммы или текста, которое отражает основные концепции и взаимосвязи между понятиями предметной области. Такое слабоформализованное описание часто называют полем знаний. Средняя продолжительность этапа составляет 2-4 недели.
3. Формализация (представление или кодификация – программирование БЗ) знаний – формализованное представление поля знаний на основе выбранного специализированного языка представления знаний (ЯПЗ). Результатом данной стадии является разработанная БЗ на ЯПЗ, который, с одной стороны, соответствует структуре поля знаний, а с другой - позволяет реализовать прототип ЭС на следующей стадии программной реализации. Средняя продолжительность этапа составляет 1-2 месяца.

Стадии процесса разработки ЭС представлены на Рисунке 1.

Данные стадии создания БЗ традиционно считают «узким местом» при разработке интеллектуальных систем, в частности, ЭС. Сложность решения задач данных стадий возрастает при необходимости обеспечения удаленного и совместного (коллективного) доступа, согласования мнений экспертов (специалистов-предметников), инженеров по знаниям и аналитиков.

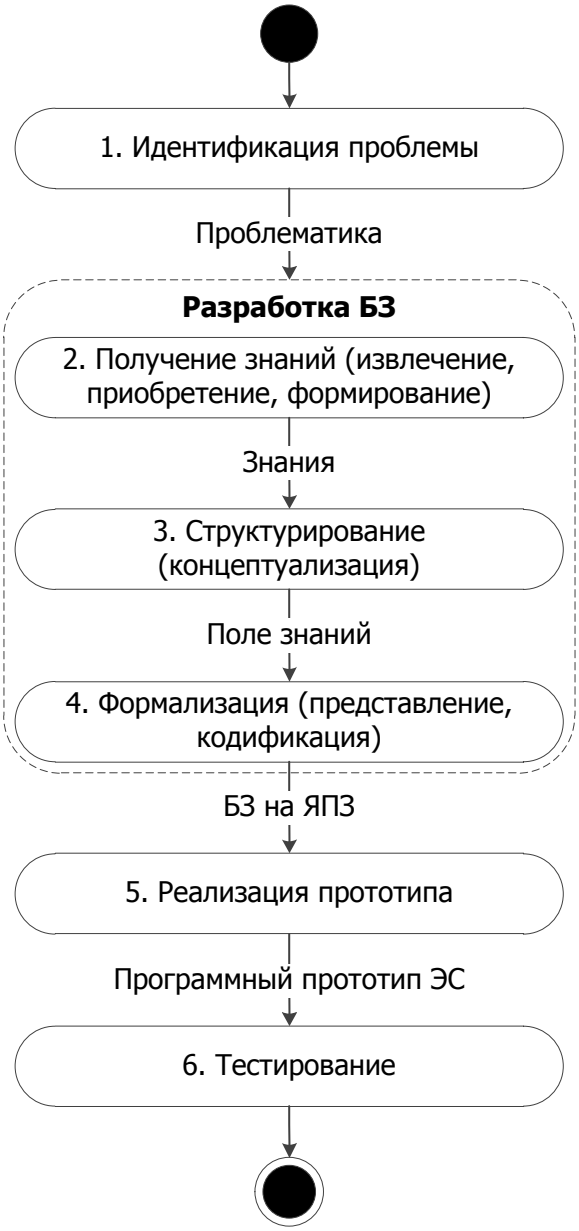

Рисунок 1. Стадии процесса разработки прототипа ЭС

Тем не менее, в последние годы наметилась тенденция автоматизации перечисленных стадий и использования принципов визуального (когнитивного) моделирования при разработке БЗ, а также повторного использования накопленной информации, представленной в разных формах, в том числе, в форме концептуальных моделей (например, диаграмм, схем и др.). Так на стадиях получения и структурирования знаний активно используются различные концептуальные (информационные) модели, в том числе, концепт-карты (concept maps) или интеллект-карты/карты-знаний (mind maps) и др. В частности, эта

тенденция подтверждается примерами [44-47]. При этом когнитивное моделирование, основанное на использовании когнитивных карт с причинно-следственными связями, активно развивается как научное направление в России: в ИПУ РАН (Трахтенгерц Э.А., Абрамова Н.А., Кулинич А.А. и др.), в УГАТУ (научная школа Юсуповой Н.И., в ИСЭМ СО РАН (научная школа Массель Л.В.) и за рубежом (например, научная школа проф. Грумпоса П., ун-т г. Патры) и др.

Концептуальные модели предметных областей создаются как в процессе разработки программного обеспечения, так и в процессе целенаправленного моделирования предметной области (построения ее онтологической модели). При создании данных концептуальных моделей используют различные нотации, языки и стандарты, в частности: DFD, IDEF0, IDEF5, UML и др. Графические нотации данных языков в первую очередь ориентированы на системных аналитиков, программистов, инженеров по знаниям и т.д. Если говорить о специалистах той или иной предметной области (специалистах не в области проектирования программных систем), то они предпочитают модели, имеющие общесистемную направленность и ориентированные на систематизацию знаний или поддержку принятия решений (например, концептуальные карты или интеллект-карты, диаграммы Венна и Исикавы, семантические модели и т.д.). К таким моделям также можно отнести «деревья отказов» и «деревья событий», применяемые в области анализа отказов и риска технических систем (fault tree analysis, event tree analysis). Результаты моделирования в данных, можно сказать, «предметных» нотациях, с одной стороны, содержат специальные знания, которые необходимо использовать для автоматизации решения предметных задач, с другой стороны, являются удобным и понятным для специалиста-предметника способом представления знаний. Создают концептуальные модели при помощи различных программных средств (специальные редакторы, системы концептуального, когнитивного моделирования, CASE-средства и т.п.). Данные системы позволяют разрабатывать графические (визуальные) модели разной степени абстракции, соответствующие знаниям эксперта предметной области и генерировать различную отчетную документацию. Однако большинство из этих систем не

охватывают все стадии создания БЗ и не обеспечивают полноты процесса разработки от моделей предметной области до программных кодов на ЯПЗ, что в некоторых случаях позволяет получать только графические артефакты (изображения) структур БЗ. Таким образом, это затрудняет практическое использование построенных моделей для создания БЗ при разработке интеллектуальных систем.

В связи с этим актуально создание расширяемого инструментального программного средства, обеспечивающего не только синтез (генерацию) кода БЗ на определенном ЯПЗ путем преобразования элементов концептуальных моделей, но и совместное (коллективное) проектирование БЗ с использованием специальных средств визуализации данного процесса. При этом под расширяемостью понимается возможность создания и включение в состав инструментального средства новых программных компонентов, реализующих функции анализа концептуальных моделей и генерации кода БЗ для других форматов (стандартов) концептуальных моделей и ЯПЗ.

Таким образом, целью данной работы является разработка метода автоматизации создания программных компонентов интеллектуальных систем, предназначенных для формирования БЗ на основе трансформации концептуальных моделей, и его реализация в виде инструментального средства для повышения эффективности обработки знаний.

1.1. Разработка баз знаний и компонентов интеллектуальных систем

Активная информатизация всех сфер жизни общества стимулирует создание программных инструментальных средств, используемых для разработки прикладных программ в различных областях. Разновидностью прикладных программ являются ЭС, моделирующие процесс рассуждения эксперта при принятии им решений.

Современные ЭС – это сложные программные комплексы, аккумулирующие знания специалистов в конкретных предметных областях и распространяющие этот эмпирический опыт для консультаций менее квалифицированных пользователей. Разработка ЭС как активно развивающаяся ветвь информатики направлена на компьютерную обработку информации в тех областях науки и техники, где малопригодны традиционные математические методы моделирования. В этих областях важны смысловая и логическая обработка информации, опыт экспертов [3].

На сегодняшний день создано множество как статических, так и динамических ЭС в различных областях человеческого знания, в частности:

- в инженерии, например, для выявления причин неисправности технических систем; выбора конструкционных материалов при построении сложных технических устройств; управления сложными уникальными системами и т.д.;
- в экономике и банковской сфере, например, для оценки финансового состояния предприятия; управления торгами; обнаружения подозрительных операций/транзакций, в частности, выявления подозрительных сделок в рамках деятельности по борьбе с отмыванием денег и т.д.;
- в медицине, например, для телемониторинга сердечной недостаточности; диагностики риска гипертонии и т.д.;
- для прогнозирования чрезвычайных ситуаций и др.

Также остается наиболее популярным использование различных диагностических ЭС [48].

Главное отличие ЭС от других программных средств – это наличие БЗ, представленной множеством систематизированных знаний, описывающих закономерности какой-либо предметной области. При создании БЗ решаются задачи моделирования предметной области, получения, концептуализации (структурирования), формализации (представления) знаний на определенном ЯПЗ. При этом в большинстве случаев разработчик должен либо иметь навыки

программирования и знать какой-либо ЯПЗ, либо в совершенстве владеть каким-либо инструментарием.

В области разработки интеллектуальных систем существует ряд крупных методологий, направленных на формализованное описание повторяющихся схем решений задач на абстрактном уровне без привязки к какой-либо конкретной предметной области. Так самой популярной методологией инженерии знаний является CommonKADS [49], которая де-факто считается стандартом проектирования и разработки систем, основанных на знаниях. Также можно отметить такие методологии (проекты), как MIKE [50] и MOKA [51].

В настоящее время можно выделить несколько основных направлений к повышению эффективности создания БЗ и ЭС:

1. Применение систем онтологического и когнитивного моделирования, CASE-средств (Protégé, OntoStudio, OntoEdit, WebODE, CmapTools, Mindjet MindManager, FreeMind, ProTheBrain, XMind, IBM Rational Rose, Poseidon for UML, StarUML и др.), которые позволяют создать графические модели, соответствующие ключевым абстракциям некоторой предметной области или программного обеспечения. Однако большинство из подобных систем не охватывают все этапы создания БЗ и ЭС и не обеспечивают комплексность процесса разработки от модели предметной области до программного кода, в некоторых случаях позволяя получить лишь графические изображения структур БЗ. За исключением Protégé [52], который позволяет синтезировать ограниченный набор элементов БЗ, в частности на CLIPS/COOL и JESS [53].
2. Применение оболочек ЭС, которые получены путем обобщения существующих ЭС (например, MYCIN, KS-300, MICRO-EXPERT и др.). Такие средства включают необходимые для реализации данной прикладной задачи структуры данных и стратегии управления и, таким образом, существенно уменьшают время программирования. Однако они недостаточно гибки и хорошо подходят лишь для класса приложений, к которым принадлежит исходная ЭС.

3. Применение специализированных сред разработки ЭС (например, KEE, FRT, LOOPS, CLIPS, JESS, ExSys Corvid и др.), которые поддерживают меньше готовых решений, но вместе с тем предоставляют больше возможностей (более гибкие по сравнению с оболочками). Они достаточно универсальны и включают целый набор необходимых средств, таких как ЯПЗ, языки программирования высокого уровня, редакторы экранных форм, библиотеки настраиваемых модулей и т.д. Однако, как правило, данные среды обладают очень простым интерфейсом, ориентированным на профессиональных программистов и мало подходят для применения специалистами-предметниками [54, 55]. Тем не менее существуют примеры сред, разработанных для автоматизированного программирования ЭС непрограммирующим пользователем (например, Expert System Development Tool) [56]. При этом данное средство ориентировано на собственный ЯПЗ – Exsys, БЗ которого не отчуждаемы от программной среды, а также отсутствуют механизмы настройки под определенную предметную область.
4. Применение специализированных редакторов БЗ (Expert System Designer, Expert System Creator, ES-Builder, ARITY Expert Development Package, CxPERT, ExSys Developer и др.), которые позволяют реализовать формализованное описание понятий предметной области и структур БЗ на определенном ЯПЗ, но обладают низкой интеграционной способностью с системами визуального моделирования и модулями интерпретации знаний, в большинстве случаев поддерживая один определенный ЯПЗ.
5. Применение методологий и программных средств проектирования БЗ с помощью специализированных визуальных (графических) нотаций. Так, использование визуального моделирования БЗ ведутся в ВШМ СПбГУ [57], среди зарубежных исследований можно выделить CAKE-технологию [58], а также исследовательский проект HeKatE (Hybrid Knowledge Engineering) [59]. В рамках данного проекта разработана методология проектирования интеллектуальных систем для управления и поддержки принятия решений, включающая визуальную разработку БЗ с использованием

специализированной нотации представления знаний в виде правил – XTT2 (eXtended Tabular Trees) [60].

6. Применение технологий для разработки интеллектуальных систем поддержки принятия решений (СППР) в слабоформализованных предметных областях [61]. Однако разработка таких систем является трудоемкой задачей, поскольку современные инструментальные средства разработки СППР либо не применимы в слабоформализованных предметных областях, либо малодоступны из-за высокой стоимости.
7. Применение подходов и программных средств автоматизации разработки гибридных ЭС [62-64];
8. Применение интегрированных сред разработки и унифицированных подходов, которые обеспечивают охват всех этапов жизненного цикла систем, основанных на знаниях, и интеграцию первых пяти направлений. При наличии в данной области решений, таких как АТ-ТЕХНОЛОГИЯ [65], необходимо отметить общую тенденцию к использованию концептуальных моделей при создании БЗ и ориентацию на непрограммирующих пользователей [66-74]. Активно развиваются подходы к созданию интеллектуальных систем на основе онтологий и семантических технологий [75-86], предназначенные для применения в сети Интернет.

В данном контексте перспективными являются подходы, основанные на порождающем программировании (generative programming) [87, 88], в частности, на модельно-управляемом (-ориентируемом) подходе – Model Driven Engineering (MDE) или Model Driven Development (MDD) [89-96] и его разновидностях, например, Model Driven Architecture (MDA) [97-101] – концепции реализации MDE/MDD от консорциума Object Management Group (OMG).

Подход MDE/MDD/MDA [89-101] – направление в области программной инженерии, предполагающее разработку программных систем на основе трансформации и интерпретации информационных моделей. Основные задачи данного направления связаны с разработкой методов и средств, автоматизирующих и снижающих сложность (трудоемкость) разработки

программных систем. Используемая идеология декларирует возможность синтеза программных кодов БЗ и охватывает отдельные аспекты рассмотренных ранее направлений [15, 39, 77, 102-105].

Преимуществами использования MDE/MDD/MDA-подхода являются:

- снижение затрат на разработку многоплатформенных ЭС (включая их БЗ) или на перенос разработанных ранее приложений на новую платформу за счет автоматической генерации кода или интерпретации моделей приложения;
- снижение затрат на этапах получения, концептуализации и формализации за счет повторного использования вычислительно-независимых моделей (Computation Independent Model, CIM) при создании БЗ;
- снижение нагрузки на специалистов-предметников за счет автоматизации этапа реализации БЗ и ЭС и предоставления возможности визуального моделирования причинно-следственных зависимостей (продукций).

Разработка теории и инструментов для создания программного обеспечения на основе отображения (трансформации) онтологий осуществляется в рамках таких направлений MDE/MDD, как Ontology Driven Architecture (ODA) [106] и Ontology Driven Software Engineering (ODSE) [107]. Данные направления основаны на исследованиях взаимодействия методов и средств программной инженерии с семантическими технологиями.

Существуют также отдельные решения, реализующие MDD/MDA-подход при разработке приложений баз данных (AndroMda, Bold for Delphi) [108], каркасов информационных систем [109], встраиваемых систем и программных компонентов для веб-приложений [110] и академические разработки веб-ориентированных сервисов и порталов [83, 111, 112].

Рассматривая все данные направления, можно сделать вывод о том, что исследователи, как правило, стараются предложить комплексное решение проблемы автоматизации создания интеллектуальных систем, в частности, систем, основанных на знаниях, ЭС, СППР, а также БЗ на основе использования различных подходов и методологий. При этом применяются разные способы и

источники получения знаний. Например, приобретение знаний из экспертов, текстов, баз данных, интернет ресурсов и др. Однако актуальными остаются вопросы обеспечения совместного (коллективного), удаленного характера процесса проектирования и генерации БЗ, а также использования различных видов концептуальных (информационных) моделей для его автоматизации.

1.2. Трансформация моделей

Задача повторного использования концептуальных моделей при разработке БЗ может быть сведена к задаче трансформации моделей. В общем случае трансформация моделей представляет собой процесс автоматической генерации целевой модели по исходной модели в соответствии с набором правил трансформации. При этом под правилом преобразования подразумевается описание того, как одна или более конструкций на исходном языке моделирования может быть преобразована в одну или более конструкций на целевом языке моделирования [97].

Трансформация является фундаментальной темой в области информатики и программной инженерии. Трансформация моделей тесно связана с областью преобразования программ (программной трансформации). Фактически их границы нечеткие и оба подхода иногда перекрывают друг друга.

Преобразование программ является более зрелой областью с сильными традициями программирования [113-117]. Так, например, при синтезирующем программировании как одной из трех форм программирования вместе со сборочным и конкретизирующим осуществляется пошаговое уточнение программ на основе спецификации задачи в виде предусловий и постусловий с использованием аксиоматического описания языковых конструкций. Пошаговое уточнение сопровождается сериями преобразований, отражающих внутреннюю природу языковых конструкций или факты о предметной области, известные

программисту. Процесс программирования приобретает характер доказательного рассуждения о существовании программы, решающей поставленную задачу [118].

С другой стороны, трансформация моделей является сравнительно новой областью. По существу, трансформация моделей является логическим продолжением (развитием) области преобразования программ. Так, трансформация модели может привести к трансформации программы, если программа основывается на модели. Тем не менее, подходы к трансформации, разработанные в обеих областях, имеют несколько разные характеристики. В то время как системы программных трансформаций, как правило, основаны на математически-ориентированных концепциях, таких как переписывание (term rewriting), атрибутивных грамматиках и функциональном программировании, системы трансформации моделей обычно применяют объектно-ориентированный подход для представления и манипулирования различными предметными моделями (абстракции системы и/или ее окружения) [119].

В программной инженерии термин модель часто используется для обозначения абстракций над программным кодом (например, спецификации требований или проектные спецификации). Некоторые исследователи при разработке программного обеспечения на основе моделей рассматривают программный код также в качестве моделей. Эта точка зрения согласуется с тем фактом, что программный код является абстракцией над машинным кодом, полученным компилятором. Однако спецификации требований или проектные спецификации часто являются моделями, которые более визуально-выразительны по сравнению с программным кодом. Так, данные модели часто представляются при помощи языков специализированных для определенного класса программных приложений или конкретного аспекта приложения (например, диаграммы взаимодействия UML сосредоточены на представлении аспекта взаимодействия широкого спектра систем). Данные языки принято называть языками предметно-ориентированного моделирования (Domain-Specific Modeling Languages, DSML) [120, 121].

Поскольку понятие модели является более общим, чем понятие программного кода, то при трансформации моделей, как правило, работают с более разнообразными наборами артефактов, чем при преобразовании программ. В литературе по трансформации моделей рассматривается широкий спектр артефактов в качестве потенциальных объектов трансформации для процесса разработки программного обеспечения. К ним относятся различные UML-модели, характеристические модели (feature models) [122], спецификации интерфейсов, схемы данных, компонентов и программный код.

Трансформация моделей является одной из основных составляющих модельно-ориентированного подхода к разработке программного обеспечения (MDE/MDD/MDA) [89-101]. В рамках данного подхода процесс разработки программного обеспечения представляет собой последовательное преобразование моделей с разным уровнем детализации, где на завершающем этапе генерируется исходный код программы.

Базовыми понятиями трансформации моделей являются [89-101]:

- Модель – абстрактное описание на некотором формальном языке характеристик системы (процесса), важных с точки зрения цели моделирования;
- Метамодель – модель языка, используемого для создания моделей (модель моделей);
- Мета-метамодель – язык, на котором описываются метамодели. Для построения метамоделей используются специальные языки метамоделирования. Наиболее распространенными языками метамоделирования являются: MOF (Meta-Object Facility) [123], Ecore [124], KM3 (Kernel Meta Meta Model) [125] и др.

Трансформация моделей представляется как процесс поочередного выполнения правил трансформации, при этом каждое правило описывает соответствия элементов исходной метамодели в элементы целевой метамодели.

В общем случае, трансформация моделей может быть представлена в виде схемы, изображенной на Рисунке 2.

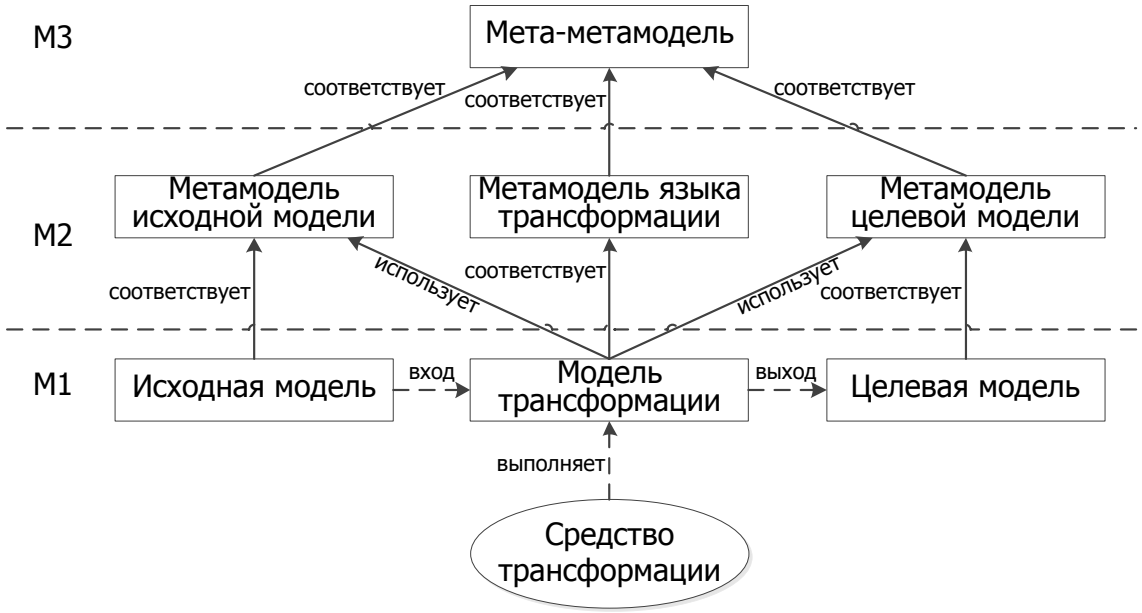

Рисунок 2. Трансформация моделей в рамках MDE/MDA-подхода [101]

В рамках MDE/MDA-подхода используется 4-х уровневая (от M0-M3) архитектура моделирования (иерархия моделей) [97-101]. На уровне «M1» (уровень моделей) находится некоторая исходная модель, которую необходимо преобразовать (перевести) в некоторую целевую модель при помощи некоторой модели (сценария или программы) трансформации. При этом уровень «M0» соответствует слою данных, т.е. он отражает объекты и процессы реального мира (данный уровень обычно не указывают на подобного рода схемах). Все модели уровня «M1» соответствуют некоторым метамоделям на уровне «M2» (уровень метамоделей), т.е. существуют некоторые формальные языки моделирования для их описания. Модель трансформации представляет собой набор правил преобразования, которые используют соответствующие элементы (конструкции) метамоделей исходной и целевой модели и описываются с использованием различных языков трансформации моделей (Model Transformation Language, MTL) [126], т.е. модель трансформации также соответствует некоторой метамодели языка трансформации. Таким образом, трансформация моделей осуществляется на уровне метамоделей («M2»). В свою очередь, все метамодели на уровне «M2» соответствуют некоторой одной мета-метамодели (мета-объектное средство) на уровне «M3» (уровень мета-метамодели), т.е. они описываются с использованием

некоторого языка метамоделирования. В подходе MDA в качестве языка метамоделирования используется стандарт – Meta-Object Facility (MOF) [123]. MOF призван служить мостом между разными метамоделями, поскольку представляет собой основу для их описания. Если две разные метамодели соответствуют MOF, т.е. могут быть описаны средствами уровня «M3», то все конкретные модели уровня «M1», базирующиеся на них, могут храниться в общем репозитории и совместно обрабатываться средствами модельных трансформаций.

Принято различать два вида трансформации моделей [119]:

1. Модель-Модель (Model-to-Model, M2M);
2. Модель-Текст (Model-to-Text, M2T или «pretty printing» – понятие из программной трансформации) и Текст-Модель (Text-to-Model, T2M).

При этом преобразование M2T в качестве целевых (выходных) текстовых артефактов может рассматривать исходный код (трансформация Модель-Код – Model-to-Code, M2C), документацию, спецификации и т.д.

Для того чтобы выполнить трансформацию моделей необходимо их описать с использованием некоторых языков моделирования, синтаксис и семантика которых заданы определенной метамоделью. В зависимости от языка моделирования, на котором описаны исходная и целевая модель, трансформации можно разделить на два типа [127]:

1. эндогенную (endogenous transformation) – преобразование моделей, описанных на одном языке моделирования;
2. экзогенную (exogenous transformation) – преобразование моделей, описанных на различных языках моделирования.

По существу, данное разделение трансформации является таким же, как и в таксономии программной трансформации [128], но переносимой на условия трансформации моделей. В этой таксономии термин «перезаписывание» (rephrasing) используется для эндогенного преобразования, в то время как термин «трансляция» (translation) используется для экзогенного преобразования.

Типичными примерами эндогенной трансформации (перезаписывания) являются:

- Модернизация или улучшение – это преобразование, направленное на улучшение некоторых эксплуатационных качеств программных систем (например, производительность и т.д.), при этом сохраняющее их семантику.
- Рефакторинг – это изменение внутренней структуры программного обеспечения для улучшения некоторых качественных характеристик (например, понятность, модифицируемость, возможность многократного использования, модульность, адаптивность и т.д.) без изменения его наблюдаемого поведения.
- Упрощение и нормализация – используется для уменьшения синтаксической сложности, например, путем перевода так называемого «синтаксического сахара» в более примитивные конструкции языка.

Типичными примерами экзогенной трансформации (трансляции) являются:

- Синтез – это преобразование более высокоуровневой, абстрактной спецификации (например, аналитической или проектной модели) в более низкоуровневую, конкретную спецификацию (например, модель Java-программы). В свою очередь, примером синтеза является генерация кода, где либо проектные (концептуальные) модели транслируются в исходный код, либо исходный код транслируется в некоторое промежуточное представление (например, байт-код, выполняемый на виртуальной машине) или непосредственно в исполняемый код.
- Обратное проектирование (обратная инженерия) – инверсия синтеза, т.е. позволяет извлекать более высокоуровневые, абстрактные спецификации из более низкоуровневых, конкретных спецификаций.
- Миграция – это преобразование программы, написанной на одном языке программирования, в программу, написанную на другом языке, но с сохранением того же уровня абстракции.

Трансформации моделей могут быть классифицированы по направлению преобразования на два типа [127]:

1. вертикальную (vertical transformation) – преобразование, при котором исходная и целевая модель принадлежат различным уровням иерархии;
2. горизонтальную (horizontal transformation) – преобразование, при котором исходная и целевая модель принадлежат одному уровню иерархии.

Примером вертикальной трансформации является уточнение (итеративная доработка), где спецификация постепенно переопределяется в полноценную реализацию путем последовательных шагов уточнения (детализации), которые добавляют более конкретные детали.

Примером горизонтальной трансформации является рефакторинг.

Определяют также и другую классификацию трансформации моделей по направлению преобразования:

1. однонаправленная трансформация (unidirectional transformation) – преобразование, при котором только из исходной модели можно получить целевую модель;
2. двунаправленная трансформация (bidirectional transformation) – преобразование, при котором, как из исходной модели можно получить целевую модель, так и наоборот.

На сегодняшний день можно выделить несколько основных направлений (подходов) к реализации трансформации моделей (M2M) [119]:

1. Подходы прямого императивного манипулирования (direct manipulation approach) – предоставляют внутреннее представление модели и API для манипулирования ею.
2. Структурные подходы (structure-driven approach) – предоставляют так называемые инкрементальные копии, которые пользователь должен наследовать для определения собственных правил трансформации. При этом базовым элементом является идея копирования элементов модели из исходной в целевую (создание иерархической структуры целевой модели).

3. Подходы на основе отношений (relational approach) – в данной категории собраны декларативные подходы, в которых концепцией являются математические отношения. Основной идеей является определение исходного и целевого типа элемента отношений и спецификация его ограничений.
4. Преобразование с использованием графовых грамматик (Graph Rewriting) [129] или подходы на основе трансформации графов (graph-transformation-based approach) – оперируют типизированными, раскрашенными графами с атрибутами, которые являются специально разработанными типами графов для представления моделей.
5. Подходы на основе логического программирования – использование логических языков (например, Prolog и т.д.) при составлении сценариев трансформации моделей.
6. Трансформация на основе различных языков (стандартов) преобразования XML-документов.
7. Гибридные подходы (hybrid approach) – являются комбинацией различных техник из предыдущих направлений.

Одним из самых широко распространенных направлений к реализации трансформации моделей (M2T) [119] является подход на основе шаблона (template-based approach). Шаблон обычно состоит из целевого текста, содержащего вставки метакода (гнезда), чтобы получить доступ к информации из источника для выполнения выбора кода и итеративного расширения. По сравнению с трансформацией на основе подхода с использованием паттерна «посетитель» (visitor-based approach), структура шаблона более близко напоминает код, который будет сгенерирован. Шаблоны поддаются итеративной разработки, поскольку они могут быть легко получены из примеров. Однако у данной категории подходов есть свой недостаток, а именно: поскольку шаблонные подходы работают с текстом, то шаблоны, которые они содержат, являются нетипизированными и могут представлять синтаксические или семантические неправильные фрагменты кода. С другой стороны, текстовые

шаблоны не зависят от целевого (выходного) языка и упрощают генерацию любых текстовых артефактов, включая различного рода документацию.

В рамках каждого направления создаются различные языки трансформации моделей. Большинство из этих языков развиваются в академическом сообществе. В настоящий момент наиболее распространенными языками трансформации моделей являются:

1. QVT (Query/View/Transformation) [130] – спецификация консорциума OMG, определяющая три языка трансформации моделей (декларативные, двунаправленные языки Core, Relations, а также императивный и однонаправленный язык Operational). Данные языки используются для модельных трансформаций (M2M) в модельно-управляемом подходе – Model Driven Architecture (MDA) [97-101].
2. ATL (ATLAS Transformation Language) [131] – язык описания трансформаций моделей базирующейся на стандарте QVT и стандартизованном языке описания ограничений OCL (Object Constraint Language) [132], позволяющий задавать преобразования любой исходной модели в указанную целевую модель. Преобразование производится на уровне метамоделей (моделей языка моделирования). Данный язык является гибридным, так как интегрирует декларативный и императивный подход, что позволяет производить преобразования для сложных предметных областей, при этом декларативный подход позволяет скрывать сложность алгоритмов трансформации.
3. VIATRA2 (VIsual Automated model TRAnsformations) [133] – основанный на правилах и паттернах язык преобразования для управления графовыми моделями, который комбинирует в единую парадигму спецификации:
    - для описания моделей – математический формализм, основанный на правилах трансформации графа;
    - для описания потока управления – абстрактные конечные автоматы.
4. GReAT (Graph REwriting And Transformation) [134] – язык описания преобразований модели, базирующийся на подходе тройных

трансформаций графа (правила перезаписывания графа применяются к входной модели и в результате создают выходную модель).

5. Henshin [135] – язык трансформации моделей, основанный на переписывании графа и использующий правила на основе шаблона, которые могут быть структурированы во вложенные единицы преобразования с четко определенной семантикой. Данный язык реализован в наборе инструментальных средств модельных трансформаций на базе платформы Eclipse Modeling Framework (EMF) [136].
6. Epsilon [137] – семейство языков и инструментальных средств для трансформации моделей, генерации кода, проверки достоверности моделей, миграций и рефакторинга. В основе Epsilon лежит императивный модельно-ориентированный язык Epsilon Object Language (EOL), который сочетает в себе процедурный стиль JavaScript с возможностями языка ограничений OCL. Epsilon Transformation Language (ETL) – язык трансформации моделей на основе правил, поддерживающий преобразование множества входных моделей в выходные, наследование правил, определение «ленивых» и «жадных» правил преобразования, а также возможность запрашивать и изменять входную и выходную модель.
7. XSLT (eXtensible Stylesheet Language Transformations) [138] – язык преобразования XML-документов. Данный язык является спецификацией консорциума W3C. Таблицы стилей XSLT состоят из набора шаблонов. При применении данных таблиц к исходному XML-документу (исходное дерево) образуется целевое конечное дерево, которое может быть представлено в виде XML-документа, HTML-документа или простого текстового файла.

Основным недостатком данных языков трансформации являются высокие требования к пользователю (специалисту) при разработке правил (сценария) трансформаций. В частности, пользователю необходимо знать не только синтаксис и семантику определенного языка трансформации моделей (который может быть достаточно сложен), но и языки метамоделирования (MOF, Ecore, KM3 и др.), используемые для описания входной и выходной моделей, а также

различных сопутствующих (дополнительных к основным языкам) языковых конструкций (например, язык ограничений – OCL [132]). Следует также отметить, что программные средства, поддерживающие данные языки, зачастую не предоставляют возможность визуализации процесса описания правил трансформации моделей, т.е. правила преобразований создаются в специальных текстовых редакторах, ориентированных на программистов. Также существенным ограничением (недостатком) почти всех языков трансформации моделей является жесткая привязка к определенному программному инструментарию, в частности, к платформе Eclipse (поддержка языков реализована в виде модулей/плагинов) [139] и EMF [136]. Совокупность этих факторов затрудняет практическое использование этих языков и программных средств непрограммирующими пользователями (специалистами-предметниками, инженерами по знаниям, аналитиками и т.д.), в частности, при разработке БЗ на основе концептуальных моделей.

1.3. Разработка БЗ на основе трансформации концептуальных моделей

В настоящее время существует ряд работ, направленных на решение задачи повышения эффективности разработки БЗ путем трансформации различных концептуальных моделей. Условно все работы по данной тематике можно разделить на две группы:

1. Преобразование исходных концептуальных моделей, описанных с использованием различных языков (стандартов) моделирования, в целевые БЗ на определенном ЯПЗ. В данной группе работ исследователи акцентируются на конкретных языках и нотациях моделирования, при помощи которых описываются исходные данные (модели), которые необходимо преобразовать. При этом, как правило, опускается информация о конкретном текстовом синтаксисе исходных данных (моделей).

В частности, Kiko K. и Atkinson C. в работе [140] провели исследование детального сравнения между конструкциями UML-моделей (диаграмм классов) [141] и OWL-онтологий [142]. В работах [143, 144] предлагается метамодельный управляемый подход к трансформации моделей для передачи (обмена) правилами между онтологиями OWL/SWRL и диаграммами UML/OCL. В качестве промежуточного представления знаний (правил) используется язык REWERSE Rule Markup Language (R2ML), а для описания правил трансформации использован язык трансформации моделей ATL. В работах [145, 146] исследователи представляют подход трансформации диаграмм классов UML в онтологии OWL 2 с использованием языка QVT-R. В [147] предложен подход преобразования диаграмм классов UML в онтологии OWL на основе графовых трансформаций и с использованием программного средства AToM3. Parreiras F.S. и Staab S. в работе [148] описывают подход TwoUse, позволяющий совместить моделирование UML и BPMN с семантическими технологиями (онтологиями OWL/SWRL). В рамках данного подхода разработан новый язык – SPARQLAS, позволяющий указывать специальные выражения ограничений и запросы. В работе [106] представлена метамодель ODM (Ontology Definition Metamodel) для описания онтологий OWL в рамках концепции 4-х уровневой архитектуры MDA с помощью MOF. ODM позволяет более широко использовать известную UML-нотацию в онтологической инженерии, в частности, для отображения между элементами UML и OWL. Данные идеи (ODM), а вместе с ними и принципы MDD/MDA-подхода, были использованы в работе [149]. В данном исследовании представлен подход для визуального моделирования онтологий OWL DL и OWL Full на основе преобразования концептуальных моделей UML Profile. Подход реализован в форме двух программных инструментов (плагинов) – Visual Ontology Modeler (VOM) и Integrated Ontology Development Toolkit (IODT) в составе платформы (фреймворка) EMF. В работах [150, 151] описываются подходы к преобразованию UML-моделей в OWL-онтологии с использованием XSLT. Другими успешными примерами решения задачи автоматизированной

разработки БЗ (онтологий) путем трансформации UML-моделей являются [152-159].

В работе [160] представлен набор шаблонов проектирования онтологий, обеспечивающих способ получения онтологии OWL/SWRL, применяя правила преобразования метамодели спецификации SBVR. Разработанный набор сопоставлений (отображений) между этими двумя стандартами предназначен для бизнес-экспертов и/или разработчиков, заинтересованных в реализации соответствующего инструментального средства отображения.

Alberts R. и Franconi E. в работе [161] предложили интегрированный метод, использующий концептуальное моделирование для создания механизма запросов на основе онтологии. Метод предусматривает преобразование моделей предметной области в формате ORM (Object Role Modelling) в онтологии OWL. Язык XSLT использован на первой стадии преобразования концептуальных моделей из формата XML в модель ORM2 с использованием программного средства (плагина) NORMA. Далее, на второй стадии, используется специально разработанный инструмент – ORM2OWL для преобразования модели ORM2 в онтологию OWL 2.

В работе [162] описан 4-х этапный метод конвертации больших тезаурусов (Medical Subject Headings (MeSH) и WordNet в формате стандартов ISO и ANSI) в онтологии форматов RDF(S) и OWL.

Исследователи Myroshnichenko I. и Murphy M.C. в работе [163] предлагают подход автоматического отображения схем сущность-связь (ER-моделей) в семантически эквивалентные онтологии в формате OWL Lite. Набор правил отображения реализован в средстве SFSU ER Design Tools.

В работе [164] предложен подход к созданию онтологий в формате RDF на основе УФО-моделей (Узел-Функция-Объект) предметной области.

Как отмечалось ранее, при разработке БЗ (в частности, онтологий) активно используются концепт-карты. Так, в работе [165] предложен метод интеграции концептуальных карт с онтологиями OWL, который представляет собой набор формальных преобразований, применяемых к концептуальным картам в формате

XML (средство MACOSOFT), и семантического анализа отношений, связывающих понятия. В работах [46, 166] представлены подходы к трансформации концепт-карт CmapTools в онтологию предметной области в формате OWL.

2. Преобразование исходных данных (моделей) представленных на XML в целевые БЗ на определенном ЯПЗ. В данной группе работ исследователи акцентируются на конкретном текстовом синтаксисе (XML) исходных данных (моделей), которые необходимо преобразовать. При этом, как правило, опускается информация о конкретных языках и нотациях моделирования, при помощи которых сформированы исходные данные (модели).

К настоящему времени в мире накоплено огромное количество данных, представленных в формате XML. XML (eXtensible Markup Language) [167] является универсальным и наиболее распространенным способом интеграции программных систем и обеспечения обмена информацией между приложениями. Существует ряд научных работ [168-175], направленных на преобразование различных исходных XML-данных в целевые БЗ.

В частности, Bohring H. и Auer S. в работе [168] предлагают подход и программное средство XML2OWL для преобразования XML-моделей данных в онтологии OWL. Трансформация осуществляется на уровне XML-схемы модели данных (сопоставление элементов XSD (XML Schema Definitions) [176] и OWL). Для реализации трансформаций используется XSLT.

В работах [169, 170] представлен подход (нотация) для отображения XML-схем в структуры онтологий OWL и автоматического перевода объектов XML-документа в онтологию OWL. Подход реализован на Java в виде фреймворка JXML2OWL для автоматического преобразования различных источников данных в формате XML в семантическую модель, определенную на языке OWL. Фреймворк поддерживает отображения: один-к-одному, один-ко-многим, многие-к-одному, многие-ко-многим. Для реализации трансформаций используется XPath.

Исследователи O'Connor M.J. и Das A.K. в [171] предлагают предметно-ориентированный язык – XMLMaster, предназначенный для преобразования XML-документов в произвольные онтологии в формате OWL. Язык основан на Манчестерском синтаксисе OWL и расширяет его с помощью языка запросов XPath для описания XML-документов.

В работе [172] представлен набор шаблонов (паттернов) для прямого автоматического преобразования XML-схем в OWL-онтологии (OWL 2 RL), позволяя тем самым интегрировать множество данных, представленных в формате XML в семантическую сеть. При этом подход предусматривает нормализацию, фильтрацию и трансформацию XML-данных. Данный подход реализован на Java в виде программного средства – Janus.

В [173] исследователи предлагают подход к автоматической трансформации XML-документов соответствующих схеме DTD (Document Type Definitions) [177] в онтологии OWL. На основе данного подхода разработан фреймворк DTD2OWL.

Другими успешными примерами решения задачи автоматизированной разработки БЗ (онтологий) путем трансформации XML-данных являются [174, 175].

Близкой (родственной) задачей к разработке БЗ на основе трансформации концептуальных моделей является интеграция (совмещение) БЗ, основанных на различных моделях представления знаний [4] и/или представленных на разных ЯПЗ (например, актуальна задача создания гибридных БЗ на основе онтологий и правил [178]). Интеграция различных моделей, методов и средств представления и обработки знаний актуальная при создании современных систем, основанных на знаниях, поскольку ни одно из этих средств, взятое в отдельности, не может обеспечить в полном объеме потребностей разработки реальной прикладной интеллектуальной системы [179]. В некоторых случаях исходная БЗ может быть использована в качестве концептуальной модели при генерации кода другой целевой БЗ (например, онтология предметной области иногда рассматривается как концептуальная модель).

В частности, Дубинин В.Н. и Вяткин В.В. в [76] предлагают подход к проектированию систем управления (онтологическое описание управляющего приложения с использованием функциональных блоков IEC 61499) на основе трансформации онтологий OWL. Для трансформации онтологий предлагается язык eSWRL, являющийся расширением онтологического языка правил SWRL (Semantic Web Rule Language) [180]. Для интерпретации конструкций eSWRL используется Prolog.

Авдеенко Т.В. и Макарова Е.С. в [181] описывают подход к упрощению процесса принятия решений в СППР. Этот подход основан на преобразовании знаний из неявного представления в форме прецедентов в явное представление в виде продукционных БЗ.

В работе [182] представлена дедуктивная объектно-ориентированная система БЗ – R-DEVICE, предназначенная для рассуждений (reasoning) над метаданными RDF. R-DEVICE импортирует RDF-документы в систему продукционных правил CLIPS, при этом преобразовывает RDF-триплеты в объекты COOL и использует дедуктивный язык правил для их вывода (reasoning). При этом пользователь может выбирать формат получаемой БЗ (выбор между OPS5/CLIPS-подобным и RuleML-подобным синтаксисом).

Исследователи Meditskos G. и Bassiliades N. в [183] предлагают средство CLIPS-OWL для преобразования онтологий OWL DL в объектно-ориентированную модель CLIPS (COOL). Это позволяет CLIPS использовать полученные объектно-ориентированные модели в качестве статических моделей для запросов с использованием машины вывода RETE без взаимодействия (настраивания) во время работы с внешним резонером (reasoner) DL. Таким образом, любое приложение на основе CLIPS может повысить свою функциональность за счет включения (использования) онтологический знаний без изменения архитектуры процессора правил CLIPS.

В работе [184] представлен алгоритм трансформации онтологий OWL в правила (продукции) для ЭС. При этом концепт-карты выступают промежуточным средством для представления извлеченных знаний.

Одним из самых популярных средств интеграции онтологических знаний OWL и правил является плагин JessTab [185] для онтологического редактора Protégé. При этом существуют работы [186], улучшающие данное решение путем повторного использования уже существующих правил.

В работе [187] предложен прототип программного средства OWL2JESS, который обеспечивает трансформацию OWL-онтологий в БЗ JESS. Трансформация реализована с использованием XSLT.

Wang E. и Kim Y.S. в [188] описывают метод автоматической конвертации правил SWRL в JESS, состоящий из 4 шагов: правила SWRL создаются с использованием плагина SWRLTab для Protégé; затем созданные SWRL-правила (OWL-синтаксис) конвертируются в формат SWRLRDF с использованием XSLT; далее осуществляется преобразование SWRLRDF в формат CLIPS/JESS с помощью средств проекта SweetRuls [189]; для преобразования расширенного синтаксиса SWRL применяется мета-программирование в JESS.

SweetRules [189] – это набор программных инструментов (инициативный проект), который предоставляет возможность преобразовывать правила между несколькими стандартными форматами, включая RuleML, JESS, SWRL/OWL, CommonRules, Jena и XSB Prolog.

В работе [190] описывается подход и система SweetProlog для преобразования OWL-онтологий и правил, представленных в OWLRuleML, в набор фактов и правил, описанных на языке Prolog.

Также можно упомянуть некоторые другие близкие по тематике исследования. В частности, в работе [191] представлен подход и средство для трансформации правил SWRL из формата Unary/Binary Datalog в Frame Datalog PSOA RuleML. В работе [192] предлагается автоматически генерировать отображения схем среди БЗ RDF, используя один образец обмена и набор соответствий многие-ко-многим. Проверка обмена данными осуществлялся на БЗ DBpedia, Freebase и GovWILD. Гадиатулин Р.А. и Чуприна С.И. в [193] описывают подход к реализации автоматизированного извлечения онтологий OWL DL из продукционных БЗ ЭС. Подход реализован в виде оболочки

OntoMaker 2.0, которая, наряду с возможностями автоматизированного извлечения знаний, предоставляет визуальные средства для редактирования и построения онтологий.

В Таблице 1 приведена краткая сравнительная характеристика рассмотренных работ в области создания БЗ на основе трансформации различных концептуальных моделей.

На основе проведенного анализа данных работ можно сделать следующие основные выводы об их недостатках:

- Узкая специализация в части поддержки форматов (языков моделирования) концептуальных моделей (как правило, это один язык, например, UML [141]). Вследствие чего отсутствует гибкость при разработке БЗ, т.к. пользователь при необходимости не сможет трансформировать концептуальные модели других форматов. Исключением здесь могут быть работы [168-175], т.к. они направлены на трансформацию различных моделей, синтаксис (concrete syntax) которых представлен (выражен) в виде XML.
- Узкая специализация в части поддержки форматов ЯПЗ. Как показано в таблице 1, в большинстве случаев исследователи используют семантические сети (онтологии) в качестве целевой модели представления знаний и, как правило, язык (стандарт) OWL [142] для их описания. При этом данный формат поддерживается не полностью. Например, поддержка только конкретного диалекта (так в ряде работ используется устаревшая спецификация OWL Lite) или определенной версии OWL (1 или 2).
- Жесткое задание (определение) соответствий между элементами исходной концептуальной модели и БЗ на целевом ЯПЗ при реализации. Вследствие чего отсутствует возможность изменить интерпретацию отображений элементов без внесения изменений в программный код соответствующего программного средства.
- Достаточно сложная реализация, что, в свою очередь, обуславливает разнообразие программного обеспечения, используемого исследователями в

рамках каждого отдельного преобразования (иногда на каждом этапе преобразования используется отдельное специализированное средство).

- Использование разных методологий (подходов) к решению поставленной задачи: от так называемых «ad-hoc-решений», пригодных для конкретного описываемого случая (конкретной задачи), до различных общих стандартов в области программной инженерии и модельно-ориентированного подхода (в том числе разных инициатив MDE).
- Высокие квалификационные требования к пользователям. Как правило, существующие подходы ориентированы на программистов, а не экспертов и инженеров по знаниям (аналитиков).

Исходя из данных особенностей, можно сделать вывод об отсутствии единых принципов построения систем для разработки БЗ на основе трансформации концептуальных моделей, что, в свою очередь, подтверждает актуальность разработки моделей, методов и программных средств автоматизации создания подобного рода систем.

В частности, для обеспечения гибкости выбора форматов языков моделирования и ЯПЗ предлагается использовать:

- Группу концептуальных моделей, которые описываются с использованием XML. Формат XML является универсальным и наиболее распространенным способом представления и хранения концептуальных моделей, интеграции программных систем и обеспечения обмена информацией между приложениями. Примерами моделей данной группы являются: UML-модели [141, 194], представленные в соответствии со стандартом XMI (XML Metadata Interchange) [195] – стандарт консорциума OMG (Object Management Group) для обмена метаданными с помощью языка XML; концепт-карты, представленные в соответствии со стандартом XTM (XML Topic Maps) [196] – стандарт ISO для представления тематических карт (topic maps), и т.д.
- CLIPS и OWL в качестве целевых ЯПЗ, как наиболее распространенных на данный момент средств представления знаний в виде продукций и

онтологий. Более подробное обоснование выбора данных ЯПЗ рассматривается в следующем разделе.

Для поддержки гибкости процесса задания соответствий между элементами исходной концептуальной модели и целевой БЗ, а также унификации программного обеспечения предлагается разработать:

- Специализированный метод и инструментальное средство обладающее свойством расширяемости в части создания дополнительных программных компонентов, обеспечивающих трансформацию концептуальных моделей в БЗ.
- Новый предметно-ориентированный язык описания трансформаций.

Для расширения категории пользователей, в том числе непрограммирующих (специалистов-предметников, инженеров по знаниям, системных аналитиков и т.д.), предлагается использовать:

- Визуальное интерактивное построение правил преобразования исходных концептуальных моделей в целевые БЗ.
- Визуальное моделирование БЗ.

Таблица 1. Сравнение работ, описывающих разработку БЗ на основе трансформации концептуальных моделей

Работы/ Критерии	Исходные концепту-альные модели	Целевые БЗ	Используемые методологии (стандарты)	Использованные языки и подходы к трансформации	Реализация	Ориентация на не программирующих пользователей
[143, 144]	UML/OCL и R2ML	R2ML и OWL DL/ SWRL	MDA/MOF, EMF/ODM (промежуточное представление R2ML)	ATL	Плагин для платформы EMF	-
[145, 146]	UML	OWL 2	MDA/MOF	QVT-R	Ad-hoc	-
[147]	UML	OWL DL	MDE/MDA	Трансформация графов	AToM3	-
[148]	UML/OCL и BPMN	OWL DL/ SWRL	Ad-hoc + EMF (MOF/Ecore)	SPARQLAS	TwoUse Toolkit	-
[106]	UML	OWL DL	ODM	Ad-hoc	-	+
[149]	UML Profile	OWL DL и OWL Full	MDA/ODM	QVT	Плагины VOM и IODT для EMF	-
[150]	Ontology UML Profile (OUP)	OWL	ODM	XSLT	Плагин для CASE-средств UML (Poseidon for UML)	+
[151]	UML	OWL	Ad-hoc	XSLT	Ad-hoc	-
[152, 153]	UML	OWL	Ad-hoc	Ad-hoc	Ad-hoc	-

Работы/ Критерии	Исходные концепту-альные модели	Целевые БЗ	Используемые методологии (стандарты)	Использованные языки и подходы к трансформации	Реализация	Ориентация на не программирующих пользователей
[154]	UML	OWL DL/ SWRL	Ad-hoc	Ad-hoc	Ad-hoc	-
[155]	UML	OWL DL	Ad-hoc	Ad-hoc (алгоритм U2OTrans) и XSLT	UML2OWL	-
[156, 157]	UML	DAML+ OIL	Ad-hoc	Ad-hoc	Модуль CAWICOMS	-
[158]	UML	OWL	Ad-hoc	Ad-hoc	Ad-hoc	-
[159]	UML (XMI)	RDF	MDA	Ad-hoc на основе API JMI	Ad-hoc	-
[160]	SBVR	OWL и SWRL	Ad-hoc	Ad-hoc на основе набора шаблонов	Ad-hoc	+
[161]	ORM	OWL 2	Ad-hoc (2 этапа отображения)	XSLT	NORMA и ORM2OWL	-
[162]	MeSH и WordNet	RDF(S) и OWL	Ad-hoc	XSLT и Prolog	Ad-hoc	-
[163]	ER	OWL Lite	Ad-hoc	Ad-hoc	SFSU ER Design Tools	-
[164]	УФО-модели	RDF	Ad-hoc	Ad-hoc	Ad-hoc	-

Работы/ Критерии	Исходные концепту-альные модели	Целевые БЗ	Используемые методологии (стандарты)	Использованные языки и подходы к трансформации	Реализация	Ориентация на не программирующих пользователей
[165]	Концепт-карты XML (на основе WordNet)	OWL Lite (DL частично)	Ad-hoc	Ad-hoc	Ad-hoc	-
[166]	CmapTools	OWL DL	Ad-hoc	Prolog	Translator Module	-
[46]	CmapTools	OWL	Ad-hoc	Ad-hoc	Ad-hoc	+
[168]	XML-схема и объекты	OWL DL	Ad-hoc	XSLT	XML2OWL	-
[169, 170]	XML-схема	OWL DL	Ad-hoc	XPath	JXML2OWL	-
[171]	XML-схема и объекты	Manchester OWL syntax	Ad-hoc	XPath	XMLMaster	-
[172]	XML-схема	OWL 2 RL	Ad-hoc	Ad-hoc	Janus	-
[173]	DTD и XML-объекты	OWL DL	Ad-hoc	XSLT	DTD2OWL	-

Работы/ Критерии	Исходные концептуальные модели	Целевые БЗ	Используемые методологии (стандарты)	Использованные языки и подходы к трансформации	Реализация	Ориентация на не программирующих пользователей
[174]	XML-схема	OWL DL	Ad-hoc	Ad-hoc	Ad-hoc	-
[175]	XML-схема (XSG)	OWL DL	Ad-hoc (цепочка средств)	Ad-hoc	Trang, XSOM, JUNG и Jena	-
[182]	RDF	CLIPS/ COOL	Ad-hoc	Ad-hoc и XSLT	R-DEVICE	-
[183]	OWL DL	CLIPS/ COOL	Ad-hoc	Ad-hoc	CLIPS-OWL	-
[185]	OWL DL	JESS	Ad-hoc	Ad-hoc	Плагин JessTab для Protégé	-
[187]	OWL DL	JESS	Ad-hoc	Ad-hoc и XSLT	OWL2JESS	-
[188]	SWRL	JESS	Ad-hoc (4-х этапный подход)	Ad-hoc	Плагин SWRLTab для Protégé и средства SweetRules	-
[190]	OWL	Prolog	Ad-hoc (+ доп. преобразования)	Ad-hoc	SweetProlog	-

1.4. Модели и языки представления знаний

В рамках исследований по искусственному интеллекту накоплен большой опыт и широкий спектр различных средств и методов представления и обработки знаний. Так, семантические сети (semantic networks) [197] рассматриваются в качестве универсального средства (памяти) для хранения любой информации, которую можно представить в виде совокупности объектов и отношений между ними. Фреймы [198], как модели абстрактных образцов (минимально возможные описания сущностей каких-либо объектов, явлений или процессов), служат для повышения уровня представления знаний, используя при этом объектно-ориентированный подход. Логическое программирование [199] используется для декларативного описания сложных процессов обработки информации.

Однако в последнее время широкую популярность у разработчиков БЗ получили онтологии. Существует множество определений понятия «онтология». Одно из самых известных и классических определений понятия «онтология» дал Томас Грубер:

«*Онтология – это эксплицитная спецификация концептуализации, где в качестве концептуализации выступает описание множества объектов предметной области и связей между ними*» [200].

Другой известный исследователь в области онтологического инжиниринга, Никола Гуарино, определяет онтологию как «*формальную теорию, ограничивающую возможные концептуализации мира*» [201].

Определения онтологии также были даны известными российскими учеными, в частности, Гавриловой Т.А., Хорошевским В.Ф., Добровым Б.В., Лукашевич Н.В.:

«*Онтологии – это БЗ специального типа, которые могут «читаться», пониматься и отчуждаться от разработчика и/или физически разделяться их пользователями*» [2].

«*Онтология – это система, состоящая из набора понятий и набора утверждений об этих понятиях, на основе которых можно описывать классы, отношения, функции и индивиды*» [202].

Рассматривая различные определения термина «онтология», исследователи, помимо рассмотрения знаний в виде сети (концептуальной модели), состоящей из набора понятий (концептов) и связей между ними, делают акцент на однозначно точной интерпретации и согласованное (непротиворечивое) понимание этой концептуальной модели определенным сообществом (группой людей) и программными агентами.

В области информационных технологий термин «онтология» вначале использовался рядом исследовательских сообществ по искусственному интеллекту в инженерии знаний, в обработке естественных языков, в представлении знаний. Но уже в конце 1990-х – начале 2000-х годов понятие онтологии также стало широко использоваться в таких областях, как интеллектуальная интеграция информации, поиск информации в Интернете и управление знаниями [3].

Позже онтологии стали рассматриваться в качестве ключевого элемента в проекте (концепции) Семантической паутины, предложенной сэром Тимом Бернерсом-Ли в 2001 году [203]. Семантическая паутина (Semantic Web) – это общедоступная глобальная семантическая сеть, формируемая на базе Всемирной паутины (новый этап развития сети – Word Wide Web, WWW) путём стандартизации представления информации в виде, пригодном для машинной обработки [204].

Для этой цели в 2004 году консорциумом Всемирной паутины (W3C) был разработан стандарт на спецификацию онтологий – OWL [205].

Web Ontology Language (OWL) – язык описания онтологий для Семантической паутины, основанный на более ранних языках (OIL и DAML) и в настоящее время является «de facto» самым мощным и распространенным средством для представления знаний в виде онтологий.

В качестве основы (базиса) для формализованного описания онтологий используется логика, в частности, дескрипционная логика (Description Logic, DL) [206]. OWL построен как расширение RDF [207] и RDF(S) [208]. Это означает, что основная конструкция – это триплет (объект-предикат-объект) языка RDF. В этом контексте язык OWL можно рассматривать как расширенный вариант RDF(S), позволяющий не только описывать классы и свойства, но также задавать ограничения на их использование [209].

На настоящий момент актуальной считается 2 версия OWL, в которой определены два диалекта: DL (содержит три профиля: OWL EL, OWL QL, OWL RL) и Full [142, 210].

OWL2 имеет пять различных конкретных синтаксисов (concrete syntax). Основным синтаксисом OWL2 является – RDF/XML [211]. Этот синтаксис должен поддерживаться всеми программными инструментами, работающими с онтологиями в формате OWL2 [212].

Однако могут использоваться и другие дополнительные конкретные синтаксисы OWL2. Они включают в себя альтернативные сериализации RDF, такие как: Turtle [213]; сериализация XML [214] и более «читаемый» синтаксис – Манчестерский Синтаксис (Manchester Syntax) [215], который используется в некоторых редакторах онтологий. Наконец, синтаксис функционального стиля (Functional Syntax) также может использоваться для сериализации онтологий OWL2, хотя его основной целью является определение структуры языка [216].

Несмотря на возрастающую популярность семантических технологий и онтологий, ЯПЗ, основанные на правилах, до сих пор остаются наиболее распространенными и востребованными при разработке ЭС.

БЗ, представленные на продукционных ЯПЗ, состоят из набора данных (фактов) и правил (функций). При этом правило включает в себя «условие» (антецедент) и «действие» (консеквент), где под «условием» понимается некоторое предложение-образец, по которому осуществляется поиск в БЗ, а под «действием» – некоторая операция, выполняемая при успешном исходе поиска.

Продукционные БЗ часто применяются в промышленных ЭС, поскольку привлекают разработчиков своей наглядностью, высокой модульностью, легкостью внесения дополнений и изменений и простотой механизма логического вывода [217].

В настоящее время существует множество ЯПЗ продукционного типа: OPS5, JESS, Drools, RuleML, SWRL и др.

Одним из наиболее широко используемых продукционных ЯПЗ и инструментальных сред для разработки продукционных ЭС является C Language Integrated Production System (CLIPS) [54]. Такую популярность CLIPS получил благодаря своей скорости, эффективности и бесплатности.

Первые версии CLIPS разрабатывались Космическим агентством NASA с 1984 года. В своей оригинальной версии данный ЯПЗ использует прямой логический вывод (реализация вывода использует алгоритм Rete) [4]. Следует также отметить, что CLIPS до сих пор обновляется и поддерживается своим изначальным автором – Гэри Райли. На сегодняшний день актуальной считается 6 версия CLIPS (6.3).

CLIPS включает полноценный объектно-ориентированный язык для написания ЭС – COOL (CLIPS Object-Oriented Language). Благодаря наличию языка COOL, пользователи CLIPS могут манипулировать не только фактами и правилами, переменными и функциями, но и такими понятиями, как классы и объекты.

Описание синтаксиса (concrete syntax) и наиболее важных определений и конструкторов CLIPS (включая язык COOL) с использованием БНФ (Форма Бэкуса-Наура) приводится в [217].

Таким образом, в качестве целевых ЯПЗ, поддерживаемых предлагаемой технологией, выбраны CLIPS и OWL, как наиболее распространенные на данный момент средства представления продукций и онтологий.

1.5. Технологии и средства разработки программных систем и их компонентов

Одна из главных задач современного программирования – создание теоретических и прикладных основ построения сложных программ из более простых программных элементов [218].

В настоящее время существует большое количество различных парадигм (подходов) программирования, представляющих совокупность идей и понятий, определяющих стиль написания компьютерных программ [219], в частности:

1. Императивное программирование (процедурное [4], структурное [220], модульное программирование). Большой вклад в теорию и практику создания систем на основе программных модулей и пактов прикладных программ сделали такие ученые, как Ершов А.П. [221], Тыугу [222], Опарин Г.А. [223] и др. Развитием модульного подхода является сборочное программирование, характеризующееся сборочным построением программ из готовых «деталей», которыми являются программные объекты различной степени сложности [218] и компонентно-ориентированное программирование (component-based development или component-oriented programming) [224], которое опирается на понятие компонента – независимого модуля исходного кода программы, предназначенного для повторного (многократного) использования и развёртывания, и реализующегося в виде множества языковых конструкций, объединённых по общему признаку и организованных в соответствии с определёнными правилами и ограничениями. В сборочной стратегии широко используется каркас (шаблон) – типовая повторно возникающая ситуация на уровне модели программной системы, которая определяет структуру проекта и имеет недоопределенные элементы, обозначенные пустыми слотами (гнездами) для расположения в них новых определенных компонентов [218, 225]. Таким образом, при создании своей конкретной программы

разработчик заново пишет лишь некоторые из вложенных модулей (ориентированные на решение какой-либо специальной задачи), заполняющих гнезда готового каркаса [226, 226]. Заполненный компонентами каркас становится контейнером (сам может быть компонентом программной системы), обеспечивающего инкапсуляцию компонентов и доступ к компонентам через каркас, который определяет возможные варианты наполнения контейнера.

2. Объектно-ориентированное программирование [224, 227].
3. Декларативное программирование, которое часто используется при создании спецификаций (средств описания, формализации задач, программ) [228]. При этом важными свойствами спецификаций являются точность, понятность и полнота [229]. К подвидам декларативного программирования зачастую относят функциональное и логическое программирование.
4. Порождающее программирование (generative programming) [87, 88] и, в частности модельно-ориентированный подход (MDE/MDD/MDA) [89-101], который уже рассматривался ранее.

Необходимо отметить, что парадигма программирования не определяется однозначно языком программирования. Практически все современные языки программирования в той или иной мере допускают использование различных парадигм, при этом существующие парадигмы зачастую пересекаются друг с другом в деталях, поэтому можно встретить ситуации, когда разные авторы употребляют названия (термины) из разных парадигм, говоря при этом, по сути, об одном и том же явлении.

Таким образом, при реализации предлагаемых моделей и методов, предполагается использовать сочетание различных подходов программирования, в частности, за основу взять некоторые идеи компонентно-каркасного подхода. Тем самым осуществить разработку программного компонента (модуля-конвертера) интеллектуальной системы, обеспечивающего генерацию кода БЗ на основе трансформации концептуальных моделей, в форме специализации

(настройки) каркаса (шаблона) типового программного компонента посредством конкретного сценария (программы) преобразования.

Помимо представленных подходов программирования, необходимо также рассмотреть некоторые виды реализации программных систем.

Так, на сегодняшний день веб-приложения получают все большую популярность. При этом они имеют ряд существенных достоинств по сравнению с настольными (desktop) приложениями, а именно [230]:

1. Приложение абсолютно не зависит от того, какая операционная система установлена на компьютере пользователя, то есть приложение является кросс-платформенным.
2. Способ распространения продукта. Отход от традиционных способов распространения программных продуктов путем продажи копий и установки их на каждый компьютер пользователей. Теперь единственная версия приложения расположена на сервере, а все пользователи имеют доступ к ней, вернее, к ее пользовательскому интерфейсу из любого места в мире, где есть Интернет. При этом пользователю не нужно устанавливать новую версию приложения – сразу после своего появления она доступна всем.
3. Пользователю нет необходимости устанавливать и настраивать программное обеспечение – все уже установлено на серверах и настроено разработчиками.
4. Для работы с приложением от пользователя потребуется только установленный браузер. Использование веб-приложений во многом снимает ограничения, накладываемые на аппаратную часть компьютера пользователя.
5. Ввиду того, что основная часть веб-приложения сконцентрирована на сервере в одном месте, значительно упрощается его настройка, не нужно содержать большое количество специалистов технической поддержки, занимающихся консультациями пользователей и настройкой приложения на компьютерах во всем мире. Это гораздо менее затратно в финансовом плане

и куда более эффективно. При этом пользователю невидима архитектура приложения, в любой момент можно добавить любое количество серверов, на которых установлена основная составляющая приложения, добавить вычислительные мощности, и пользователь этого даже не заметит.

Как отмечено выше, веб-приложения имеют большое количество достоинств при отсутствии видимых недостатков, самым большим и очевидным из которых является невозможность использования веб-приложений при отсутствии или ограничении (например, ширины канала, не стабильности соединения и т.д.) доступа к Интернет [230].

Таким образом, предлагается применить принципы веб-ориентированной разработки приложений при создании инструментального средства, реализующего предлагаемые модели и методы.

1.6. Выводы

- При наличии специализированных методов, средств и подходов, направленных на создание БЗ, в том числе, на основе концептуальных моделей, практически отсутствуют единые принципы или унифицированные технологии создания подобных систем и их компонентов.
- Существующие программные решения ориентированы на высококвалифицированных специалистов-программистов, что обуславливает необходимость создания технологии и средств поддерживающих непрограммирующих пользователей: специалистов-предметников, инженеров по знаниям, системных аналитиков и т.д.
- Практически отсутствуют системы программирования БЗ обеспечивающие одновременную, коллективную работу разработчиков над проектами БЗ, что обуславливает актуальность разработки специализированного

алгоритмического и программного обеспечения, в частности, в форме веб-ориентированной программной системы.

- Наличие множества форматов концептуальных моделей и БЗ обуславливает разработку моделей и алгоритмов, обеспечивающих свойство расширяемости разрабатываемой системы с целью создания дополнительных модулей трансформации.
- Сложность и избыточность существующих языков описания трансформаций моделей обуславливают необходимость разработки нового предметно-ориентированного языка для представления и хранения модели (сценария) трансформации концептуальных моделей в БЗ.
- В качестве целевых ЯПЗ, поддерживаемых предлагаемым алгоритмическим и программным обеспечением, выбраны CLIPS и OWL, как наиболее распространенные на данный момент средства представления знаний в виде продукций и онтологий.
- Разработка современных программных систем осуществляется на основе сочетания различных подходов программирования, при этом актуальным является создание веб-ориентированных приложений на основе каркасного подхода.

Глава 2. Модели и методы создания программных компонентов трансформации концептуальных моделей

На основе проведенного аналитического обзора сделан вывод об актуальности разработки новых моделей, методов и средств автоматизации процесса создания специализированных программных компонентов интеллектуальных систем – модулей-конверторов, обеспечивающих проектирование БЗ и синтез их кодов путем трансформации концептуальных моделей предметных областей. Модели, метод и инструментальное средство в совокупности образуют новую информационную технологию [14-38], основными элементами которой являются:

1. Модель типового программного компонента трансформации, представляющего собой каркас (шаблон), который может быть настроен на определенные форматы концептуальной модели и БЗ путем интерпретации блока модели трансформации в форме декларативной программы.
2. Метод создания программных компонентов на основе копирования типового программного компонента и его настройки на основе модели трансформации.
3. Предметно-ориентированный декларативный язык – Transformation Model Representation Language (TMRL), предназначенный для представления и хранения моделей трансформаций и описания механизма взаимодействия с внешними программными компонентами трансформаций.
4. Методика автоматизированной разработки БЗ на основе трансформации концептуальных моделей с использованием программных компонентов.
5. Веб-ориентированная программная система, реализующая предлагаемый метод и язык и представляющая собой прикладное программное обеспечение, обеспечивающее не только поддержку создания программных компонентов трансформации, но и предоставляющее комплексное решение

задачи автоматизированной разработки БЗ путем совместного (коллективного) визуального проектирования и кодогенерации.

В данной работе под программным компонентом понимается отдельная самостоятельная прикладная программа, обеспечивающая автоматизированное создание БЗ на целевом ЯПЗ CLIPS или OWL путем трансформации исходных концептуальных моделей, построенных при помощи различных систем концептуального, когнитивного, онтологического моделирования или CASE-средств и представленных в виде XML-документов.

Основной особенностью веб-ориентированной программной системы является ее расширяемость в части создания и добавления в ее состав новых программных компонентов, обеспечивающих импорт и анализ различных концептуальных моделей и генерацию кода БЗ на ЯПЗ CLIPS или OWL. При этом системой предоставляется специальный интерфейс взаимодействия, при помощи которого внешние программные средства могут обращаться к соответствующим программным компонентам трансформации по части импорта концептуальных моделей и получения кода БЗ на их основе. Также создаваемые программные компоненты могут отчуждаться и использоваться в качестве автономных подсистем в сторонних программных средствах, например, в составе интеллектуальных систем для автоматического получения знаний из концептуальных моделей.

Далее подробнее рассмотрим отдельные элементы разработанной технологии.

2.1. Модель типового программного компонента

Разработку программных компонентов предлагается осуществлять методом копирования типового шаблонного программного компонента для трансформации моделей и его дальнейшей специализации (настройки). При этом типовой компонент может быть рассмотрен как каркас с разъемами/слотами [225, 226],

специализация которого, в свою очередь, осуществляется путем интерпретации модели трансформации в форме декларативной программы, содержащей описание трансформаций и вызываемых блоков модулей анализа и генерации.

Модель типового программного компонента предлагается описать следующим образом:

$$M_{TPC} = \langle M_T, A_{IN}, CG_{OUT}, I \rangle, \quad (1)$$

где M_T – модель трансформации, используемая для специализации программного компонента (слот сценария преобразования); A_{IN} – анализатор (парсер) входных моделей (слот для модуля анализатора); CG_{OUT} – генератор выходных моделей (слот для модуля генератора); I – интерфейс взаимодействия с внешними системами, обеспечивающий доступ к анализатору и генератору.

При этом $A_{IN} \in \{A_{IN}^{CM}, A_{IN}^{ONT}\}$, где A_{IN}^{CM} – анализатор входных концептуальных моделей, представленных в формате XML; A_{IN}^{ONT} – анализатор входной концептуальной модели, представленной в виде модели онтологии [22] (унифицированное представление знаний, не зависящее от какого-либо ЯПЗ, предназначенное для промежуточного хранения полученных знаний из концептуальных моделей; данная модель может выступать как в качестве входной, так и выходной модели; описание данной модели рассмотрено далее).

$CG_{OUT} \in \{CG_{OUT}^{ONT}, CG_{OUT}^{CLIPS-KB}, CG_{OUT}^{OWL-KB}\}$, где CG_{OUT}^{ONT} – генератор выходной модели онтологии; $CG_{OUT}^{CLIPS-KB}$ – генератор кода БЗ на ЯПЗ CLIPS; CG_{OUT}^{OWL-KB} – генератор кода БЗ на ЯПЗ OWL.

$I = \{i_1, \ldots, i_n\}, i_j = \langle name_j, command_j \rangle, j \in \overline{1,n}$, где $name_j$ – наименование j метода взаимодействия; $command_j$ – управляющая команда метода взаимодействия. Данный интерфейс взаимодействия позволяет внешним программным (интеллектуальным) системам обращаться к программному компоненту (по средствам запроса) с целью автоматического создания БЗ на определенном ЯПЗ путем передачи (импорта) концептуальной модели.

Таким образом, в процессе создания программного компонента путем специализации модели типового программного компонента M_{TPC}, необходимо сформировать модель трансформации M_T, которая определяет правила преобразования, формирующие сценарий трансформации исходных концептуальных моделей в целевые БЗ, а также определить (вставить в слоты) соответствующие блоки анализа и генерации, из которых произойдет сборка конкретного программного компонента.

2.1.1. Модель трансформации

Основным элементом (ядром) модели типового программного компонента M_{TPC} является модель трансформации M_T.

В данной работе модель трансформации M_T описывает преобразование моделей [14], обобщенная концептуальная схема которой представлена на Рисунке 2.

Под трансформацией модели понимается автоматическая генерация целевой модели из исходной модели в соответствии с некоторым набором правил трансформации, которые в совокупности описывают, как модель на исходном языке может быть преобразована в модель на целевом языке. Следовательно, каждое отдельное правило трансформации – это описание того, как одна или более конструкций на исходном языке могут быть преобразованы в одну или несколько конструкций в целевом языке.

Следует отметить, что трансформация модели также применима к нескольким исходным и/или целевым моделям. Классическим примером первого варианта является слияние моделей, т.е. объединение нескольких исходных моделей, которые были разработаны пользователями параллельно, в одну целевую модель. Примером второго варианта является преобразование, которое занимает центральное место в модельно-управляемом подходе (MDE/MDD/MDA)

[89-101], а именно: преобразование одной общей платформно-независимой модели (Platform Independent Model, PIM) в ряд платформно-зависимых моделей (Platform Specific Model, PSM) для различных конкретных платформ.

Таким образом, используя (1), определим структуру модели трансформации M_T:

$$M_T = \langle MM_{IN}, MM_{OUT}, T \rangle, \quad (2)$$

где MM_{IN} – метамодель исходной (входной) концептуальной модели; MM_{OUT} – метамодель целевой (выходной) модели представления знаний (модели БЗ); T – оператор преобразования моделей.

Использование метамоделирования является одним из основных подходов к определению абстрактного синтаксиса языков, в том числе концептуальных языков моделирования и ЯПЗ.

При этом:

$$MM_{IN} \in \{MM_{CM}, MM_{PR}, MM_{ONT}\}, \quad (3)$$
$$MM_{OUT} \in \{MM_{PR} MM_{ONT}, MM_{CLIPS}, MM_{OWL}\},$$

где MM_{CM} – метамодель концептуальной модели CM_{XML}; MM_{PR} – метамодель обобщенной модели продукций M_{PR}; MM_{ONT} – метамодель обобщенной модели онтологии M_{ONT}; MM_{CLIPS} – метамодель языка CLIPS; MM_{OWL} – метамодель языка OWL.

Исходя из (3), следует, что помимо метамоделей MM_{CM}, MM_{CLIPS} и MM_{OWL}, которые необходимы для описания преобразования исходных концептуальных моделей в целевые БЗ в формате CLIPS или OWL, в модель трансформации также могут входить метамодели MM_{PR} и MM_{ONT}, описывающие некоторые обобщенные модели продукций M_{PR} и онтологии M_{ONT}. Данные модели позволяют абстрагироваться от особенностей описания знаний на различных ЯПЗ, которые используются при реализации

(программировании) БЗ. Таким образом, они представляют собой высокоуровневые абстракции (спецификации), предназначенные для унифицированного представления и хранения знаний, извлеченных из концептуальных моделей, и, в свою очередь, могут выступать как в качестве исходной концептуальной модели для генерации кода БЗ, так и целевой модели БЗ. Подробное описание модели продукций и онтологии приводится в следующих подразделах.

При этом следует отметить, что метамодели MM_{PR}, MM_{ONT}, MM_{CLIPS}, MM_{OWL} должны быть предоставлены пользователю по умолчанию для создания модели трансформации M_T.

Используя (2), рассмотрим подробнее элементы M_T. Для этого, исходя из (3), опишем элементы MM_{CM}:

$$MM_{CM} = \langle E_{CM}, R_{CM} \rangle,$$

где E_{CM} – множество элементов метамодели исходной концептуальной модели; R_{CM} – множество отношений между элементами метамодели исходной концептуальной модели.

При этом:

$E_{CM} = \{e_1^{cm},\ldots,e_n^{cm}\}, e_i^{cm} = \langle name_{i,1}, p_{i,2},\ldots,p_{i,k} \rangle, i \in \overline{1,n}$, где $name_{i,1}$ – наименование i элемента; $p_{i,2},\ldots,p_{i,k}$ – остальные свойства i элемента.

Таким образом, предполагается, что у каждого элемента MM_{CM} должно быть наименование, а также произвольный набор других свойств (атрибутов).

В MM_{CM} определено три типа отношений:

$$R_{CM} = \langle R_{AS}^{cm}, R_{ID}^{cm}, R_{part-of}^{cm} \rangle,$$

где R_{AS}^{cm} – ассоциация – бинарное отношение между двумя элементами метамодели; R_{ID}^{cm} – связь по идентификатору – бинарное отношение между

двумя элементами метамодели по одному идентификатору (числовому или текстовому значению); $R^{cm}_{part-of}$ – связь «часть-целое» (композиция) – n-арное отношение между n элементами метамодели.

При этом:

$R^{cm}_{AS} = \{r^{as}_1,\ldots,r^{as}_m\}, r^{as}_j = \langle e^{cm}_{j,1}, e^{cm}_{j,2}\rangle, j \in \overline{1,m}$, где $e^{cm}_{j,1}$ – левая часть отношения типа ассоциация (левый элемент метамодели), $e^{cm}_{j,2}$ – правая часть отношения типа ассоциация (правый элемент метамодели).

$R^{cm}_{ID} = \{r^{id}_1,\ldots,r^{id}_h\}, r^{id}_l = \langle r^{id}_{lhs_l}, r^{id}_{rhs_l}\rangle, l \in \overline{1,h}$, где $r^{id}_{lhs_l}$ – левая часть связи; $r^{id}_{rhs_l}$ – правая часть связи. В свою очередь $r^{id}_{lhs_l} = e^{cm}_i(p_i), i \in \overline{1,n}$ и $r^{id}_{rhs_l} = e^{cm}_j(p_j), j \in \overline{1,n}$, где $e^{cm}_i(p_i)$ и $e^{cm}_j(p_j)$ – элементы метамодели участвующие в отношении по идентификатору, т.е. значение свойства p_j равно значению свойства p_i.

Отношение «часть-целое» предполагает, что i элемент метамодели e^{cm}_i разбит на составные части, т.е. $e^{cm}_i = \sum_{j=1}^{n}\left(e^{cm}_{i,j}\right), i \in \overline{1,k}$, которые в свою очередь связываются (соединяются) между собой отношением R^{cm}_{AS} или R^{cm}_{ID}:

$$R^{cm}_{part-of} = \{r^{part-of}_1,\ldots,r^{part-of}_g\}, r^{part-of}_f = \left\langle (r^{part-of}_{lhs_f}, r^{part-of}_{rhs^1_f}),\ldots,(r^{part-of}_{lhs_f}, r^{part-of}_{rhs^u_f})\right\rangle, f \in \overline{1,g}$$

Описание элементов MM_{PR}, MM_{ONT}, MM_{CLIPS}, MM_{OWL} приводится в следующих подразделах.

Для представления и хранения данных метамоделей разработана специальная мета-метамодель (уровень M3, в архитектуре метамоделирования MDA, концептуальная схема которой представленная на Рисунке 2),

определяющая концептуальное пространство моделирования (Conceptual Modeling Space).

Данная мета-метамодель призвана служить мостом между разными метамоделями, определенными в данной работе, поскольку представляет собой основу для их описания. Если две разные метамодели соответствуют этой мета-метамодели, т.е. могут быть описаны средствами «М3», то все конкретные модели уровня «М1», базирующиеся на них, могут храниться в общем репозитории и совместно обрабатываться средствами модельных трансформаций.

Разработанная мета-метамодель, описывающая основные мета-сущности, приводится на Рисунке 3. Данная мета-метамодель является упрощением спецификации Ecore. Полное описание языка метамоделирования Ecore (abstract syntax) приводится в [124].

Для построения данной мета-метамодели, а также остальных метамоделей (MM_{CM} , MM_{PR} , MM_{ONT} , MM_{CLIPS} , MM_{OWL}), использовалось платформа Eclipse Modeling Framework (EMF) [136] и программное средство Graphical Modeling Framework (GMF) [231], а также нотация Ecore [124].

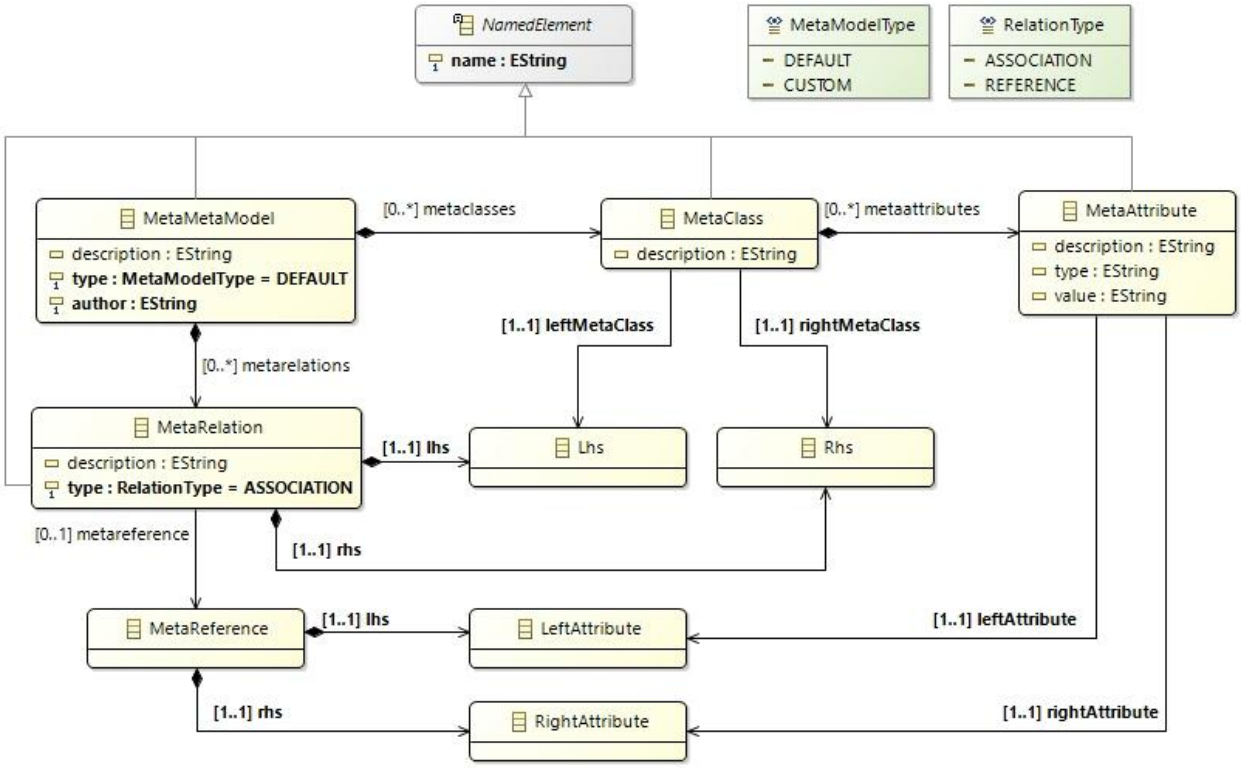

Рисунок 3. Мета-метамодель

Основным элементом модели трансформации M_T является оператор преобразования T:

$$T : CM \to KB, \tag{4}$$

где CM – исходная концептуальная модель; KB – целевая БЗ.

Для уточнения (4) выделим виды синтаксиса концептуальных языков моделирования и ЯПЗ согласно [232]:

1. Абстрактный синтаксис (abstract syntax) – определяет важнейшие концепции и структуру выражений языка, т.е. характеризует в абстрактной форме виды элементов, из которых состоит язык, и правила их комбинирования.
2. Конкретный синтаксис (concrete syntax) – определяет, как языковые элементы проявляются в конкретной используемой человеком форме (текстовой или графической), т.е. конкретный синтаксис является нотацией. Абстрактный синтаксис напрямую связан с семантикой, а конкретный синтаксис – с абстрактным синтаксисом.
3. Синтаксис сериализации или служебный синтаксис (serialization syntax) – подобен конкретному синтаксису, за исключением того, что он не обязан быть воспринимаемым человеком, т.е. данный синтаксис используется для хранения и обмена языковыми выражениями в сериализованной (последовательной) форме между различными программными средствами.

Специализируем (4) на абстрактном и конкретном (служебном) уровне:

$$T_{AS} : MM_{CM} \to MM_{KB},\ T_{CS} : CM_{XML} \to Code_{KRL}, \tag{5}$$

где T_{AS} – оператор преобразования исходной концептуальной модели в целевую БЗ на абстрактном уровне с использованием метамоделей; MM_{CM} – метамодель исходной концептуальной модели; MM_{KB} – метамодель целевой БЗ; T_{CS} – оператор преобразования исходной концептуальной модели в целевую БЗ на конкретном (служебном) уровне; CM_{XML} – исходная концептуальная модель, представленная в формате XML; $Code_{KRL}$ – код БЗ, представленной на целевом

ЯПЗ (Knowledge Representation Language, KRL). При этом $Code_{KRL} \in \{Code_{CLIPS}, Code_{OWL}\}$, где $Code_{CLIPS}$ – целевой код БЗ на ЯПЗ CLIPS; $Code_{OWL}$ – целевой код БЗ на ЯПЗ OWL.

Используя (4), определим типы операторов преобразования исходя из (3):

$$T \in \{T_{CM-KB}, T_{CM-UM}, T_{UM-KB}\}, \quad (6)$$

где T_{CM-KB} – оператор преобразования исходной концептуальной модели в код БЗ на целевом ЯПЗ; T_{CM-UM} – оператор преобразования исходной концептуальной модели в унифицированный формат представления знаний в виде модели онтологии или продукций; T_{UM-KB} – оператор преобразования модели онтологии или продукций в код БЗ на целевом ЯПЗ CLIPS или OWL соответственно.

Детализируем (6) для более подробного описания трансформации, исходя из (5):

$$T_{CM-KB} : MM_{CM} \rightarrow MM_{KB},$$

$$T_{CM-UM} : MM_{CM} \rightarrow MM_{UM},\ T_{UM-KB} : MM_{UM} \rightarrow MM_{KB},$$

где $MM_{KB} \in \{MM_{CLIPS}, MM_{OWL}\}$, $MM_{UM} \in \{MM_{PR}, MM_{ONT}\}$.

Согласно (5), процесс трансформации CM_{XML} в $Code_{KRL}$ осуществляется путем установления соответствий (определение правил трансформации) между абстрактными элементами исходной метамодели MM_{CM} и целевой метамодели MM_{KB}.

При этом можно выделить три типа таких соответствий:

1. Взаимно однозначное соответствие (эквивалентность) – каждому элементу MM_{CM} можно сопоставить ровно один элемент MM_{KB}.
2. Неоднозначное соответствие (синонимия) – несколько элементов MM_{CM} соответствуют одному элементу MM_{KB}.

3. Неразличимое соответствие (омонимия) – один элемент MM_{CM} соответствует сразу нескольким элементам MM_{KB}.

Крайние случаи:

1. Избыточность – отдельные элементы MM_{CM} не могут быть отражены ни в один элемент MM_{KB}.
2. Дефицит выразительной способности – отдельные элементы MM_{KB} не имеют соответствующего им элемента MM_{CM}.

Далее рассмотрим построение правил трансформации для данных типов соответствий.

Для этого введем специальный оператор R_T для обозначения набора правил соответствия элементов метамоделей (правил трансформации):

$$R_T = (r_1, r_2 .. r_n) : MM_{CM} \to MM_{KB},$$

где r_i – правило трансформации.

Правило трансформации может быть двух видов:

$$r_i \in \left\{ r_i^s, r_i^c \right\}, i \in \overline{1,n},$$

где r_i^s – простое бинарное правило трансформации; r_i^c – комплексное (сложное) n-арное правило трансформации.

$$r_i^s = \left\langle e_i^{in}, e_i^{out}, p_i \right\rangle, i \in \overline{1,n},$$

где e_i^{in} – исходный элемент MM_{CM} (левая часть правила); e_i^{out} – целевой элемент MM_{KB} (правая часть правила); p_i – приоритет выполнения правила, определяющий последовательность выполнения правил, $p_i \in \overline{1,k}$.

$$r_i^c = \left\langle \left\{ e_{i,1}^{in}, \ldots, e_{i,j}^{in} \right\}, e_i^{out}, p_i \right\rangle, i \in \overline{1,n},$$

где $\left\{ e_{i,1}^{in}, \ldots, e_{i,j}^{in} \right\}$ – множество исходных элементов MM_{CM} (левая часть правила); e_i^{out} – целевой элемент MM_{KB} (правая часть правила); p_i – приоритет

выполнения правила, определяющий последовательность выполнения правил, $p_i \in \overline{1,k}$.

Существование r_i^c обуславливается третьим видом отношений ($R_{part-of}^{cm}$), которые могут быть в MM_{CM}. Поэтому можно сказать, что элемент MM_{KB} формируется на основе разных частей одного исходного понятия из MM_{CM}.

Данные виды правил трансформации применимы ко всем трем типам соответствий.

2.1.2. Обобщенная модель онтологии

Для унифицированного представления и хранения знаний, извлеченных из концептуальных моделей, предлагается использовать специальную модель онтологии M_{ONT}. Данная модель позволяет абстрагироваться от особенностей описания знаний в различных ЯПЗ, используемых при реализации БЗ (например, CLIPS, JESS, Drools, RuleML, SWRL, OWL, RDF и др.), и хранить знания в собственном независимом формате.

Данная модель является высокоуровневой абстракцией представления знания в виде онтологий (средство для описания понятийного знания) и содержит только общие конструкции, разделяемые большинством онтологических языков.

Формализуем описание метамодели для модели онтологии, используя представление, предложенное в [233]:

$$MM_{ONT} = \langle DT, CN, PN, Obj, R \rangle,$$

где DT – перечень базовых типов данных, при этом $DT = \{\text{литерал, объект, коллекция}\}$; CN – имена классов; PN – имена свойств классов; Obj – понятия (константы, объекты) предметной области; R – конечное множество отношений между концептами заданной предметной области, при этом:

$R \in \{R^{subclass\text{-}of}, R^{is-a}, R^{p}, R^{c}\}$, где $R^{subclass\text{-}of}$ – отношение наследования между классами CN; R^{is-a} – отношение между классами CN и их экземплярами (объектами) Obj; R^{p} – отношение между классами и свойствами, $pn\, R^{p} \langle cn, cn\_dt \rangle$, где $pn \in PN$, $cn \in CN$, $cn\_dt \in CN \cup DT$, последнее означает, что свойство с именем pn_i характеризует класс cn_j, а значениями свойства могут быть элементы типа другого класса из CN или основного множества типов DT; R^{c} – отношения между классами, при этом $R^{c} = \{r_1^{c} .. r_k^{c}\}$, $r_j^{C} = \langle name_j, cn_{LHS_j}, cn_{RHS_j} \rangle$, $j \in \overline{1,k}$, где $name_j$ – имя отношения; cn_{LHS_j} – левый класс отношения (ссылка на класс); cn_{RHS_j} – правый класс отношения (ссылка на класс).

Метамодель MM_{ONT} (abstract syntax), определяющая основные концепты, из которых состоит данная модель онтологии, представлена на Рисунке 4.

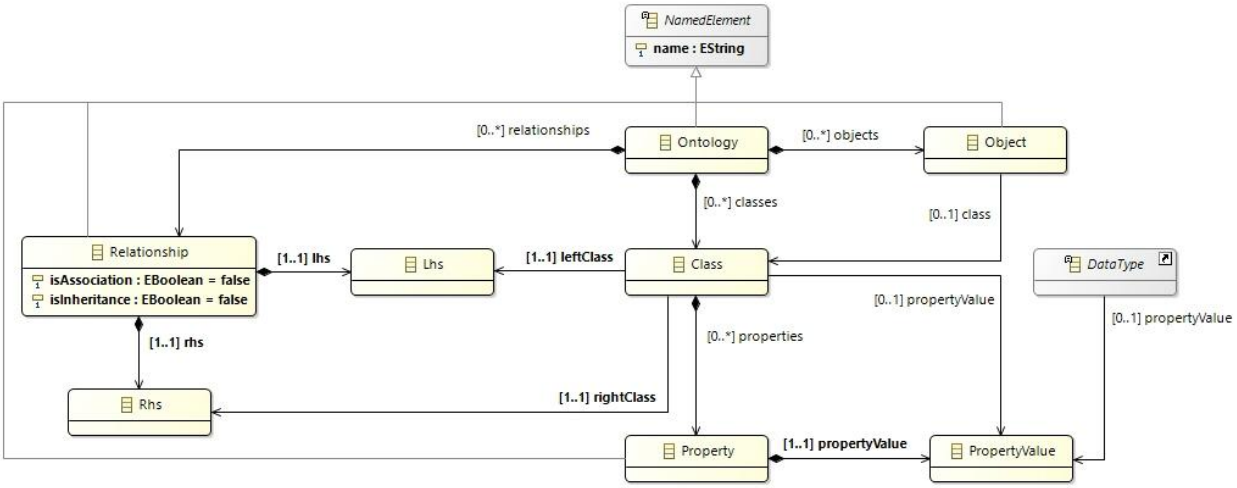

Рисунок 4. Метамодель, описывающая модель онтологии

2.1.3. Обобщенная модель продукций

Для поддержки возможности работы со знаниями в виде правил разработана специальная модель продукций M_{PR}.

Данная модель является высокоуровневой абстракцией представления знания в виде продукций и содержит только общие конструкции, разделяемые большинством языков продукционных (логических) правил.

Таким образом, модель продукций позволяет представить знание в виде предложений типа «Если (условие), то (действие)», тем самым обеспечивая унифицированное представление и хранение продукционных БЗ. Для представления модели продукций использована специализированная графическая нотация – Rule Visual Modeling Language (RVML) [13, 234].

RVML – язык визуального моделирования правил, предназначенный для моделирования и описания логических правил, обладающий большей выразительностью при описании причинно-следственных зависимостей по сравнению с UML, в частности, RVML, позволяет:

- использовать отдельные графические примитивы для отображения всех элементов продукций, а не стереотипы или типизированные классы как в UML;
- присваивать отдельным фактам субъективные вероятности в виде коэффициентов уверенности;
- более наглядно отображать тип выполняемых действий, таких как добавление, изменение, удаление;
- отображать логические операторы в условиях правил («И», «ИЛИ» и «НЕ»).

Метамодель MM_{PR} (abstract syntax), определяющая основные концепты, из которых состоит данная модель продукций, представлена на Рисунке 5.

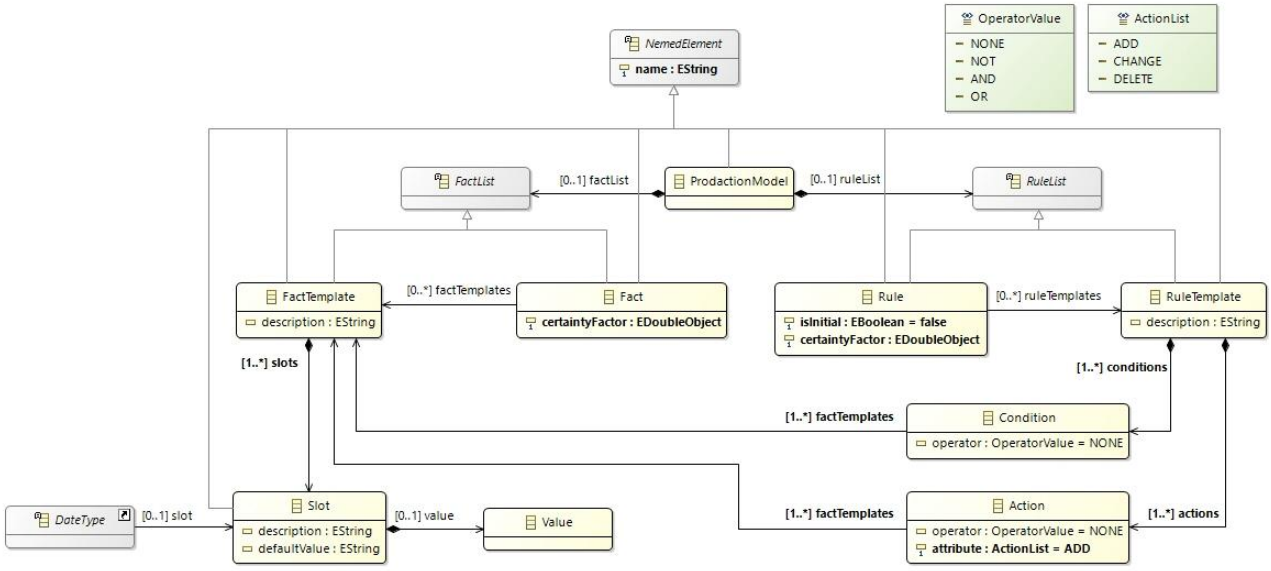

Рисунок 5. Метамодель, описывающая модель продукций

2.1.4. Метамодель OWL

На основе результатов аналитического обзора современных методов и средств представления и обработки знаний сделан вывод об актуальности использования в качестве целевого ЯПЗ для описания онтологий OWL2 DL [142] с поддержкой синтаксиса RDF/XML [211].

Метамодель OWL2 (abstract syntax), определяющая основные концепты (семантику), из которых состоит данный язык, представлена на Рисунке 6 и 7.

Метамодель OWL2, соответствующая техническому пространству моделирования MOF (MOF Technical Space) [95] и представленная в формате Ecore [124], определена в [235]. Данная метамодель также представлена в формате XSD [176] и определена в [235].

Полное описание абстрактного синтаксиса (abstract syntax) OWL2 DL приводится в Приложении А.

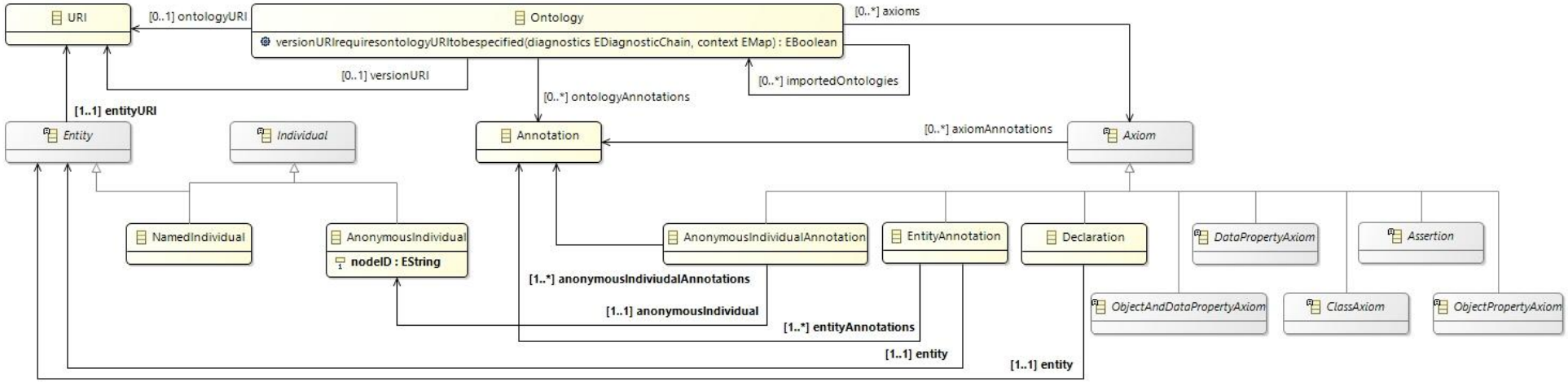

Рисунок 6. Метамодель OWL2 DL (основные концепты онтологии)

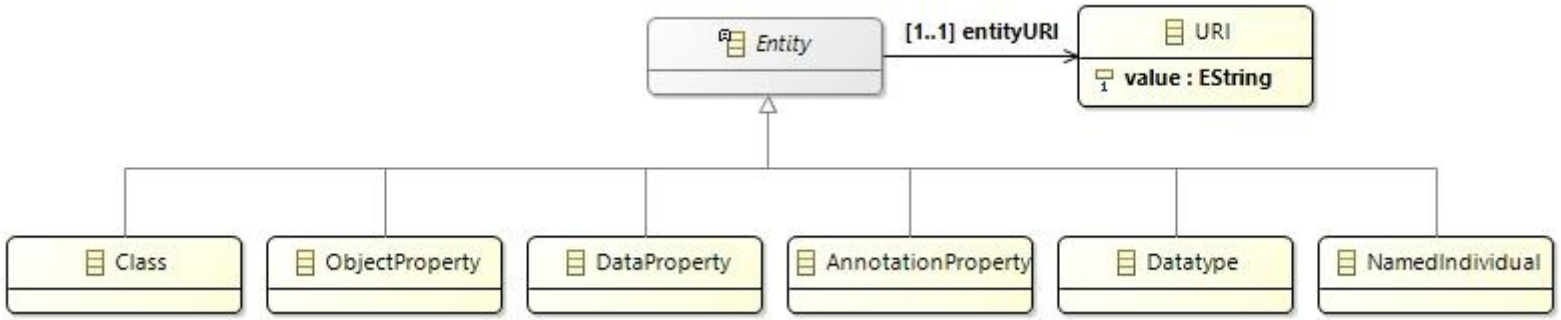

Рисунок 7. Метамодель OWL2 DL (основные сущности)

2.1.5. Метамодель CLIPS

В дополнение к OWL2 в качестве второго целевого ЯПЗ для описания продукций (правил) предлагается использовать CLIPS [54].

Метамодель CLIPS (abstract syntax), определяющая основные концепты (семантику), из которых состоит данный язык, представлен на Рисунках 8-10. Следует отметить, что в данной работе не используется язык COOL, возможность его применения планируется осуществить в будущих исследованиях, поэтому метамодель содержит только основные определения и конструкторы CLIPS.

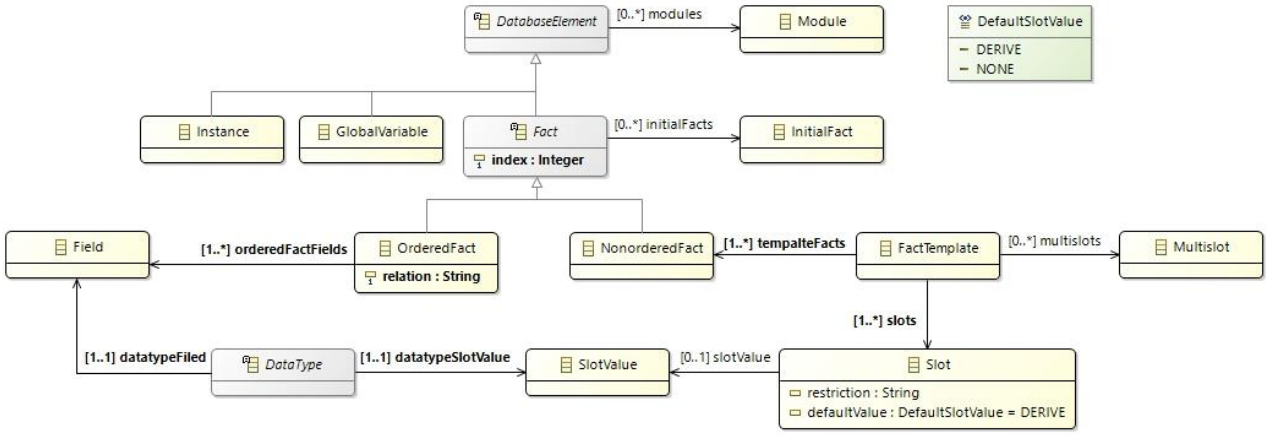

Рисунок 8. Метамодель CLIPS (данные)

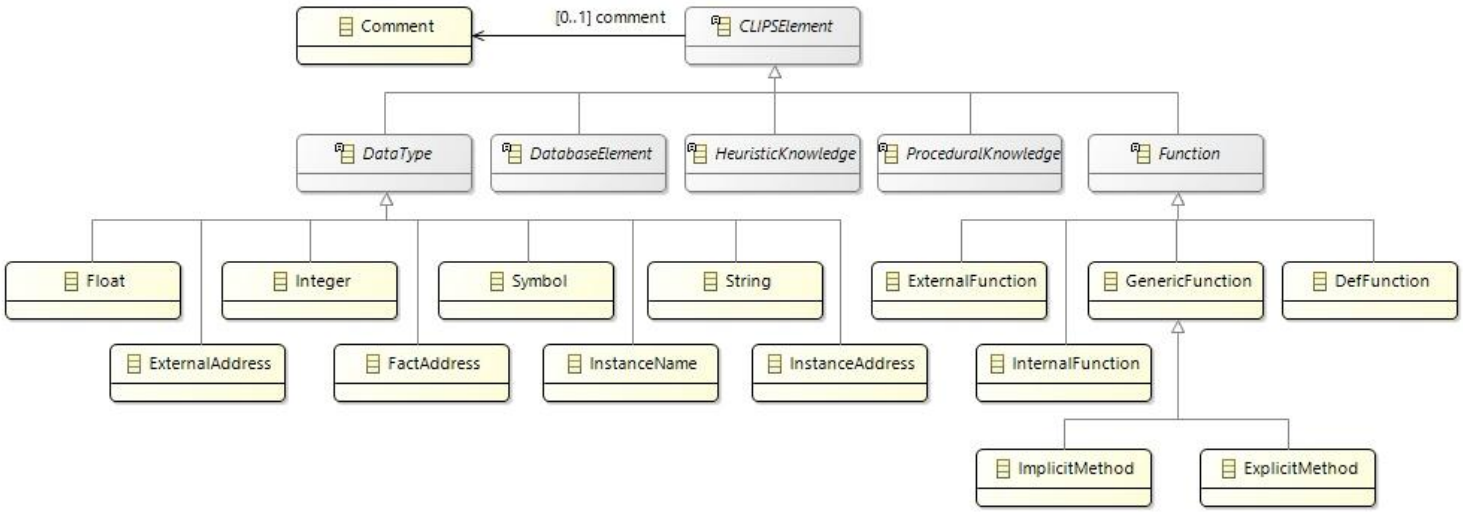

Рисунок 9. Метамодель CLIPS (иерархия основных элементов)

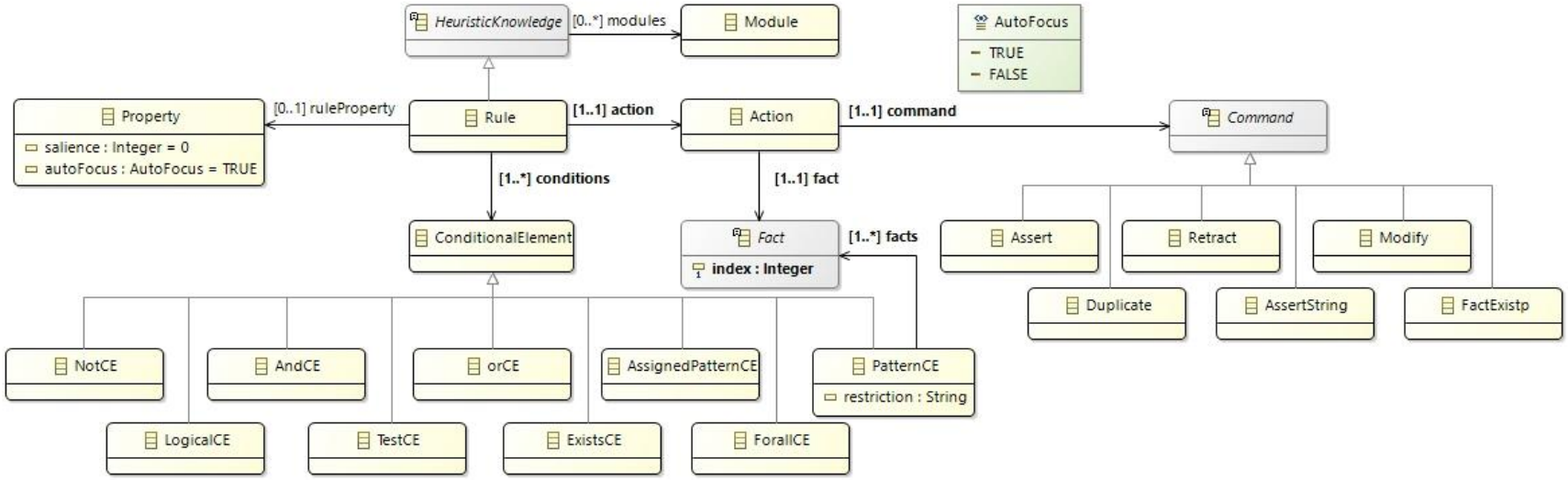

Рисунок 10. Метамодель CLIPS (эвристическое представление знаний)

2.2. Метод создания программных компонентов

Предложенная модель типового программного компонента и концепция использования языков модельных трансформаций [119, 126] определяют особенности метода создания программных компонентов [14, 16-20]. Метод представляет собой систематизированную совокупность действий, которые нацелены на решение задачи автоматизированной разработки программного компонента на основе копирования и настройки типового программного компонента трансформации (специализация модели типового программного компонента M_{TPC} путем формирования модели трансформации M_T, а также выбора соответствующих блоков анализа и генерации).

Особенностью предлагаемого метода создания программных компонентов, определяющими его новизну, является использование языков модельных трансформаций для описания M_T в совокупности с оригинальной моделью типового программного компонента M_{TPC}. При этом трансформация моделей определяется на абстрактном уровне (abstract syntax), понятном пользователю (разработчику программного компонента). В связи с этим, необходимо сформулировать ограничения на входные и выходные метамодели, а также определить некоторые допущения:

1. Метамодель исходной концептуальной модели MM_{CM} может быть представлена в формате XML-схемы – XML Schema Definition (XSD) [176]. Для автоматического построения XML-схем разработаны многочисленные паттерны (шаблоны) проектирования, наиболее распространенными из которых являются: Russian Doll, Salami Slice, Venetian Blind, Garden of Eden. Данные паттерны различаются количеством используемых глобальных элементов и типов. Предполагается, что использование паттернов Venetian Blind и Garden of Eden [236] является наиболее перспективным.

2. Метамодель исходной концептуальной модели MM_{CM} может быть также получена на основе процедуры обратной инженерии (reverse engineering) [127] путем анализа исходных концептуальных моделей и извлечением из них элементов.
3. Сгенерированная метамодель исходной концептуальной модели MM_{CM} может быть семантически некорректной, если исходный XML-документ концептуальной модели была составлен неверно (например, если в исходной концептуальной модели наименования тегов отражают семантику предметной области – уровня модели «M1», а не метамодели – уровня «M2»).
4. Не все связи элементов метамодели исходной концептуальной модели MM_{CM} могут быть получены с использованием XML-схем или процедуры обратной инженерии, а именно: связь «по идентификаторам» R_{ID}^{cm}, установленная путем внутренней индексации элементов; связь «часть-целое» $R_{part-of}^{cm}$, установленная путем переноса части семантики элемента в дочерние элементы.
5. Не все элементы XML-схемы могут быть однозначно отображены в элементы метамодели исходной концептуальной модели.
6. Не все элементы метамодели исходной концептуальной модели могут быть однозначно отображены в элементы целевой метамодели БЗ (проблема избыточности и дефицита выразительной способности).
7. Метамодели целевых моделей БЗ в виде онтологии и продукций доступны для построения модели трансформации M_T по умолчанию.
8. Метамодели целевых БЗ, представленных на ЯПЗ CLIPS и OWL, доступны для построения модели трансформации M_T по умолчанию.

Введение данных ограничений, с одной стороны, позволяет понизить сложность разработки программных компонентов, включая модели трансформаций, за счет максимальной автоматизации данного процесса, а с

другой – учесть их при реализации данного метода с целью недопущения некоторых ошибок.

Таким образом, основные этапы создания программных компонентов на основе модели типового программного компонента M_{TPC}, включая создание модели трансформации M_T, представлены на Рисунке 11.

Подробнее рассмотрим этапы создания программного компонента:

На этапе 1 осуществляется создание метамодели для исходной концептуальной модели. Данная метамодель может быть сформирована как на основе анализа XML-схемы (XSD-файла), описывающая структуру входного XML-документа, так и на основе анализа исходной концептуальной модели.

На этапе 2 осуществляется выбор целевой метамодели БЗ. Это может быть либо метамодель обобщенной модели онтологии (Рисунок 4) или продукций (Рисунок 5), либо метамодель OWL (Рисунки 6, 7) или CLIPS (Рисунки 8-10). Выбор целевой метамодели БЗ влияет на определение соответствующих блоков генераторов CG_{OUT} при автоматическом создании (сборке) программного компонента на 4 этапе.

На этапе 3 создается модель (сценарий) трансформации M_T.

На подготовительном этапе 3.1 осуществляется анализ структуры исходной и целевой метамодели, при этом извлекаются соответствующие элементы.

На этапе 3.2 на основе выделенных элементов исходной и целевой метамодели автоматически формируется модель трансформации M_T, содержащая только наборы элементов исходной и целевой метамодели (MM_{CM} и MM_{KB}) и не включающая правила трансформации T.

Далее, на этапе 3.3 определяются правила трансформации T путем установления соответствий между исходными элементами метамодели MM_{CM} и целевыми элементами метамодели MM_{KB}.

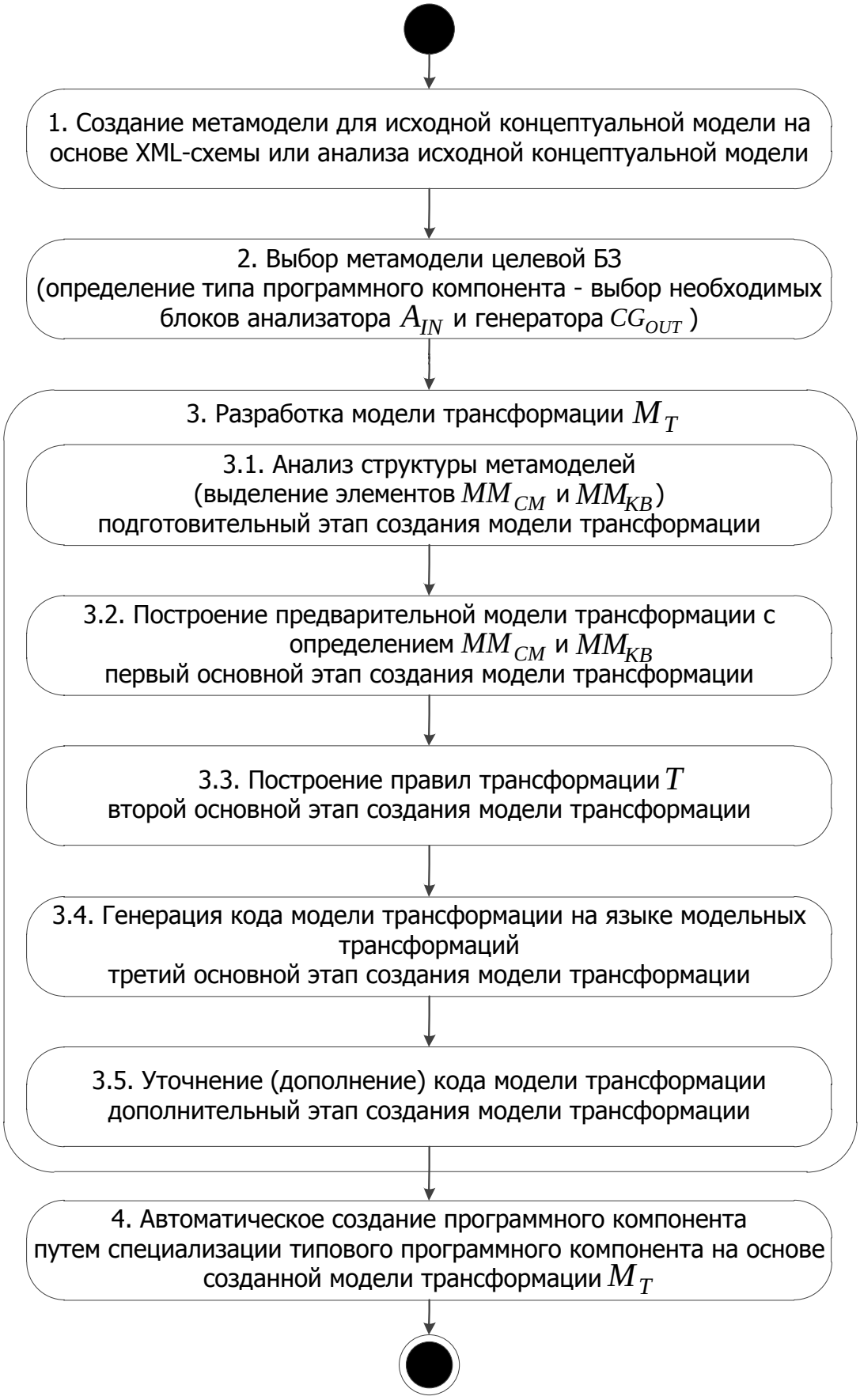

Рисунок 11. Этапы создания программного компонента для трансформации концептуальных моделей в БЗ

После чего на этапе 3.4 автоматически генерируется модель трансформации M_T на языке модельных трансформаций. Необходимо отметить, что данная модель трансформации может быть неполной или некорректной по двум основным причинам:

1. Неполнота представления исходной модели: не все связи элементов метамодели исходной концептуальной модели могут быть получены с использованием XML-схем или процедуры обратной инженерии. Данная причина обусловлена особенностями механизма установления связи между элементами концептуальной модели. Например, связь элементов может осуществляться путем указания ссылок на идентификаторы элементов, как показано в Листинге 1.

Листинг 1. Пример связи элементов по идентификатору

```
1  <DataType id="номер типа данных"… />
2  <Attribute type="номер типа данных" …> … </UML:Attribute>
```

Связь элементов, установленная с использованием уникальных идентификаторов, не будет отображена на XML-схеме (данную связь также невозможно извлечь процедурой обратной инженерии), вследствие чего при автоматической генерации модель трансформации M_T будет неполной (некорректной). Таким образом, это является одним из недостатков использования как XML-схем в качестве исходных входных метамоделей, так и процедуры обратной инженерии. Эта проблема решается путем установления в метамодели данного типа связи вручную.

2. Неоднозначность интерпретации исходной концептуальной модели: не все элементы метамодели исходной концептуальной модели MM_{CM} могут быть однозначно интерпретированы и отображены в элементы метамодели целевой БЗ MM_{KB}. Возможны ситуации, когда в качестве значения целевого элемента метамодели MM_{KB} используется некоторая текстовая последовательность (константа), значение которой, в свою очередь, может зависеть от значений исходных элементов метамодели MM_{CM}.

Исходя из этого, правила трансформации элементов можно разделить на два типа:

1. простые правила определения (биективные правила) – задают однозначные соответствия между исходными и целевыми элементами (их значениями) метамоделей (данные правила определяются на этапе 3.3);
2. сложные (составные) правила определения – задают значения (константы) целевых элементов метамодели БЗ, исходя из соответствующих значений исходных элементов метамодели концептуальной модели.

Таким образом, установленные правила трансформации исходных и целевых элементов метамоделей (этап 3.3) могут быть недостаточными для полного определения модели трансформации M_T.

Для решения данной проблемы на этапе 3.5 модель трансформации M_T уточняется, т.е. определяются недостающие правила трансформации (сложные правила определения), которые не были установлены на этапе 3.3. Результатом данного этапа является окончательная модель трансформации M_T, описывающая преобразование исходной концептуальной модели в целевую БЗ.

На этапе 4 автоматически создается программный компонент путем специализации типового программного компонента на основе сформированной модели трансформации M_T и выбранных блоках анализатора A_{IN} и генератора CG_{OUT}.

Разработанный в соответствии с данным методом программный компонент должен обеспечивать генерацию кода БЗ на целевом ЯПЗ CLIPS или OWL путем трансформации исходных концептуальных моделей.

Как отмечалось выше, метод основан на концепции трансформации моделей, которая является одним из важнейших аспектов в области модельно-управляемого подхода (MDD/MDA) [89-101].

Таким образом, используя концептуальную схему трансформации моделей, представленной на Рисунке 2, определим общую схему трансформации концептуальных моделей в БЗ (Рисунок 12).

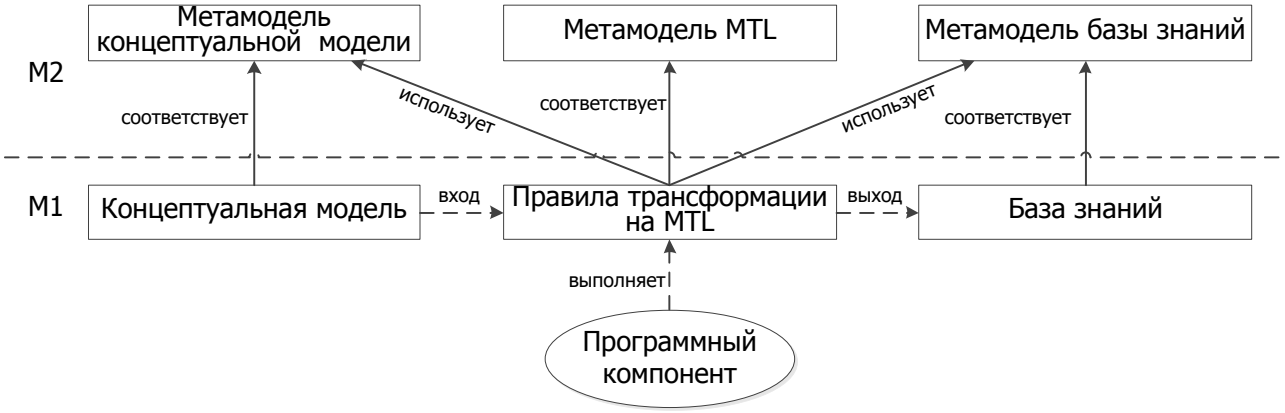

Рисунок 12. Трансформация концептуальных моделей в БЗ

Подробнее рассмотрим данную схему трансформации:

Схема определяет горизонтальную экзогенную трансформацию [127], т.е. преобразование исходной и целевой модели одного уровня иерархии, но описанные на различных языках моделирования (разные концептуальные языки моделирования и ЯПЗ).

Схема соответствует четырехуровневой архитектуре метамоделирования (иерархии моделей) MOF [123] и содержит два основных уровня: моделей «M1» и метамоделей «M2». Однако, для упрощения восприятия в ней отсутствует уровень мета-метамодели «M3» и уровень данных «M0». При этом описание мета-метамодели приводится в пункте 2.1.1 модели трансформации.

На уровне «M1» находятся исходная концептуальная модель, которую необходимо преобразовать в целевой код БЗ, представленных на ЯПЗ CLIPS или OWL, и целевая модель БЗ. Несмотря на то, что осуществляется трансформация моделей, начальными и конечными артефактами является текст (XML-документ концептуальной модели и программный код БЗ CLIPS или OWL). Следовательно, первоначальной задачей является преобразование текста исходного XML-документа концептуальной модели в полноценную модель (перевод конкретной спецификации в абстрактную спецификацию, T2M-трансформация). При этом полученная исходная модель должна соответствовать некоторой метамодели на уровне «M2». Для преобразования полученной исходной концептуальной модели в модель БЗ необходимо определить правила трансформации (M2M-

трансформация). Эти правила, в данном случае, содержатся в модели трансформации M_T, которая соответствует описанию (2) и представляется с использованием языка модельных трансформаций и соответствует его метамодели (Рисунок 2), а целевая модель БЗ соответствует некоторой метамодели БЗ. После того, как будет получена целевая модель БЗ, её необходимо преобразовать в код на ЯПЗ CLIPS или OWL (перевод абстрактной спецификации в конкретную спецификацию, M2T-трансформация). T2M и M2T осуществляется в автоматическом режиме.

В процессе генерации кода БЗ может быть использована модель онтологии или модель продукции, которая представляется либо как исходная концептуальная модель, на основе которой синтезируется код БЗ, либо как целевая модель представления знаний (модель БЗ), полученная путем автоматизированного анализа исходных концептуальных моделей. Полное описание данных моделей приводится в пункте 2.1.2 и 2.1.3 соответственно.

2.3. Предметно-ориентированный язык представления модели трансформации

Для представления и хранения модели трансформации M_T разработан специальный предметно-ориентированный язык (Domain Specific Language, DSL) – Transformation Model Representation Language (TMRL) [14, 16-18, 20]. Грамматика TMRL принадлежат к классу контекстно-свободных грамматик (КС-грамматик – LL(1)) [237, 238]. Конструкции TMRL позволяют в декларативном виде описывать элементы модели трансформации, в частности, правила соответствия элементов метамоделей R_T, а также механизм взаимодействия с ранее разработанными (внешними) программными компонентами трансформации. Созданные на TMRL спецификации удовлетворяют требованиям точности, понятности и полноты [228], т.е. в спецификациях на TMRL содержится

вся необходимая (в рамках предложенного метода) информация для решения поставленной задачи, все объекты модели хорошо формализованы, при этом спецификации достаточно компактны и в то же время понятны (читабельны).

Метамодель TMRL (abstract syntax), определяющая основные концепты (семантику), из которых состоит данный язык, представлена на Рисунке 13.

Полное описание синтаксиса (concrete syntax) TMRL с использованием Расширенной Формы Бэкуса-Наура (нотация РБНФ) [239] приводится в Приложении Б.

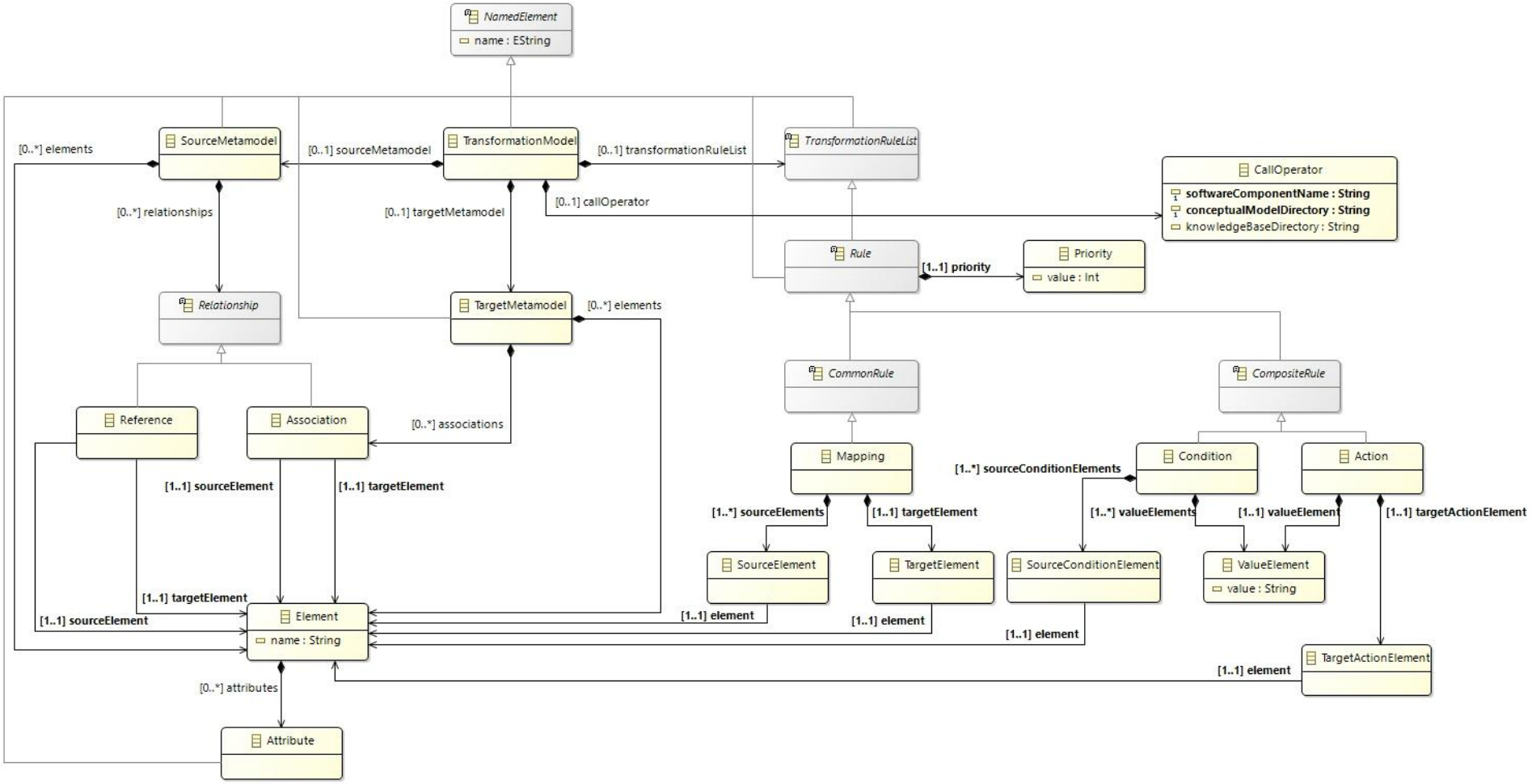

Рисунок 13. Метамодель TMRL

Приведем общую структуру модели трансформации M_T, представленной на TMRL (элементы языка выделены жирным шрифтом), в Листинге 2.

Листинг 2. Общая структура модели трансформации

```
Transformation Model наименование модели трансформации {
  Source Meta-Model наименование исходной метамодели {
    Elements [
      элемент 1,
      элемент 2 attributes (свойства 1, свойство 2, …),
      …
    ]
    Relationships [
      элемент 1 is associated with элемент 2,
      элемент 3(свойство) is associated with элемент 4
        (свойство),
      …
    ]
  }
  Target Meta-Model наименование целевой метамодели {
    Elements (
      элемент 1,
      элемент 2 attributes (свойства 1, свойство 2, …),
      …
    ]
    Relationships [
      элемент 1 is associated with элемент 2,
      элемент 3(свойство) is associated with элемент 4
        (свойство),
      …
    ]
  }
  Transformation исходная метамодель to целевая метамодель {
    Rule элемент 1 исходной метамодели to элемент 1 целевой
      метамодели priority номер 1 [
        элемент 1 целевой метамодели(свойство 1) is элемент 1
```

```
32            исходной метамодели(свойство 1)
33          элемент 1 целевой метамодели(свойство 2) is элемент 1
34            исходной метамодели or элемента 2 исходной метамодели
35      ]
36      Rule (элемент 3 исходной метамодели, элемент 4 исходной
37        метамодели) to элемент 2 целевой метамодели priority
38        номер 2 [
39        …
40      ]
41      Rule (элемент 5 исходной метамодели, элемент 6 исходной
42        метамодели) to элемент 3 целевой метамодели priority
43        номер 3 [
44        элемент 3 целевой метамодели(свойство 1) is "значение 1"[
45          if (элемент 5 исходной метамодели(свойство 1) is
46            "значение 1") and (элемент 5 исходной метамодели
47            (свойство 2) is "значение 2")
48        ] or "значение 2" [
49          …
50        ] …
51      ]
52      …
53    }
54  }
```

Приведем пример фрагмента модели трансформации M_T, представленной на TMRL и описывающей преобразование UML-модели (диаграммы классов) в модель онтологии M_{ONT} [14, 233]. Модель трансформации состоит из трех блоков:

1. Описание исходной метамодели (элементы и отношения). Пример фрагмента исходной метамодели для диаграмм классов UML приведен в Листинге 3.

Лисниг 3. Пример фрагмента описания исходной метамодели для диаграмм классов UML

```
1 Source Meta-Model UML-diagram-class {
```

```
  Elements [
    Model,
    Class attributes (xmi.id, name),
    …
  ]
  Relationships [
    Model is associated with Namespace.ownedElement,
    Namespace.ownedElement is associated with Class,
    DataType(xmi.id) is Attribute(type),
    …
  ]
}
```

Блок исходной метамодели «**Source Meta-Model**» содержит описание диаграммы классов UML: «UML-diagram-class», включая элементы модели (раздел «**Elements**»). В данном примере – это элементы «Model» и «Class», при этом элемент «Class» обладает свойствами «xmi.id» и «name».

Помимо описания элементов, исходная метамодель содержит описание связей между элементами метамодели (раздел «**Relationships**»), в том числе по идентификаторам R_{ID}^{cm}, например связь атрибута с типом данных («DataType(xmi.id) **is** Attribute(type)»).

Описание данного блока на TMRL генерируется автоматически в результате анализа исходной метамодели (этап 3.4 алгоритма создания модели трансформации).

2. Описание целевой метамодели (элементы и отношения). Пример фрагмента целевой метамодели для модели онтологии M_{ONT} приведен в Листинге 4.

Листинг 4. Пример фрагмента описания целевой метамодели для модели онтологии

```
Target Meta-Model Ontology {
  Elements [
    ExtendedOntology attributes (id, name),
    Class attributes (id, name),
```

```
    …
  ]
  Relationships [
    Ontology is associated with Class,
    …
  ]
}
```

Блок целевой метамодели «**Target Meta-Model**» содержит описание модели онтологии: «Ontology». Структура блока аналогична структуре блока исходной метамодели, в частности, раздел «**Elements**» содержит описание основных элементов, «**Relationships**» – связей между элементами.

Описание данного блока на TMRL генерируется автоматически в результате анализа целевой метамодели (этап 3.4 алгоритма создания модели трансформации).

3. Описание правил преобразования моделей. Пример фрагмента описания правил преобразования диаграмм классов UML в модель онтологии приведен в Листинге 5.

Листинг 5. Пример правил преобразования диаграмм классов UML в модель онтологии

```
Transformation UML-diagram-class to Ontology {
  Rule Model to Ontology priority 1 [
    Ontology(name) is Model or ModelElement.name
    Ontology(id) is Model(xmi.id)
  ]
  Rule (Class, ModelElement.name) to Class priority 2 [
    Class(name) is Class(name) or ModelElement.name
    Class(id) is Class(xmi.id)
  ]
  …
}
```

Блок описания трансформаций «**Transformation**» содержит описание правил преобразования (отображения) между элементами исходной («UML-

diagram-class») и целевой («Ontology») метамодели. Каждое правило помимо описания возможных соответствий элементов содержит также указание на приоритет выполнения правила («**priority**»).

В приведенном примере рассмотрены правила отображения «Модель-Онтология» и «КлассМодели-КлассОнтологии».

Описание данных правил на TMRL генерируется автоматически (этап 3.4 алгоритма создания модели трансформации).

В создаваемых правилах могут использоваться логические операторы («**and**», «**or**») и оператор условного выбора («**if**»), в частности, с целью задания определенного значения (константы) для целевого элемента метамодели в соответствии с выполнением условия, определенного в блоке «**if**» (например, исходя из определенного значения исходного элемента). Данные правила задаются на этапе 2.5 алгоритма создания модели трансформации.

Дополнительно к описанию модели трансформации M_T, на TMRL возможно задать взаимодействие с другими ранее разработанными программными компонентами трансформации при помощи специальной конструкции «**Call**», описанной в Листинге 6.

Листинг 6. Дополнительный блок описания взаимодействия с другими компонентами трансформации

```
1  Call <название программного компонента трансформации> ,
2    "<путь к концептуальным моделям>" ,
3    "<путь сохранения баз знаний>"
```

При этом если путь сохранения БЗ не задан, то предполагается, что вызываемый программный компонент трансформации осуществляет синтез модели продукций M_{PR} или онтологии M_{ONT}.

Таким образом, программный компонент трансформации, в котором содержится модель трансформации с заданным оператором вызова, является не только конвертором, но и средством для связи с другими компонентами трансформации для поддержки импорта различных форматов концептуальных моделей.

Всего TMRL содержит 15 специальных элементов (лексем), из них: для описания блока исходной и целевой метамодели используется – 7; для блока трансформации – 7; для блока вызова программного компонента трансформации – 1. В качестве начального нетерминала (стартового символа) используется – «*Модель трансформации на TMRL*».

TMRL специализирован на поддержку представления и хранения модели трансформации M_T, вследствие чего он не имеет прямых аналогов, с которыми можно было бы осуществить полное и корректное сравнение. Однако наиболее близкими к TMRL являются языки модельных трансформаций (Model Transformation Language, MTL), например, ATL [131] или языки стандарта QVT (QVT-R, QVT-C, QVT-O) [130] и др.

Приведем краткое наглядное синтаксическое сравнение правила преобразования элемента «Class» из диаграммы классов UML в онтологию OWL на TMRL и ATL. В Листинге 7 приводиться пример правила трансформации на TMRL. В Листинге 8 приводиться пример правила трансформации на ATL (пример взят из работы [143]).

Листинг 7. Правило трансформации на TMRL

```
1 Rule Class to Class priority 1 [
2   Class(ID) is Class(name)
3 ]
```

Листинг 8. Правило трансформации на ATL

```
 1 rule UMLClass2OWLClass {
 2   from
 3     c : UML!uml::Class (
 4       c.oclIsTypeOf(UML!uml::Class) and
 5         not thisModule.sequenceOfUnionClass.includes(c)
 6     )
 7   to
 8     oc : OWL!OWLClass (uriRef <- u),
 9     u : OWL!URIReference (fragmentIdentifier <- l,uri <- uri),
10     l : OWL!LocalName (name <- c.name),
11     uri : OWL!UniformResourceIdentifier (name <- c.name)
```

```
12 }
```

Таким образом, главным отличием TMRL от существующих языков трансформации моделей общего назначения является простота его использования, достигаемая за счет ограниченного набора конструкций. TMRL не является расширением других языков и не использует конструкции других языков, как это очень часто делают MTL, в частности, ATL использует язык ограничений OCL [132]. Кроме того, TMRL обладает человекочитаемым синтаксисом с целью внесения необходимых дополнений (уточнений) в модель трансформации вручную при необходимости. Дополнительной возможностью TMRL является способность описывать взаимодействие с ранее разработанными программными компонентами трансформации по части поддержки импорта различных форматов концептуальных моделей.

2.4. Методика автоматизированной разработки баз знаний с использованием программных компонентов

Методика автоматизированной разработки БЗ представляет собой систематизированную совокупность действий, которые нацелены на решение задачи автоматического создания кода БЗ на целевом ЯПЗ путем трансформации исходных концептуальных моделей с использованием программных компонентов трансформации и возможностью промежуточного хранения и редактирования (уточнения) полученных знаний [15, 21, 22, 24-26, 28, 30, 32, 33, 35, 36].

Основные принципы созданной методики автоматизированной разработки БЗ:

1. возможность использования двухступенчатой трансформации, что повышает технологичность процесса разработки БЗ: сначала преобразование концептуальной модели в модель продукций или онтологии, затем преобразование этой модели в код БЗ на целевом ЯПЗ CLIPS или OWL соответственно;

2. использование модели продукций, представленной в нотации RVML и обеспечивающей унифицированное описание правил, для редактирования (уточнения и дополнения) полученных знаний.

Ограничения на входные и выходные модели:

1. исходные концептуальные модели должны быть представлены в формате XML;
2. в качестве целевого ЯПЗ используется CLIPS и OWL.

Исходя из (6), методика учитывает три варианта преобразования:

1. трансформация исходной концептуальной модели в модель онтологии или продукций;
2. трансформация модели онтологии или продукций в код БЗ на целевом ЯПЗ CLIPS или OWL;
3. трансформация исходной концептуальной модели непосредственно в код БЗ на целевом ЯПЗ CLIPS или OWL без использования унифицированного представления знаний в виде модели онтологии или продукций.

В общем виде автоматизированная разработка БЗ путем трансформации концептуальных моделей может быть представлена в виде последовательности действий на Рисунке 14.

На этапах 1 и 2 средствами внешних программ пользователь строит концептуальную модель предметной области, которая представляется в формате XML. Этот формат является универсальным и наиболее распространенным способом интеграции программных систем и обеспечения обмена информацией между ними.

На этапе 3 процесса автоматизированного анализа XML-структуры концептуальной модели выделяются понятия предметной области и их отношения. Анализ происходит на основе разработанной модели трансформации M_T в форме правил преобразования.

Далее (этап 4) на основе извлеченных понятий и их связей формируется модель онтологии или продукций, как универсальное представление знаний, независящее от исходной концептуальной модели или ЯПЗ.

Рисунок 14. Методика автоматизированной разработки БЗ на основе трансформации концептуальных моделей

На этапе 5 при помощи специальной графической нотации RVML предоставляется возможность визуального отображения, модификации (проверки) полученных знаний в виде продукций. Для визуального отображения и редактирования полученной модели онтологии в виде графа используется специальный графический редактор онтологий.

На этапе 6 происходит автоматическая генерация кода БЗ в формате CLIPS или OWL на основе модели онтологии или продукций соответственно.

Следует отметить, что этап 4 и 5 может отсутствовать, т.к. преобразование исходной концептуальной модели может происходить напрямую в код БЗ без

использования модели онтологии или продукций (первый тип оператора преобразования в (6)).

2.5. Проверка корректности трансформации моделей

В настоящее время разработка любого набора правил трансформации моделей включает их тестирование (верификацию) с целью выявления различного рода ошибок. При этом ошибки могут быть допущены как при создании моделей трансформаций и их метамоделей, так и при их интерпретации.

При создании модели трансформации могут произойти следующие основные ошибки:

1. Создание синтаксически неправильных метамоделей: ошибки или опечатки при создании (кодировании) элементов метамодели.
2. Создание семантически неправильных метамоделей: ошибки, связанные с неверным заданием структуры метамодели (например, неверные связи между элементами, отсутствие каких-либо связей у элементов и т.д.).
3. Неполное покрытие метамоделей: правило трансформации построено без полного охвата элементов исходной и целевой метамодели, что приводит к проблеме того, что некоторые исходные концептуальные модели не могут быть преобразованы (например, правило трансформации работает только для определенных типов элементов и т.д.).
4. Создание семантически неверных отображений между элементами исходной и целевой метамодели.
5. Ошибки или опечатки из-за неправильного кодирования (уточнения) правил трансформации.

При выполнении (интерпретации) модели трансформации могут произойти следующие основные ошибки:

1. Генерация синтаксически неправильных моделей (БЗ): правила трансформации или их часть реализованы неверно, что приводит к

несоответствию целевой модели ее метамодели или нарушению ограничений модели.

2. Генерация семантически неправильных моделей (БЗ): правила трансформации семантически некорректны, что приводит получению неадекватной синтаксически корректной целевой модели.

Для устранения подобных ошибок используется два подхода: первый основан на предупреждении большинства рассмотренных ошибок, второй – на проверке корректности (тестирования, верификации) полученных БЗ и моделей, как результатов трансформаций.

Первый подход реализован путем автоматизированного контроля спецификаций создаваемых моделей в разработанных редакторах и определения ряда особенностей и ограничений, в частности:

1. При построении метамоделей и правил преобразования используется интерактивное визуальное программирование.
2. Код метамоделей и правила преобразования на языке модельных трансформаций генерируются автоматически.
3. При построении метамоделей автоматически проверяются следующие требования:
    - метамодель не должна содержать несвязанные элементы;
    - метамодель не должна содержать элементы без атрибутов (свойств);
    - пары элементов метамодели не могут связываться одной и той же связью несколько раз;
    - пары элементов не могу связываться кольцевой связью одного типа;
    - элемент метамодели не может связываться сам с собой (не должно быть рекурсивных связей).
4. При построении правил преобразования автоматически проверяются следующие требования:
    - модель трансформации должна содержать хотя бы одно правило преобразования;

- правила преобразования могут содержать только соответствия между элементами исходной и целевой метамодели;
- соответствия между атрибутами (свойствами) элементов исходной и целевой метамоделей устанавливаются только после связывания этих элементов;
- соответствия между атрибутами элементов и самих элементов не допускаются;
- модель трансформации допускает содержание не связанных соответствием элементов метамоделей, если они принадлежат к крайним случаям соответствий – избыточность и дефицит выразительной способности;
- у каждого правила преобразования должен быть приоритет, задающий порядковый номер выполнения данного правила в интерпретаторе;
- приоритеты правил преобразования задаются только натуральными числами;
- правила преобразования не должны иметь одинаковые приоритеты;
- приоритеты должны задавать правильную последовательность выполнения правил преобразования исходя из структурных особенностей исходной и целевой метамодели.

Второй подход основан на проверке корректности (тестирования, верификации) полученных БЗ. Проверка корректности БЗ ЭС предполагает обнаружение логических ошибок в представлении знаний и структурах вывода. В ряде случаев, рассматривая верификацию ЭС по аналогии с верификацией традиционных программ, ее трактуют как доказательство правильности БЗ. В настоящее время кроме нахождения обычных ошибок и опечаток в коде БЗ, можно отметить обнаружение различного рода аномалий, которые выделяются в две основные группы [240]:

1. Нарушения согласованности (consistency) БЗ ЭС (например, противоречивость, наличие циклов, избыточность, наличие пересечений и

т.д.). Примерами статических аномалий нарушения согласованности в продукционных БЗ ЭС являются: противоречивые, циклические, избыточные и пересекающиеся правила.

2. Нарушения целостности (completeness) БЗ ЭС (например, неполнота, отсутствие ссылок, некорректность и т.д.). Примерами статических аномалий нарушения целостности в продукционных БЗ ЭС являются: отсутствие правил, приводящих к значениям целевых атрибутов; неиспользованные входные значения; наличие атрибутов, на которые нет ссылок; неверные значения атрибутов.

Данный подход реализован путем использования внешних программных средств, содержащий интерпретаторы и валидаторы полученных целевых моделей и кодов БЗ, в частности:

1. Все разработанные в рамках данной диссертационной работы метамодели (см. Рисунки 3-10) создавались и проверялись с использованием средства GMF на платформе EMF и соответствуют мета-метамодели Ecore.
2. Проверка корректности сгенерированных продукционных БЗ в формате CLIPS производилась в системе программирования БЗ – Personal Knowledge Base Designer (PKBD) [241-243] с использованием подключаемых динамических библиотек машин вывода, в частности, реализующих алгоритм RETE.
3. Проверка отдельных правил БЗ на наличие нарушений целостности может быть произведена в системе CHECK, при этом анализируется структура правила (выявление невыполнимых условий и недостижимых заключений) и весь набор правил сравнивается с множеством возможных значений атрибутов (поиск неиспользуемых значений атрибутов и недостижимых целей).
4. Проверка на непротиворечивость БЗ может быть осуществлена в системах КЛАСС и ДИФКЛАСС.
5. Проверка корректности сгенерированных онтологических БЗ в формате OWL производилась в системе онтологического моделирования Protégé с

использованием машин вывода (reasoners), в частности Pellet, FaCT++, HermiT.

2.6. Выводы

Предложены модели и методы автоматизации процесса создания программных компонентов интеллектуальных систем, обеспечивающих проектирование и синтез кодов БЗ путем трансформации концептуальных моделей предметных областей, включая:

- Оригинальную модель типового программного компонента, особенностью которой является использование модели трансформации в форме сценария преобразования (декларативной программы) и интерпретатора данной модели, обеспечивающих специализацию типового программного компонента в соответствии с определенными форматами входных и выходных данных.
- Метод создания программных компонентов анализа концептулаьных моделей и синтеза кода БЗ, особенностью которого является использование языка модельных трансформаций и оригинальной модели типового программного компонента.
- Предметно-ориентированный декларативный язык TMRL, предназначенный для представления и хранения модели трансформации, отличающийся от известных языков трансформации общего назначения: узкой специализацией, направленной на обеспечение трансформации концептуальных моделей в БЗ и простотой его использования, за счет небольшого количества конструкций (TMRL не использует сложный синтаксис; не является расширением других языков; не использует конструкции других языков); возможностью описания механизма взаимодействия с ранее разработанными программными компонентами трансформации.

- Обобщенную модель онтологии и модель продукций для унифицированного представления знаний, не зависящие от какого-либо источника концептуальных моделей и ЯПЗ, предназначенные для промежуточного хранения полученных знаний из концептуальных моделей.
- Методику автоматизированной разработки БЗ, отличием которой от известных является использование концептуальных моделей в качестве исходных данных, специализированных программных компонентов в качестве средств, и специализированного визуального языка RVML для проектирования и синтеза программных кодов БЗ.

Глава 3. Веб-ориентированное инструментальное программное средство

На основе созданных моделей и метода разработано инструментальное средство в форме веб-ориентированной расширяемой системы программирования – Knowledge Base Development System (KBDS) [23, 27, 29, 31, 34, 37, 38, 40, 41, 244].

Разработанная система позволяет создавать программные компоненты интеллектуальных систем, обеспечивающие автоматизированный анализ концептуальных моделей и генерацию кода БЗ на целевом ЯПЗ CLIPS или OWL, а также предоставляет повсеместный и удобный сетевой доступ по требованию к общему пулу проектов БЗ, тем самым обеспечивая совместную удаленную работу пользователей при проектировании и разработки БЗ.

Веб-ориентированная программная система реализована с использованием языка – PHP [230] и Yii2 Framework [245] – высокоэффективный, основанный на компонентной структуре PHP-фреймворк для быстрой разработки крупных веб-приложений, позволяющий применить концепцию повторного использования кода и существенно ускорить процесс веб-разработки [246].

Веб-ориентированная программная система разрабатывалась с использованием шаблона проектирования – «Модель-Представление-Контроллер» (Model-View-Controller, MVC). MVC предназначен для разделения бизнес-логики и пользовательского интерфейса, чтобы разработчики могли легко изменять отдельные части приложения, не затрагивая другие. В архитектуре MVC модель предоставляет данные и правила бизнес-логики, представление отвечает за пользовательский интерфейс (например, текст, поля ввода), а контроллер обеспечивает взаимодействие между моделью и представлением [247].

При разработке KBDS также использовались библиотеки jQuery [248] и jsPlumb [249] для построения графических редакторов. В качестве СУБД

использован MySQL [250], однако, применяя механизм миграций, существует возможность использовать PostgreSQL [251].

3.1. Назначение и основные принципы

Назначением разработанной веб-ориентированной программной системы является поддержка процесса создания БЗ путем разработки программных компонентов, обеспечивающих автоматизированный анализ исходных концептуальных моделей предметных областей и генерацию кода БЗ, представляющего собой декларативные программы на ЯПЗ. Данная система ориентирована на непрограммирующих пользователей: экспертов, инженеров по знаниям, системных аналитиков и др.

Основные принципы разработанной веб-ориентированной программной системы:

1. Обеспечение удаленного доступа через Интернет к функциям веб-ориентированной программной системы без необходимости размещения дополнительного программного обеспечения на компьютерах пользователей.
2. Организация единого пространства (среды) для функционирования различных программных компонентов синтеза БЗ.
3. Предоставление авторизованного доступа к функциональным возможностям веб-ориентированной программной системы при помощи единой подсистемы администрирования, которая, прежде всего, подразумевает предотвращение несанкционированного доступа к модулю разработки программных компонентов.
4. Поддержка совместной (коллективной) и удаленной деятельности пользователей в процессе автоматизированной разработки БЗ.

5. Обеспечение накопления и развития программных компонентов интеллектуальных систем создания БЗ на основе трансформации различных концептуальных (информационных) моделей.
6. Поддержка мультиязычности (интернационализация) системы (поддержка английского и русского языка).
7. Обеспечение кроссбраузерности (интерфейс системы идентично отображается и работает во всех браузерах).
8. Поддержка адаптивности дизайна веб-страниц системы (правильное отображение страниц системы на различных устройствах, подключённых к Интернету, а также динамически подстраивающиеся страницы системы под заданные размеры окна браузера).

3.2. Пользователи системы

Веб-ориентированная программная система предусматривает работу трех типов пользователей:

1. Гость – незарегистрированный пользователь, который может просматривать справочную информацию о системе и ее существующих модулях (системных и прикладных), а также может ознакомиться со всеми открытыми проектами БЗ. Данному типу пользователя доступна функция регистрации.
2. Зарегистрированный пользователь (разработчик и владелец программного компонента) – пользователь успешно прошедший регистрацию. Данному типу пользователя доступны все функции системной группы модулей веб-ориентированной программной системы, а именно: информационная поддержка, разработка прикладных программных компонентов, визуализация (редакторы онтологии и продукций), а также все разработанные программные компоненты создания БЗ (если к ним

установлен открытый доступ администратором соответствующего программного компонента).

3. Администратор (разработчик и владелец веб-ориентированной программной системы) – обладает полными правами доступа ко всем модулям и программным компонентам веб-ориентированной программной системы и обеспечивает управление всей системой в целом.

3.3. Архитектура веб-ориентированной программной системы

С целью программной реализации предлагаемых моделей и метода разработана концептуальная архитектура (схема) веб-ориентированной программной системы, описывающая основные функциональные элементы и их взаимодействие [14, 16-18].

Концептуальная схема архитектуры веб-ориентированной программной системы, изображенная на Рисунке 15, позволяет описать ее структуру, включая состав и типы элементов, а также принципиальные особенности функционирования.

Веб-ориентированная программная система состоит из трех основных групп модулей:

1. Информационные – отражают все информационные ресурсы программной системы, предназначенные для хранения служебной информации, которая используется подсистемами (в системной и прикладной части), как для обеспечения собственного функционирования, так и для решения задач автоматизированной разработки БЗ;
2. Системные – представляют собой набор всех предлагаемых пользователям подсистем, обеспечивающих как базовое взаимодействие пользователей с веб-ориентированной программной системой, так и предоставляющие интерфейс взаимодействия с внешними программными системами, а также

предоставляющие инструментарий для создания прикладных программных компонентов на основе типового;

3. Прикладные – представляют собой набор всех разработанных пользователями программных компонентов, обеспечивающих возможность автоматического синтеза БЗ путем трансформации концептуальных моделей. Программные компоненты создаются на основе типового программного компонента путем его «клонирования» (копирования) и настройки (специализации).

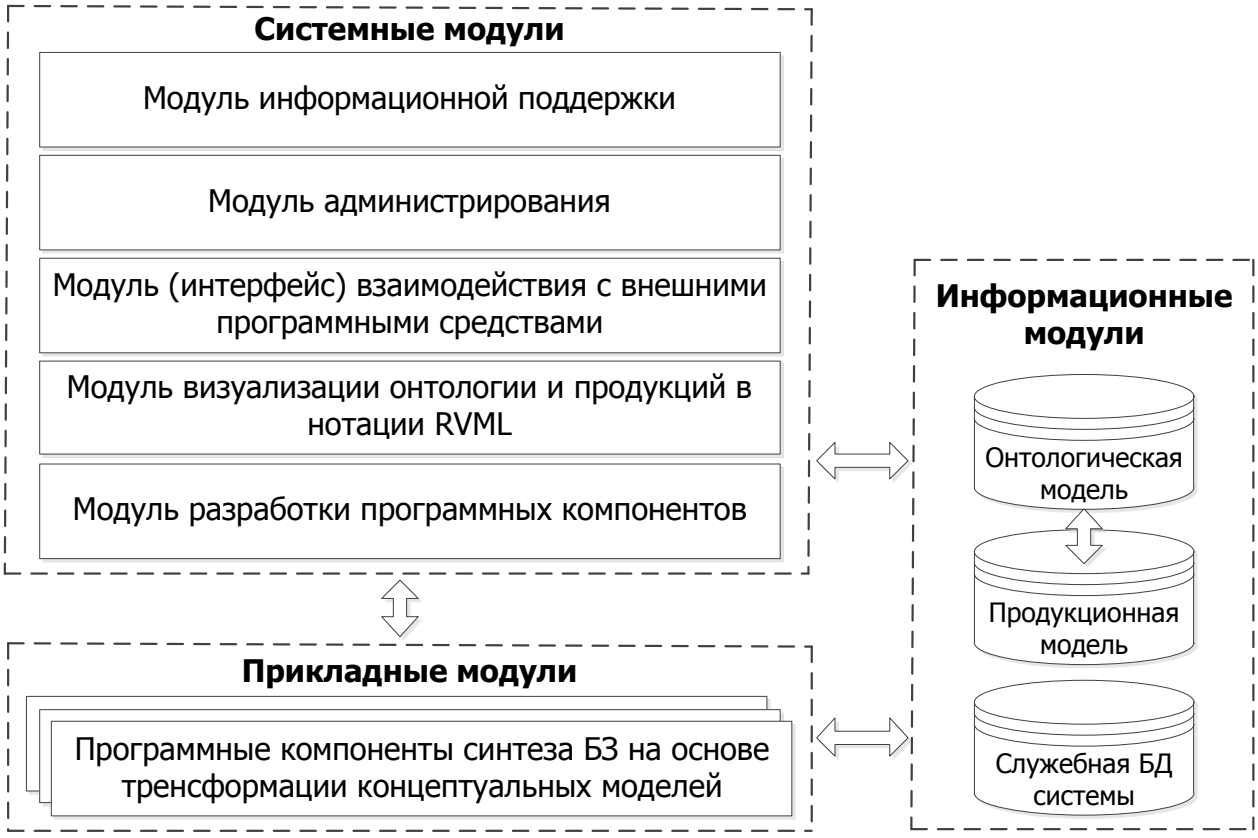

Рисунок 15. Концептуальная архитектура (схема) веб-ориентированной программной системы

Уточним множество функций, предоставляемых системной группой модулей веб-ориентированной программной системы:

1. Разработка программного компонента на основе модели типового шаблонного программного компонента M_{TPC}, включая визуальное и

текстовое построение правил трансформации между элементами исходной и целевой метамодели с целью создания модели трансформации M_T .

2. Хранение полученных знаний из концептуальных моделей с использованием специальной модели онтологии и продукций.
3. Визуальное отображение и редактирование понятий предметной области и их отношений в виде графа (модель онтологии).
4. Визуальное отображение и редактирование (моделирование) модели продукций с использованием специальной графической нотации RVML [13, 234].
5. Обеспечение взаимодействия внешних программных систем с информационными ресурсами веб-ориентированной программной системы по части создания онтологической и продукционной модели на основе концептуальной модели, генерации кода БЗ на основе онтологической и продукционной модели, а также предоставления специальных функций для манипулирования элементами онтологической и продукционной моделью: создание, редактирование и удаление.

Уточним множество функций, предоставляемых прикладной группой модулей веб-ориентированной программной системы:

1. Формирование модели онтологии и продукций на основе автоматизированного анализа концептуальных моделей предметных областей.
2. Генерация кода БЗ на целевом ЯПЗ путем трансформации модели онтологии или продукций.
3. Генерация кода БЗ на целевом ЯПЗ на основе трансформации концептуальных моделей предметных областей.
4. Обеспечение взаимодействия внешних программных систем с соответствующими программными компонентами по части импорта концептуальных моделей и генерации на их основе прототипов БЗ.

С целью реализации описанной выше концептуальной схемы веб-ориентированной программной системы (см. Рисунок 15) разработана клиент-серверная архитектура [40, 41], представленная на Рисунке 16.

Рисунок 16. Архитектура веб-ориентированной программной системы

Клиентская часть системы включает следующие основные модули:

- графический редактор продукций в нотации RVML [38];
- графический редактор онтологий – обеспечивает визуальное отображение и редактирование знаний в виде графа (онтологической модели);
- графический редактор метамоделей;
- графический редактор моделей трансформации – обеспечивает визуальное отображение и редактирование правил соответствия (отображения) исходных элементов метамодели (концептуальной модели) в целевые элементы метамодели (модели БЗ) [37].

Серверная часть системы включает следующие основные модули:

- администрирования – обеспечивает взаимодействие пользователей с системой (ограничение прав доступа, учет всех зарегистрированных пользователей и групп пользователей, сбор и анализ различной статистической информации, мониторинг работоспособности и др.);

- управления БЗ – обеспечивает создание и учет проектов БЗ;
- управления мета-уровнем представления знаний – обеспечивает внутреннее представление знаний в виде продукционной и онтологической модели, которое позволяет абстрагироваться от особенностей описания знаний в различных ЯПЗ, используемых при реализации БЗ (например, CLIPS, JESS, Drools, RuleML, OWL, SWRL и др.), и хранить знания в собственном независимом формате;
- разработки программных компонентов – обеспечивает создание и учет проектов программных компонентов, а также генерацию кода программных компонентов на основе созданной модели трансформации и выбранных блоков анализатора и генератора.
- программные компоненты – обеспечивают генерацию модели БЗ (продукции или онтология) на основе анализа исходных концептуальных моделей и генерацию кода БЗ CLIPS или OWL на основе анализа модели БЗ или исходной концептуальной модели.

3.4. Интерфейс пользователя

Графический интерфейс пользователя веб-ориентированной программной системы представлен на Рисунке 17 и состоит из пяти основных элементов:

1. главное меню – содержит основные разделы системы («учетная запись», «мои проекты» и «администрирование»);
2. навигационная цепочка – представляет собой путь по страницам системы от корня до рабочей страницы, которую в данный момент просматривает пользователь;
3. правое дополнительное (вспомогательное) меню – содержит все возможные действия, которые доступны пользователю в рамках выбранного раздела;
4. рабочая область – основной элемент интерфейса пользователя, содержащий смысловое наполнение выбранной страницы (раздела);

5. блок в нижней части страницы (подвал) – содержит контактную информацию и авторские права.

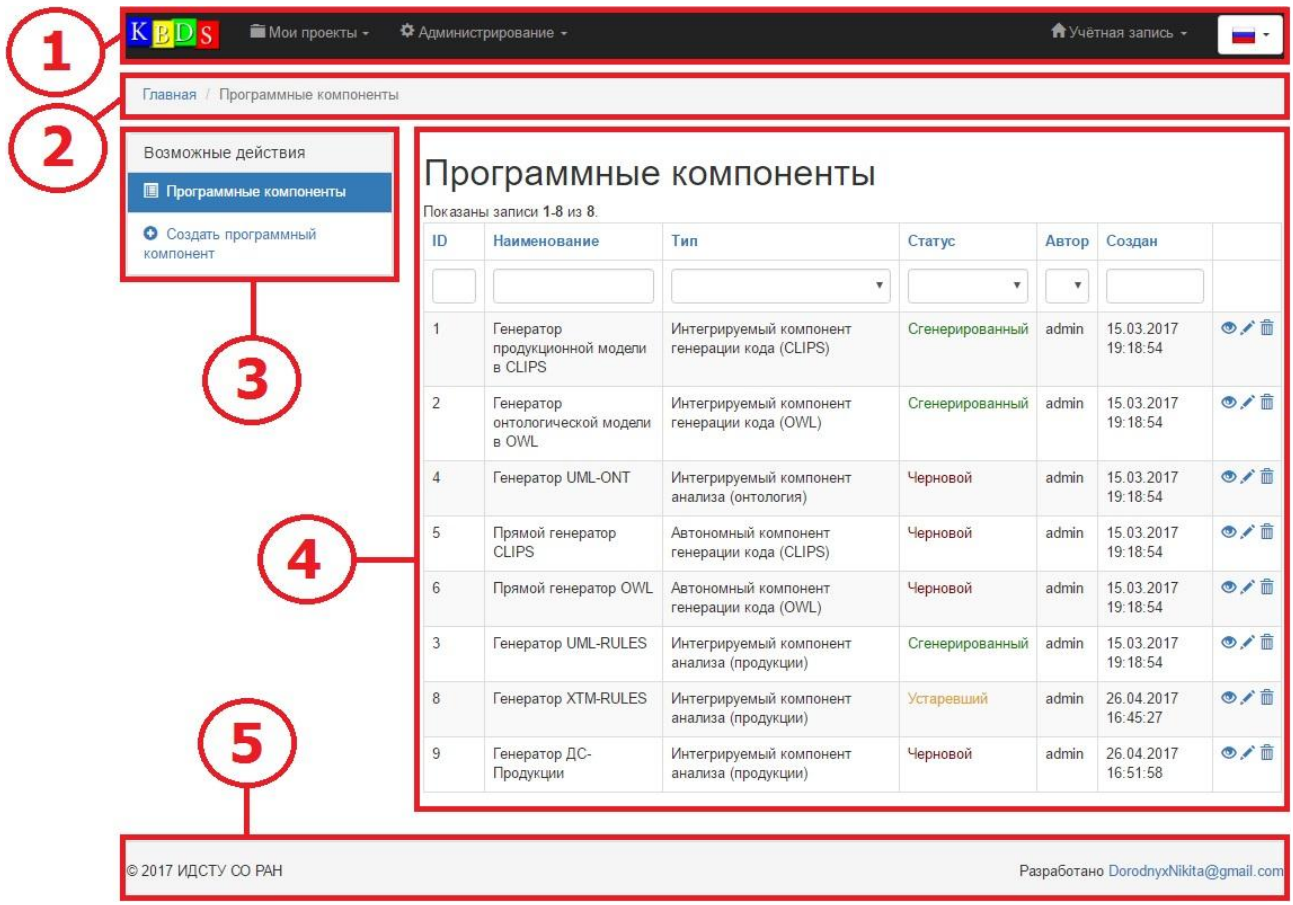

Рисунок 17. Основные элементы интерфейса пользователя

Следует отметить, что у каждого проекта программного компонента есть свой статус, определяющий состояние разработки данного компонента:

- «сгенерированный» программный компонент – это разработанный компонент (код компонента сгенерирован; код модели трансформации находится в актуальной состоянии), доступный для генерации БЗ;
- «черновой» программный компонент – это компонент, находящийся на стадии разработки и не доступный для генерации БЗ;
- «устаревший» программный компонент – это компонент, требующий перегенерации своего кода (код компонента сгенерирован, но модель трансформации была изменена), однако он доступен для генерации БЗ.

На рисунке 18 показана экранная форма (веб-страница) администрирования всех проектов БЗ, созданных в системе.

У каждого проекта БЗ есть свой статус, определяющий уровень доступа к данной БЗ:

- «открытая» БЗ – это БЗ доступная для редактирования и генерации кода всем пользователям системы;
- «закрытая» БЗ – это БЗ доступная для редактирования и генерации кода только владельцу (создателю) данной БЗ.

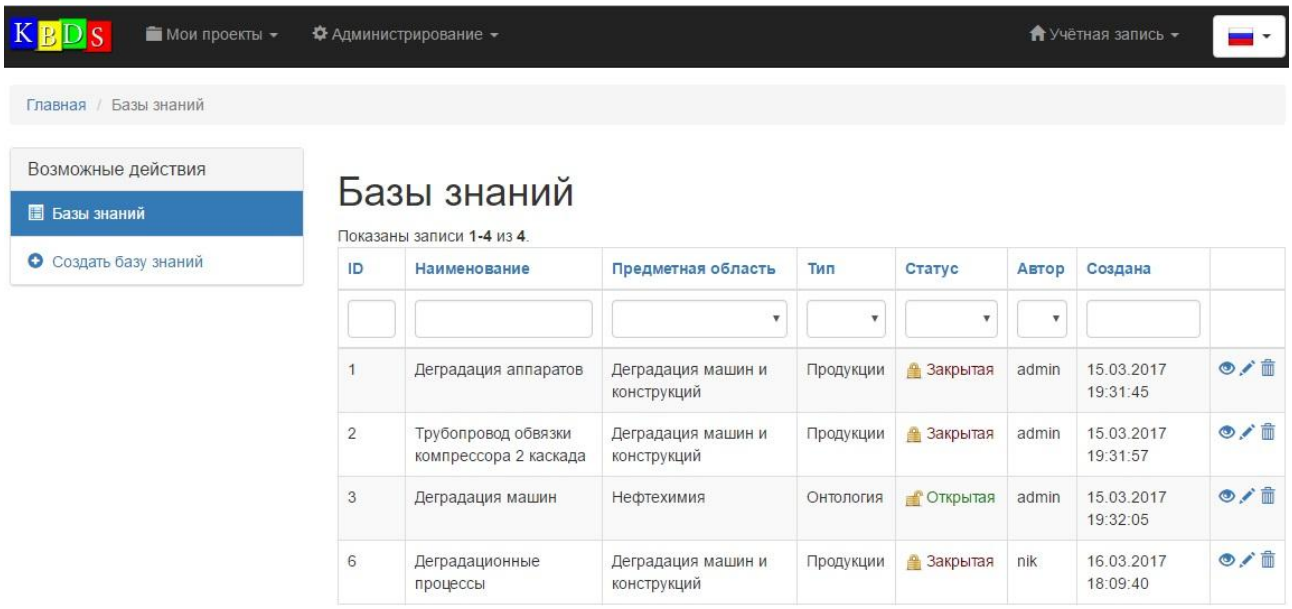

Рисунок 18. Экранная форма администрирования проектов БЗ

На рисунке 19 показана экранная форма (веб-страница) визуального интерактивного редактора модели онтологии.

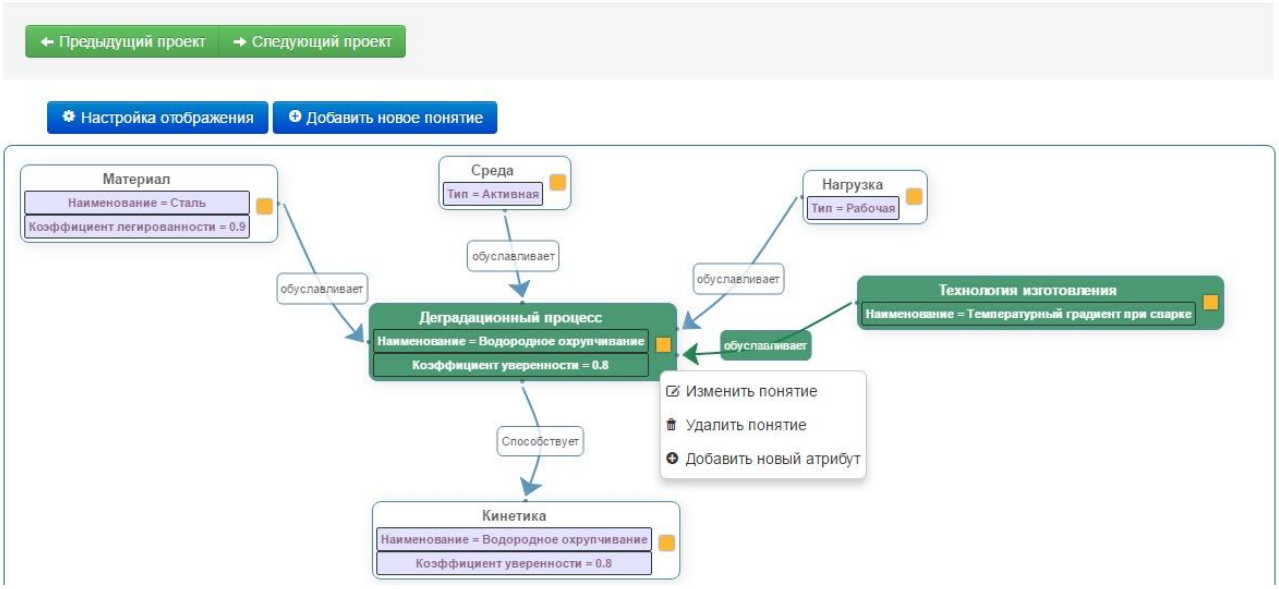

Рисунок 19. Экранная форма графического редактора модели онтологии

На рисунке 20 показана экранная форма (веб-страница) визуального интерактивного редактора модели продукций в нотации RVML.

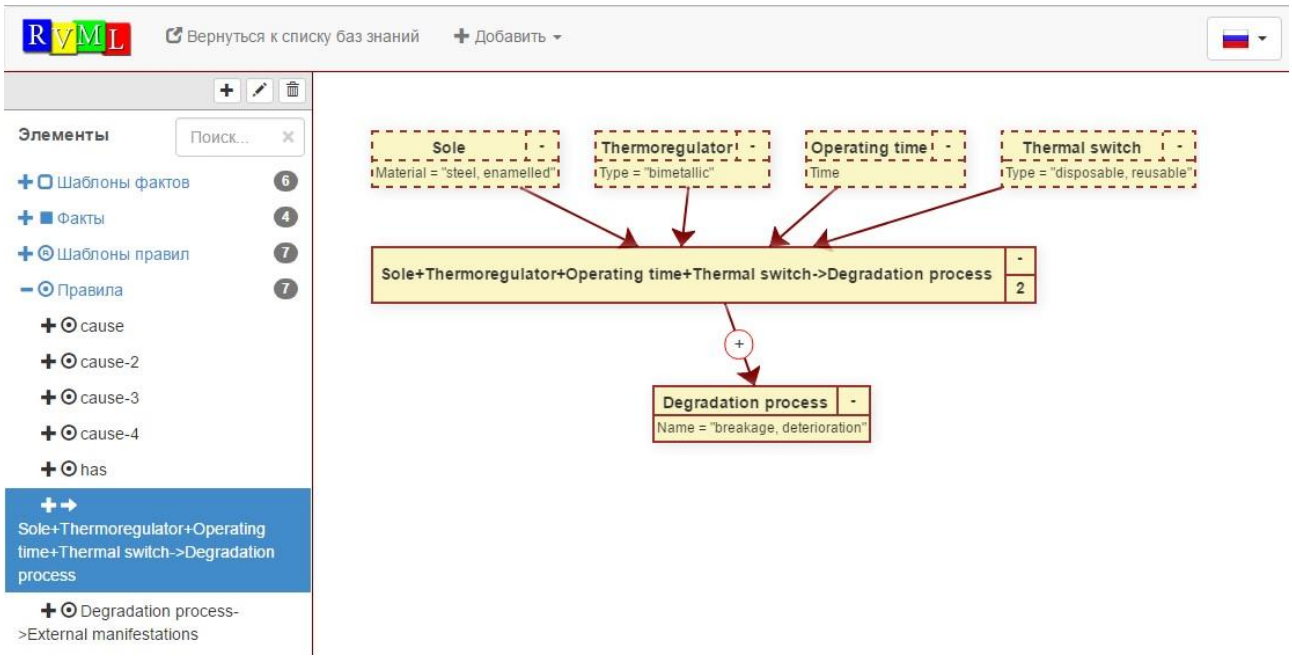

Рисунок 20. Экранная форма графического редактора RVML

3.5. Интерфейс взаимодействия программ

С целью обеспечения взаимодействия веб-ориентированной программной системы с внешними программными средствами, реализован специальный программный интерфейс.

Интерфейс предоставляет внешним программным системам ряд методов для доступа к разработанным программным компонентам в части импорта концептуальных моделей и генерации кода БЗ на ЯПЗ CLIPS или OWL.

Основные методы (функции) данного интерфейса взаимодействия:

- getAllModulesList – получение списка всех программных компонентов, созданных в KBDS;
- getModulesList – получение списка программных компонентов с определенным типом и статусом, созданных в KBDS;
- getAllKnowledgeBasesList – получение списка всех БЗ, созданных в KBDS;

- getKnowledgeBasesList – получение списка БЗ с определенным типом и статусом, созданных в KBDS;
- importConceptualModel – импорт концептуальной модели и создание на ее основе БЗ в системе KBDS
- exportKnowledgeBase – экспорт кода БЗ из системы KBDS.

Полное описание данных методов взаимодействия приводится в Приложении В.

В дополнение к приведенным выше методам веб-ориентированная программная система предоставляет внешним программным системам ряд методов (функций) для взаимодействия с моделью онтологии и продукций (осуществления различного рода манипуляций над этими моделями). Полное описание данных методов взаимодействия также приводится в приложении В.

Предполагается, что разработанный программный интерфейс взаимодействия может быть использован любым внешним программным (интеллектуальным) средством. В рамках диссертационного исследования отработка (апробация и тестирование) взаимодействия производилось на примере системы программирования БЗ – PKBD. Таким образом, в работе предполагается интеграция данных средств по части обмена данными между ними (импорт концептуальных моделей и экспорт синтезируемого кода БЗ, а также работа с общим пулом онтологических и продукционных моделей) с целью повышения эффективности разработки прогностических ЭС.

3.6. Выводы

- Разработано инструментальное программное средство в форме веб-ориентированной расширяемой системы программирования БЗ (KBDS), реализующее предлагаемые метод и язык. Система обладает клиент-серверной архитектурой и состоит из трех основных групп модулей: системных, прикладных и информационных.

- KBDS обеспечивает поддержку автоматизации разработки программных компонентов интеллектуальных систем, предназначенных для генерации кода БЗ на целевом ЯПЗ путем трансформации концептуальных моделей.
- Особенностью KBDS является поддержка повсеместного и сетевого доступа по требованию к общему пулу проектов БЗ, тем самым обеспечивая совместную (коллективную) удаленную работу пользователей при проектировании и синтезе БЗ.
- Разработанная система обеспечивает взаимодействие с внешними программными средствами при помощи специального программного интерфейса для удобного доступа к моделям онтологии и продукций, а также в части импорта концептуальной модели и экспорта сгенерированного кода БЗ на ЯПЗ CLIPS или OWL.

Глава 4. Апробация

Апробация предлагаемых моделей, метода инструментального средства (KBDS) осуществлена на примере разработки программных компонентов анализа концептуальных моделей в форме:

- деревьев событий (ДС) [42], разработанных в графическом редакторе ДС TreeEditorET;
- диаграмм классов UML [23, 27, 29, 34], разработанных в CASE-средстве IBM Rational Rose Enterprise [252] и представленных с использованием стандарта XMI [195];
- концепт-карт (карт знаний/интеллект-карт) [43], разработанных в редакторе IHMC CmapTools [253] и представленных с использованием стандарта XTM [196].

А также на примере разработки продукционной БЗ в формате CLIPS [39] для прогнозирования развития деградационных процессов аппаратов в нефтехимии [254-256] на основе созданного программного компонента анализа ДС.

Апробация включает оценку эффективности разработки БЗ [40, 41] по сравнению с классическим («ручным») методом без использования концептуальных моделей в качестве основы для синтеза кодов БЗ.

4.1. Разработка программного компонента анализа диаграмм классов UML

Для разработки программного компонента анализа концептуальных моделей в форме диаграмм классов UML [141, 194] использовалась разработанное инструментальное программное средство (KBDS) [19, 23, 31, 34, 40, 41, 244].

Процесс разработки соответствует этапам создания программного компонента, представленным на Рисунке 11.

Перед тем как приступить к процессу создания программного компонента, пользователь должен создать соответствующий проект, в рамках которого будет осуществляться данная разработка.

На Рисунке 21 показана экранная форма (веб-страница) создания проекта нового программного компонента. Следует отметить, что пользователю необходимо определить тип компонента, в зависимости от которого в дальнейшем будут доступны соответствующие целевые метамодели (в частности метамодель для модели продукций и CLIPS), а также необходимые для данного типа компонента анализатор и генератор.

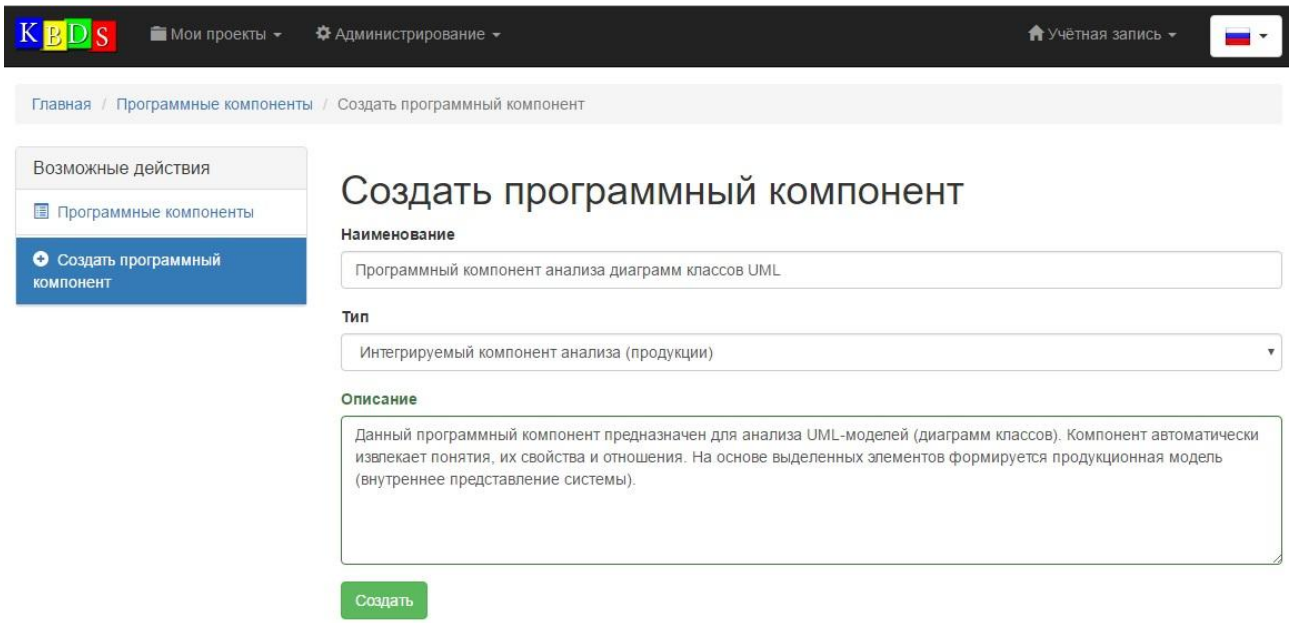

Рисунок 21. Экранная форма создания нового проекта программного компонента

Система проинформирует пользователя об успешном создании нового проекта программного компонента специальным сообщением. Также пользователю будут доступны функции изменения, удаления и просмотра общей информации по данному программному компоненту.

Перед тем как приступить к разработке модели трансформации, пользователю необходимо сформировать метамодель исходной концептуальной модели. Данную метамодель можно получить тремя разными способами:

вручную, при помощи специального графического редактора; с использованием XML-схем (XSD-спецификаций) [176]; путем анализа исходной концептуальной модели (процедура обратной инженерии).

На Рисунке 22 показана экранная форма (веб-страница) создания новой метамодели.

Метамодель для диаграмм классов UML создавалась на основе процедуры обратной инженерии. Для этого необходимо произвести импорт исходной диаграммы классов UML в формате XML. На Рисунке 23 показана экранная форма (веб-страница) импорта исходной диаграммы классов UML. Следует отметить, что пользователю доступны другие возможные операции (действия) с метамоделью (на панели вспомогательного правого меню). Также если метамодель была сгенерирована ранее, то система проинформирует пользователя соответствующим сообщением, и дальнейший импорт исходной модели приведет к потере существующей метамодели.

Анализ XML-структуры исходной концептуальной модели подобен построению XML-схемы на основе шаблона (паттерна) проектирования XSD – Venetian Blind и Garden of Eden. Названия элементов метамодели соответствуют названиям узлов XML-элементов. Свойства элемента метамодели соответствуют атрибутам узла XML-элемента. Связи между элементами метамодели определяются путем вложенности XML-улов друг в друга. Также при анализе учитываются пространства имен, объявленные в исходной модели.

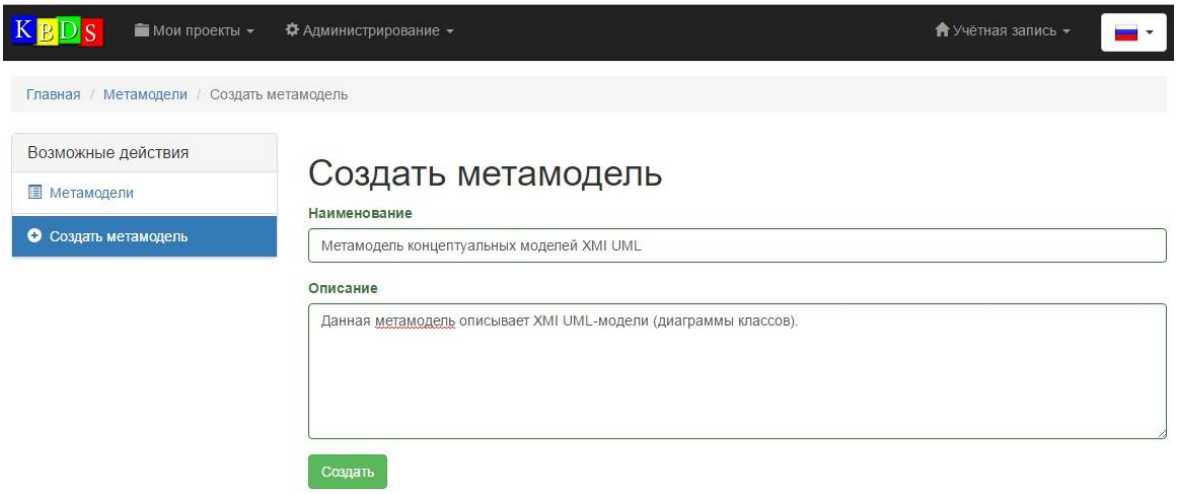

Рисунок 22. Экранная форма создания метамодели

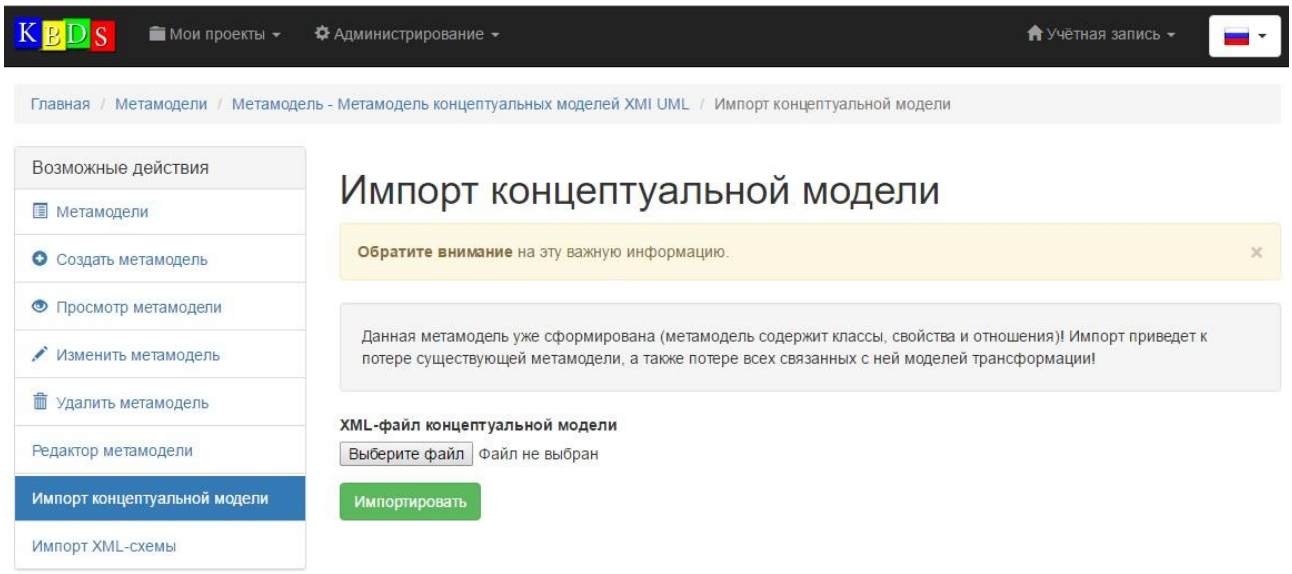

Рисунок 23. Экранная форма импорта исходной концептуальной модели

После завершения процедуры анализа XML-структуры исходной диаграммы классов UML сформированную метамодель можно визуально отобразить и при необходимости дополнить с помощью специального графического редактора метамоделей, представленного на Рисунке 24.

Полученная полная метамодель исходной концептуальной модели в форме диаграмм классов UML приведена на Рисунке 25. Следует отметить, что она содержит некоторые лишние конструкции, которые не будут задействованы при построении правил трансформации. Это связано с издержками (несовершенством) процедуры обратной инженерии.

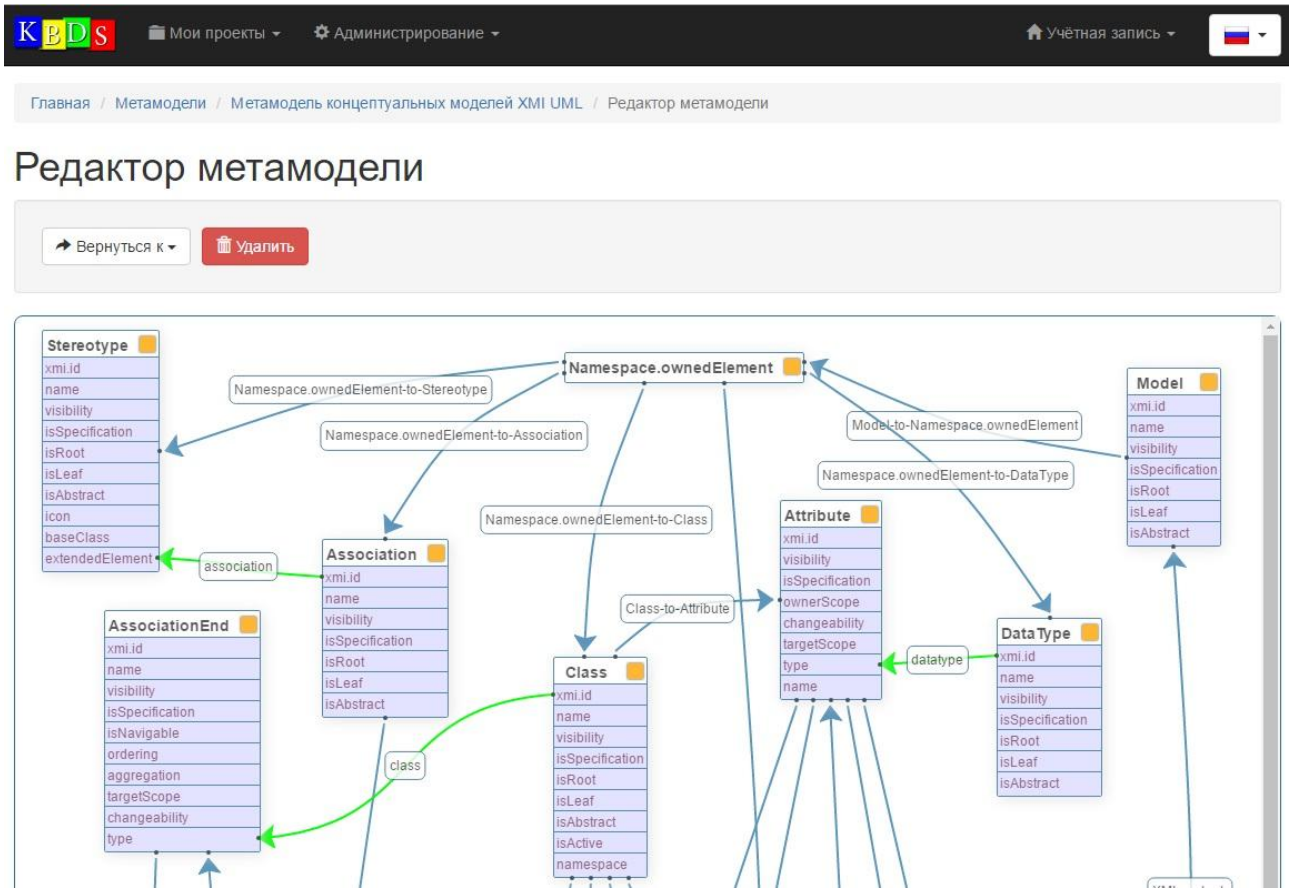

Рисунок 24. Экранная форма графического редактора метамоделей (метамодель диаграмм классов UML)

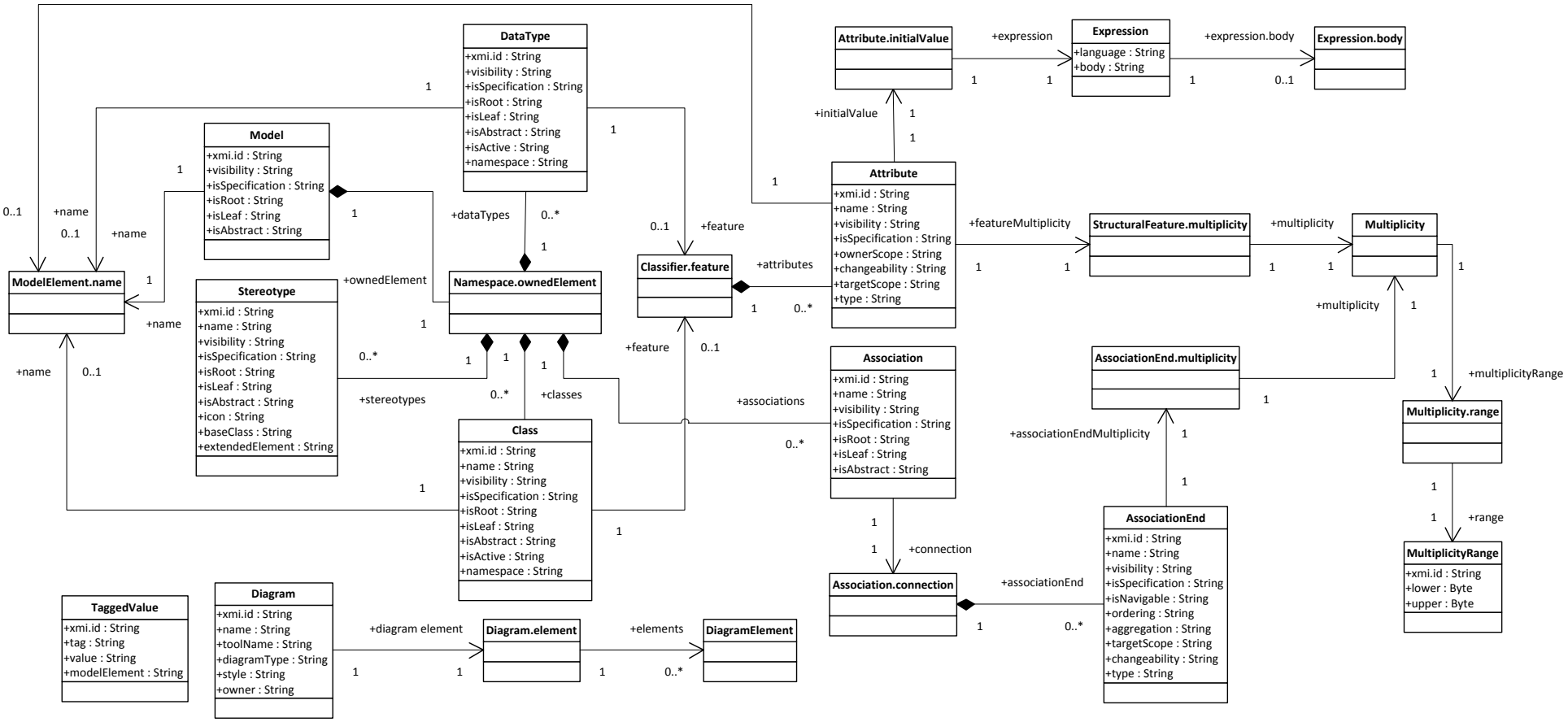

Рисунок 25. Метамодель исходной диаграммы классов UML (XMI)

После создания проекта нового программного компонента и формирования метамодели исходной концептуальной модели следует основной этап разработки программного компонента – построение модели трансформации. Для этого пользователю необходимо создать проект модели трансформации.

На Рисунке 26 показана экранная форма (веб-страница) создания проекта новой модели трансформации.

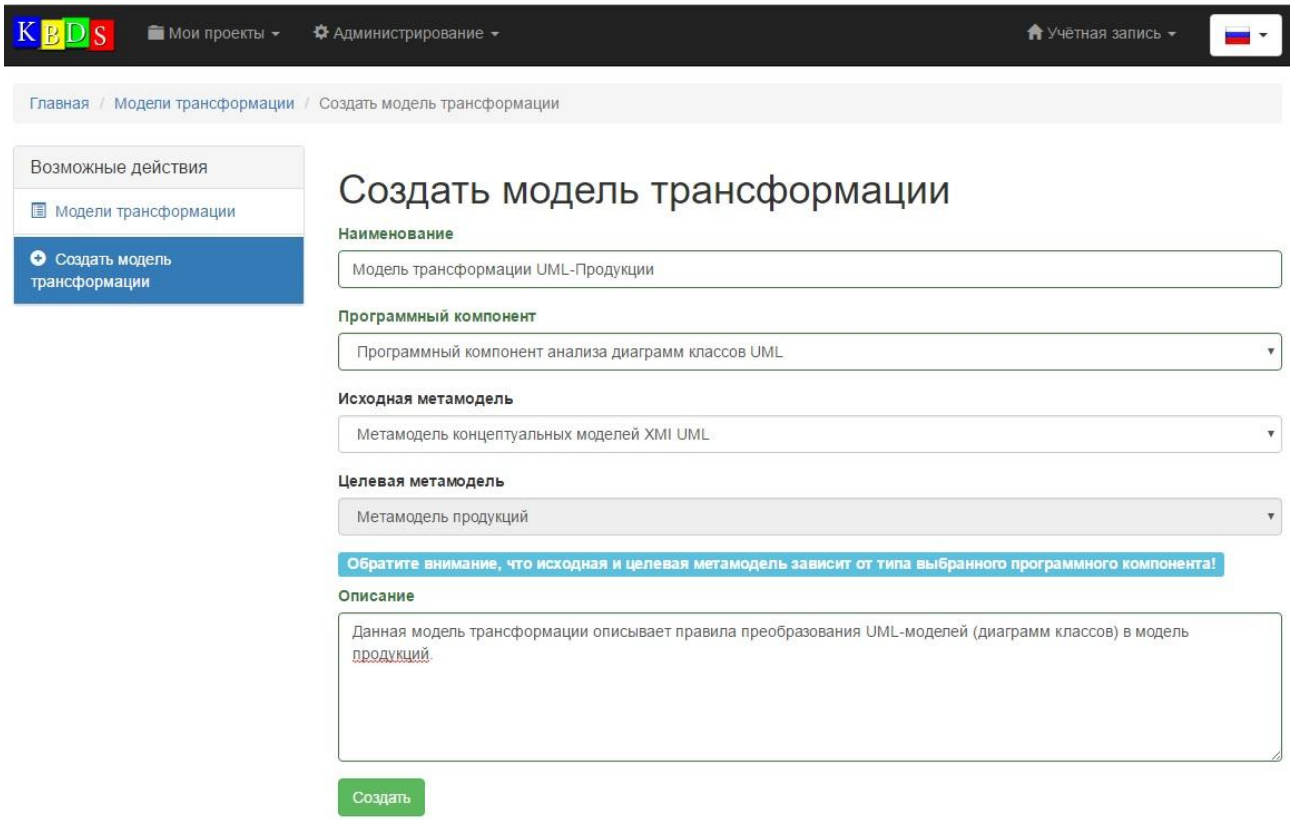

Рисунок 26. Экранная форма создания проекта модели трансформации

Далее при помощи специального графического редактора модели трансформации [37] устанавливаются необходимые соответствия (правила трансформации) между исходными элементами метамодели (диаграммы классов UML) и целевыми элементами метамодели (модели продукций).

На Рисунке 27 показана экранная форма (веб-страница) графического редактора построения правил трансформации (соответствий элементов метамоделей). При этом модель трансформации еще не содержит ни одного правила трансформации.

Редактор модели трансформации состоит из трех основных элементов:

1. Панель с кнопками (номер 1 на Рисунке 27), содержащая, в том числе, кнопку «Генерация программного компонента» для сборки (синтеза кода) программного компонента. После сборки данному программному компоненту присваивается статус «сгенерированный», и он будет доступен для разработки БЗ.
2. Правое меню (номер 2 на Рисунке 27) содержит иерархию (деревья) всех элементов исходной и целевой метамодели. При этом с помощью специальных переключателей (галочек) пользователь может скрыть лишние элементы метамоделей, которые не участвуют в соответствиях. Также пользователь может осуществить поиск необходимого элемента метамодели при помощи поля поиска.
3. Рабочая область – основной элемент редактора, содержащий все исходные элементы метамодели диаграммы классов UML (номер 3 на Рисунке 27) и все целевые элементы метамодели модели продукций (номер 4 на Рисунке 27).

Далее пользователю необходимо установить соответствия между исходными и целевыми элементами метамоделей (между мета-классами и мета-атрибутами). При этом задается приоритет выполнения данного правила трансформации, определяющий последовательность выполнения правила в интерпретаторе.

На Рисунке 28 показана экранная форма (веб-страница) графического редактора модели трансформаций с установленными соответствиями между элементами метамоделей.

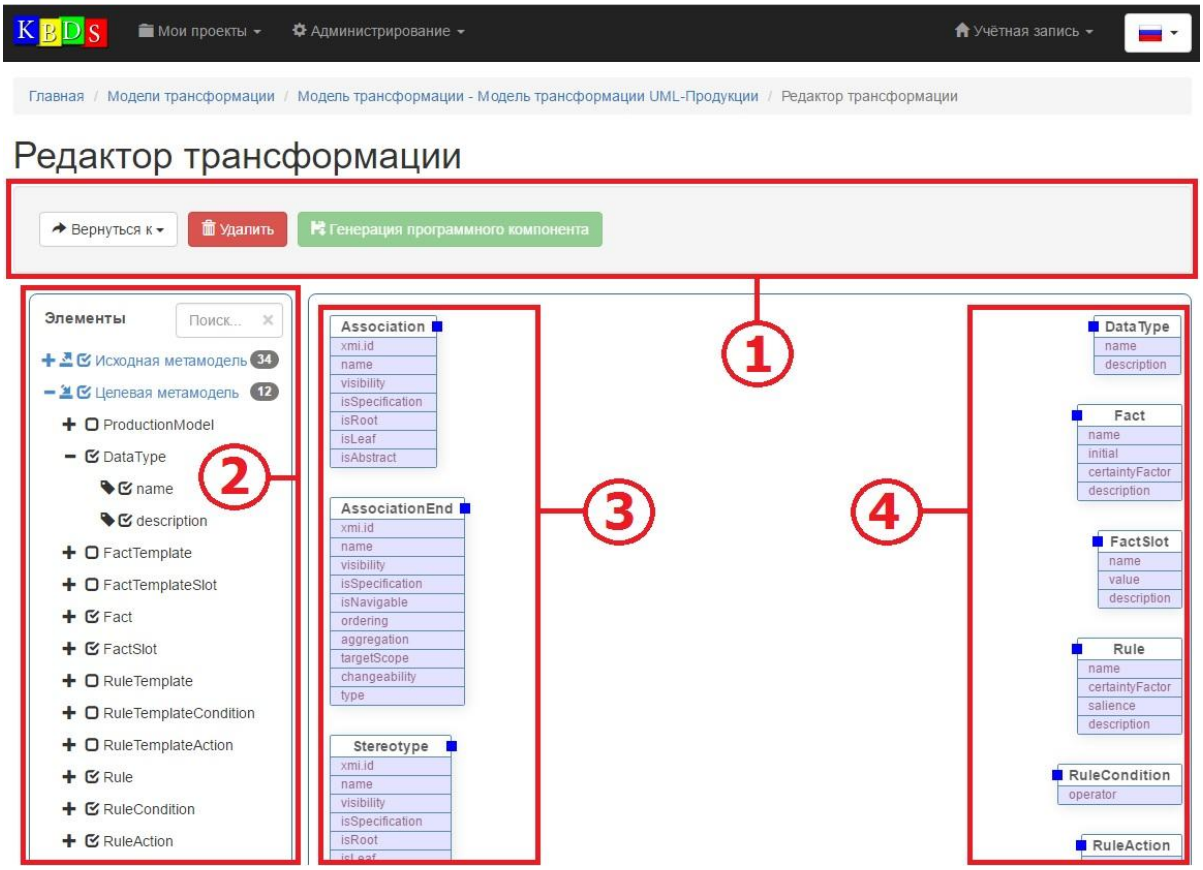

Рисунок 27. Экранная форма графического редактора модели трансформации

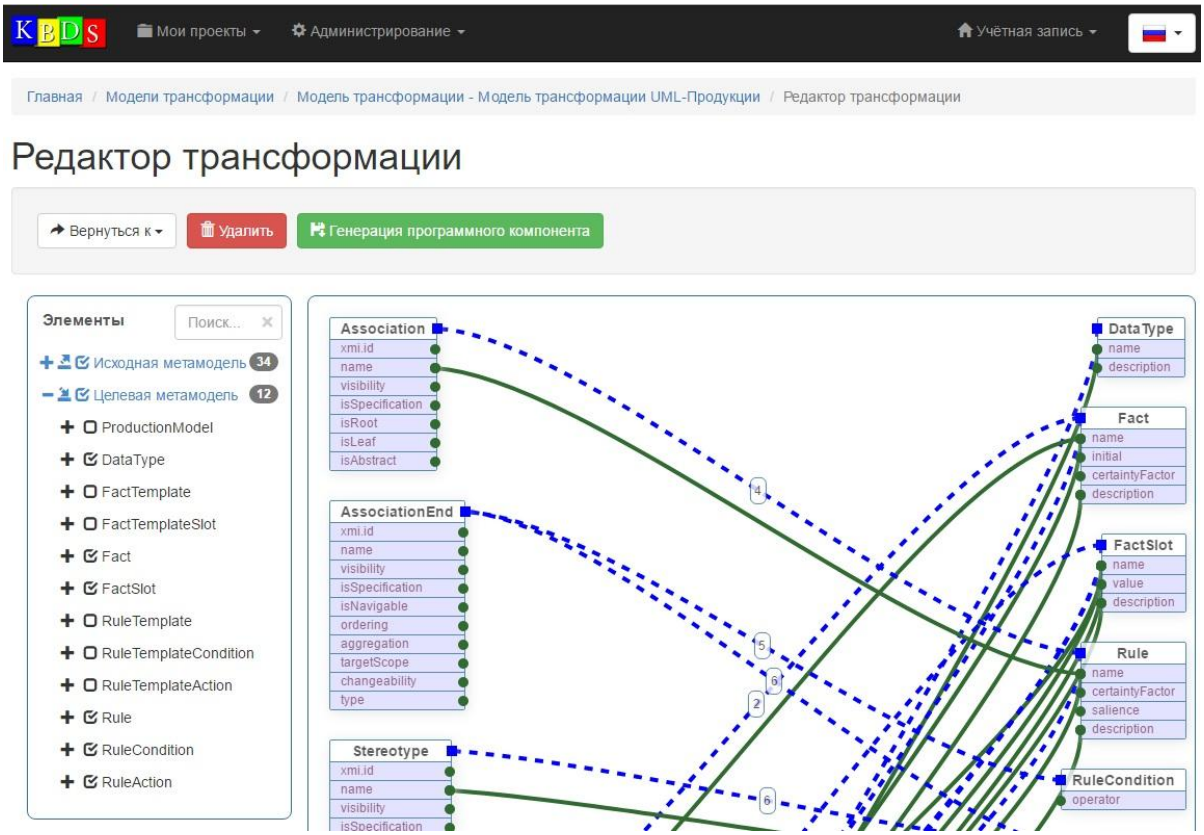

Рисунок 28. Экранная форма графического редактора модели трансформации (модель трансформации диаграмм классов UML в модель продукций)

Построенная полная модель трансформации содержит 6 правил, включая 1 простое бинарное правило трансформации и 5 комплексных (сложных) правил трансформации. Подробнее рассмотрим построенную полную модель трансформации.

На Рисунке 29 показано простое бинарное правило трансформации (с приоритетом равным 1), которое отражает взаимно-однозначное (эквивалентное) соответствие между элементами исходной и целевой метамодели (соответствие между типами данных).

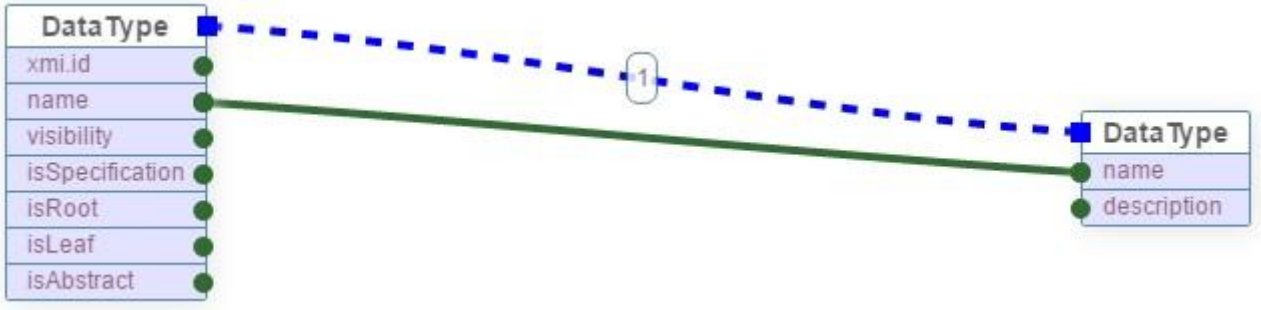

Рисунок 29. Пример простого бинарного правила трансформации

На Рисунке 30 показаны два комплексных (сложных) n-арных правила трансформации (с приоритетом равным 2 и 3 соответственно). Правило с приоритетом 2 устанавливает соответствие между классом UML и фактом модели продукций. Правило с приоритетом 3 устанавливает соответствие между атрибутом класса UML и слотом факта в модели продукций. При этом элемент «ModelElement.name» отражен сразу в два элемента, т.к. он может являться составной частью как класса, так и его атрибута (связь «часть-целое» $R^{cm}_{part-of}$ в исходной метамодели диаграммы классов XMI UML). Следует отметить, что слоту факта соответствует сразу два элемента исходной метамодели, а именно – атрибут «Attribute» и значение (выражение) «Expression», таким образом, данное правило выражает неоднозначное соответствие между элементами (синонимия отношений).

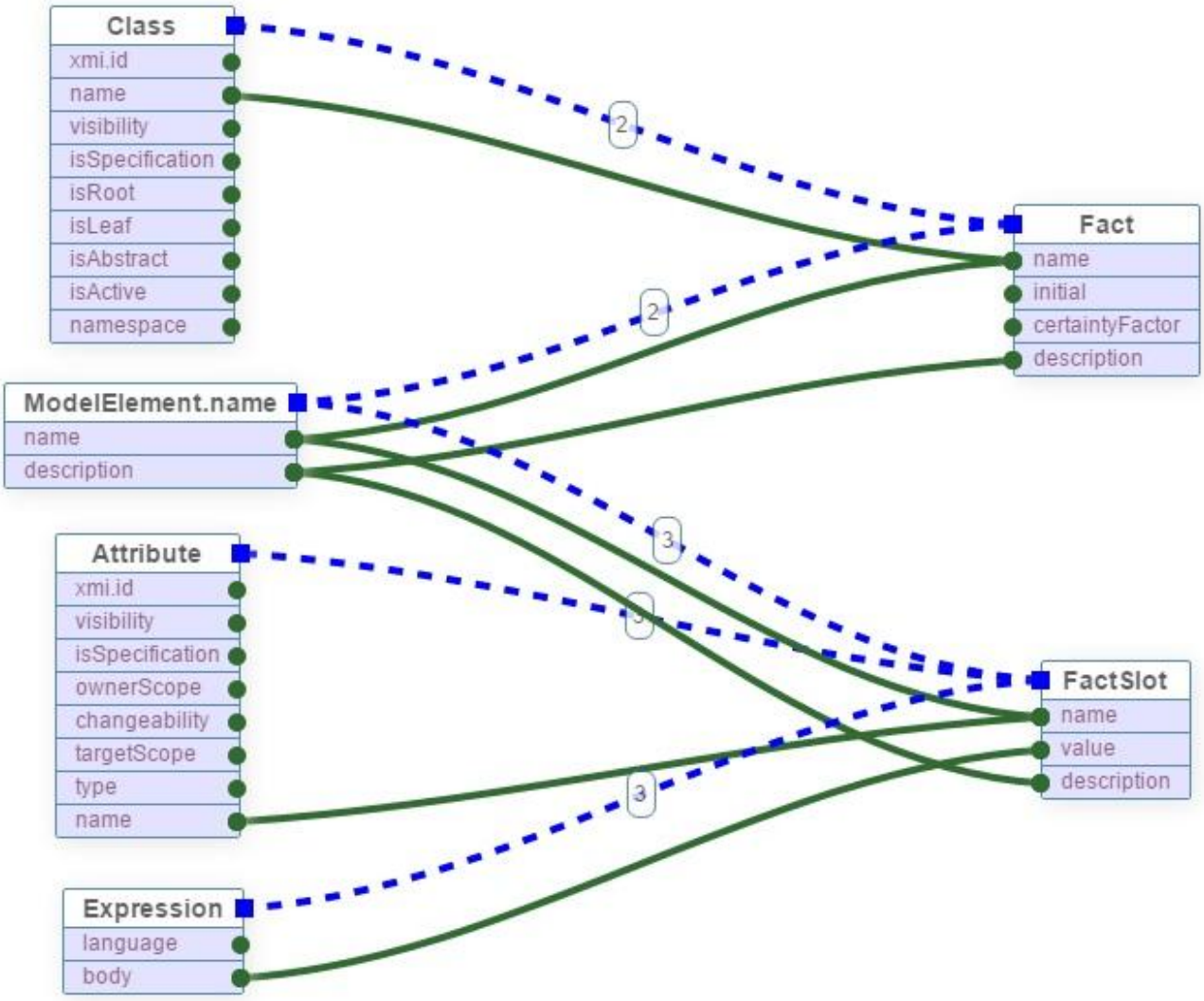

Рисунок 30. Пример комплексного (сложного) правила трансформации (соответствие класс-факт и атрибут-слот)

На Рисунке 31 показаны три комплексных (сложных) n-арных правила трансформации (с приоритетом равным 4, 5 и 6 соответственно). Правило с приоритетом 4 устанавливает соответствие между ассоциацией UML и правилом модели продукций. Правила трансформации с приоритетом 5 и 6 устанавливает соответствие части ассоциации UML с условием и действием в модели продукций. При этом исходный элемент – часть ассоциации UML – соответствует сразу двум элементам целевой метамодели, а именно, условию продукционного правила и действию продукционного правила, таким образом, данное правило выражает неразличимое соответствие между элементами (омонимия отношений).

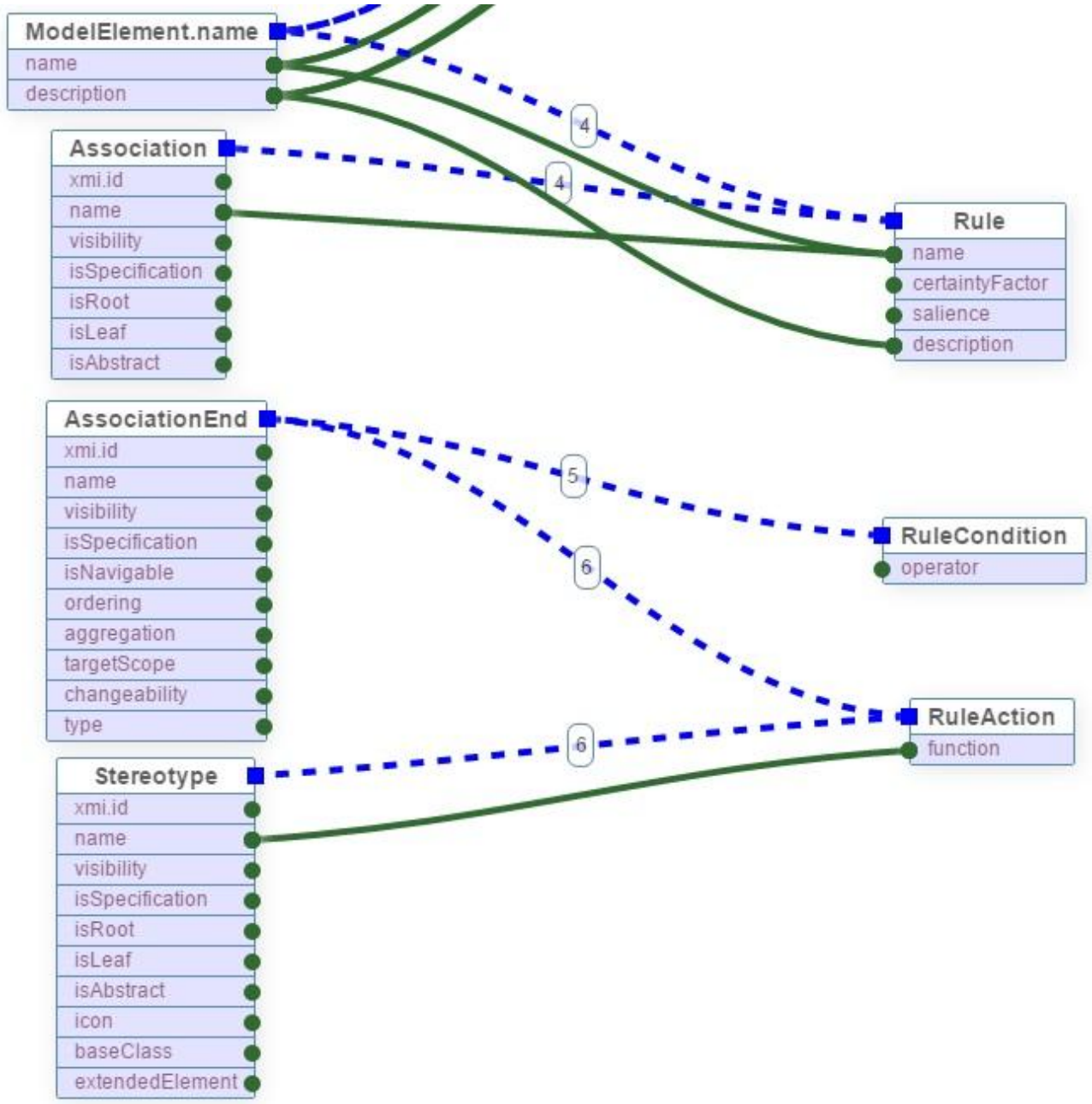

Рисунок 31. Пример комплексного (сложного) правила трансформации (соответствие ассоциация-правило, часть ассоциации-условие и -действие)

После того как пользователь определит все необходимые соответствия, автоматически сгенерируется код модели трансформации на TMRL. На Рисунке 32 показана экранная форма (веб-страница) просмотра сгенерированного TMRL-кода модели трансформации для преобразования диаграммы классов UML в модель продукций.

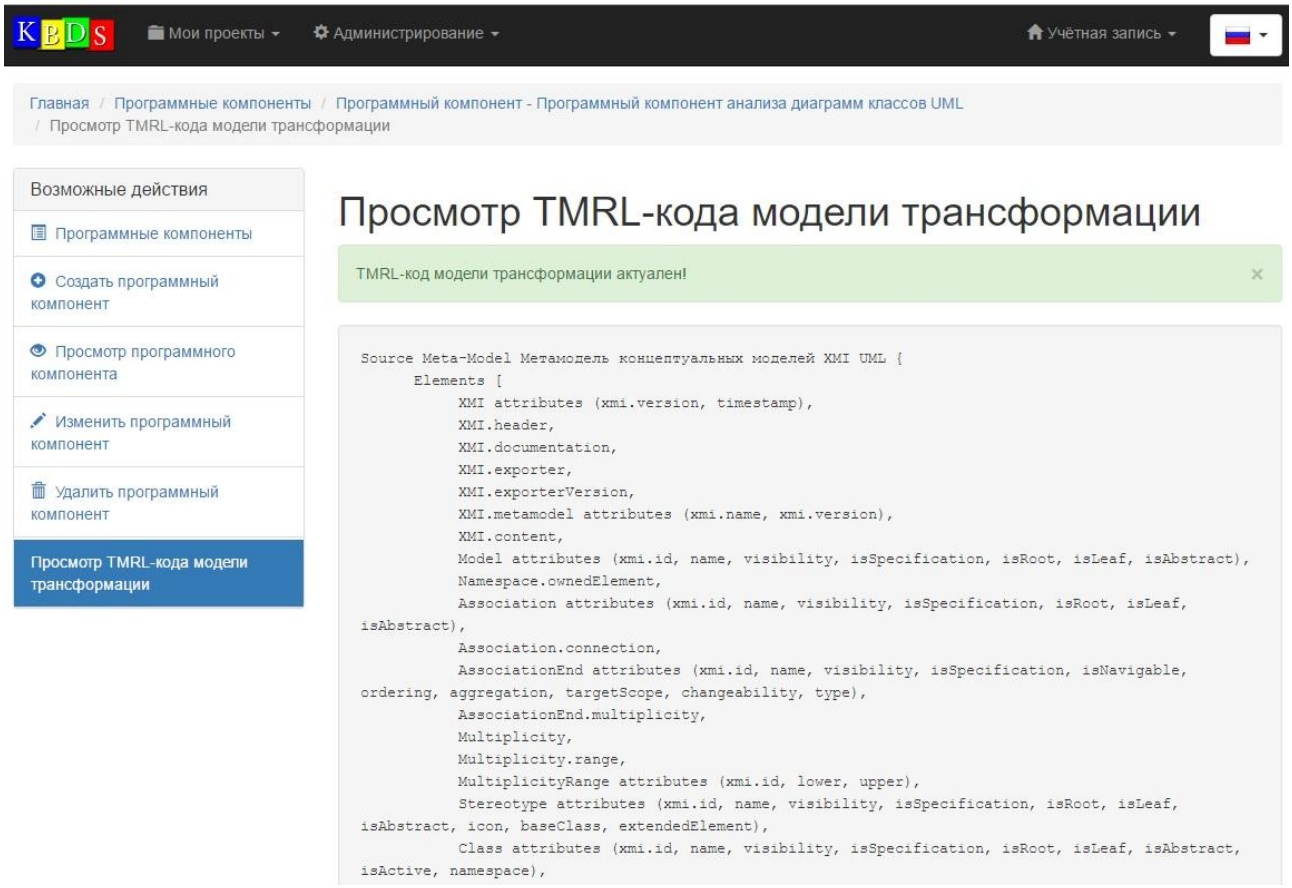

Рисунок 32. Экранная форма просмотра TMRL-кода модели трансформации диаграмм классов UML в модель продукций

Однако в сгенерированной модели трансформации могут отсутствовать необходимые правила, т.е. модель может быть не полной. Эти правила не могут быть установлены при помощи визуального редактора («сложное правило определения» – данный вид правил обсуждался в главе 2.3). Уточнить (доопределить) модель трансформации возможно только при помощи специального текстового редактора TMRL вручную.

Полный листинг уточненной модели трансформации диаграммы классов UML в модель продукций, представленной на TMRL, приводится в Приложении Г.

Далее осуществляется автоматическая сборка (синтез) программного компонента из соответствующих готовых блоков анализатора и генератора (исходя из выбранного ранее типа программного компонента) и сгенерированного TMRL-кода модели трансформации.

Таким образом, разработанный программный компонент автоматически добавляется в набор всех прикладных компонентов трансформации, содержащихся в составе веб-ориентированной программной системы.

Данный программный компонент обеспечивает трансформацию исходных диаграмм классов UML в модель продукций (преобразование T_{CM-UM}). Для генерации целевой БЗ в формате CLIPS необходимо воспользоваться другим (дополнительным) уже готовым программным компонентом, обеспечивающим трансформацию модели продукций в целевой код БЗ на ЯПЗ CLIPS (преобразование T_{UM-KB}).

На создание данного программного компонента была затрачена 41 минута, включая:

1. Создание метамодели для исходной концептуальной модели (определение и добавление недостающих связей между элементами) – 10 мин.
2. Выбор целевой метамодели БЗ (создание проекта нового программного компонента и определение его типа) – 4 мин.
3. Создание модели трансформации – 27 мин., включая:
    - анализ структуры метамоделей (автоматическое выделение элементов) – осуществляется автоматически;
    - построение предварительной модели трансформации (без правил трансформации) – осуществляется автоматически;
    - построение полной модели трансформации (установление соответствий между элементами с использованием графического редактора) – 14 мин.;
    - генерация кода модели трансформации на TMRL – осуществляется автоматически;
    - уточнение (дополнение) кода модели трансформации – 13 мин.
4. Создание (сборка) программного компонента – осуществляется автоматически.

Следует отметить, что этапы 3.1, 3.2, 3.4 и 4 осуществляются системой автоматически и практически не требуют временных затрат. Также важно отметить, что самыми трудоемкими (и почти одинаковыми по времени) этапами при разработке данного программного компонента являются:

- этап 3.3, т.к. пользователю необходимо продумать и установить требуемые соответствия между элементами исходной и целевой метамодели;
- этап 3.5, т.к. на данном этапе осуществляется «ручное» текстовое редактирование (уточнение) TMRL-кода модели трансформации.

Для сравнения приведем пример решения данной задачи стандартным способом: путем разработки программистом специального модуля (конвертера) для анализа диаграмм классов UML из файлов IBM Rational Rose (MDL-формат) и генерации на их основе продукционной модели (см. Рисунок 5). Данный модуль реализован в среде программирования Delphi. Фрагмент листинга программного кода конвертора представлен в Приложении Д.

На решение данной задачи было затрачено 5 часов 12 минуты, включая:

- анализ структуры MDL-файла, который представляет собой структурированный текстовый файл, и выделение языковых конструкций, которые соответствуют элементам диаграмм классов UML – 35 мин.;
- разработку программного кода модуля – 3 часа 14 минут;
- тестирование и отладку программных кодов на тестовых примерах – 1 час 23 минуты.

В итоге разница по времени разработки составила: 4 часа 31 минута. Таким образом, разработка программного компонента (модуля) анализа диаграмм классов UML средствами KBDS показало свою эффективность в сравнение со стандартным программированием (быстрее в 7 раз).

Также важно отметить, что разработка программных компонентов с использованием KBDS направлена на более широкий круг пользователей (в основном за счет частичной автоматизации данного процесса и использования визуальных методов разработки), в отличие от стандартного решения задачи способом программирования, которое ориентированно только на программистов.

4.2. Разработка программного компонента анализа концепт-карт XTM

Для разработки программного компонента анализа концептуальных моделей в форме концепт-карт XTM [196] также использовалась разработанная веб-ориентированная программная система (KBDS) [19, 23, 31, 34, 40, 41, 244].

Процесс разработки соответствует этапам создания программного компонента, представленным на Рисунке 11.

Этапы (алгоритм) разработки программного компонента анализа концепт-карт XTM совпадают с этапами разработки программного компонента анализа концептуальных моделей в форме диаграмм классов UML, подробно описанными в предыдущем подразделе.

Поэтому кратко приведем основные этапы и время, затраченное на разработку прикладного программного компонента анализа концептуальных моделей в форме концепт-карт XTM:

- создание нового проекта программного компонента (включая выбор типа) – 2 мин.;
- построение (генерация) метамодели для исходных концепт-карт XTM – 4 мин.;
- создание новой модели трансформации и привязывание ее к создаваемому программному компоненту – 3 мин.;
- формирование правил трансформации с использованием графического редактора – 12 мин.;
- генерация кода модели трансформации на TMRL – осуществляется автоматически;
- уточнение кода модели трансформации (добавление сложных правил определения) – 5 мин.;
- финальная сборка (синтез) программного компонента (настройка типового программного компонента) на основе сгенерированной модели

трансформации и выбранных блоках анализатора и генератора (в соответствии с выбранным типом программного компонента) – осуществляется автоматически.

Таким образом, общее время разработки данного программного компонента составило 26 минут.

Следует отметить, что так же, как при разработке программного компонента анализа диаграмм классов UML, некоторые этапы осуществлялись автоматически и практически не требовали временных затрат. Самым трудоемким этапом оказался этап формирования правил трансформации (12 мин.). При этом на этапе уточнения TMRL-кода модели трансформации было затрачено всего 5 минут, т.к. не потребовалось вводить (кодировать) сложные правила.

В качестве наглядного примера приведем некоторые снимки экранных форм:

На Рисунке 33 показана экранная форма (веб-страница) графического редактора метамоделей после генерации метамодели на основе исходной концепт-карте XTM. Более детальная метамодель концепт-карт XTM показана на Рисунке 34.

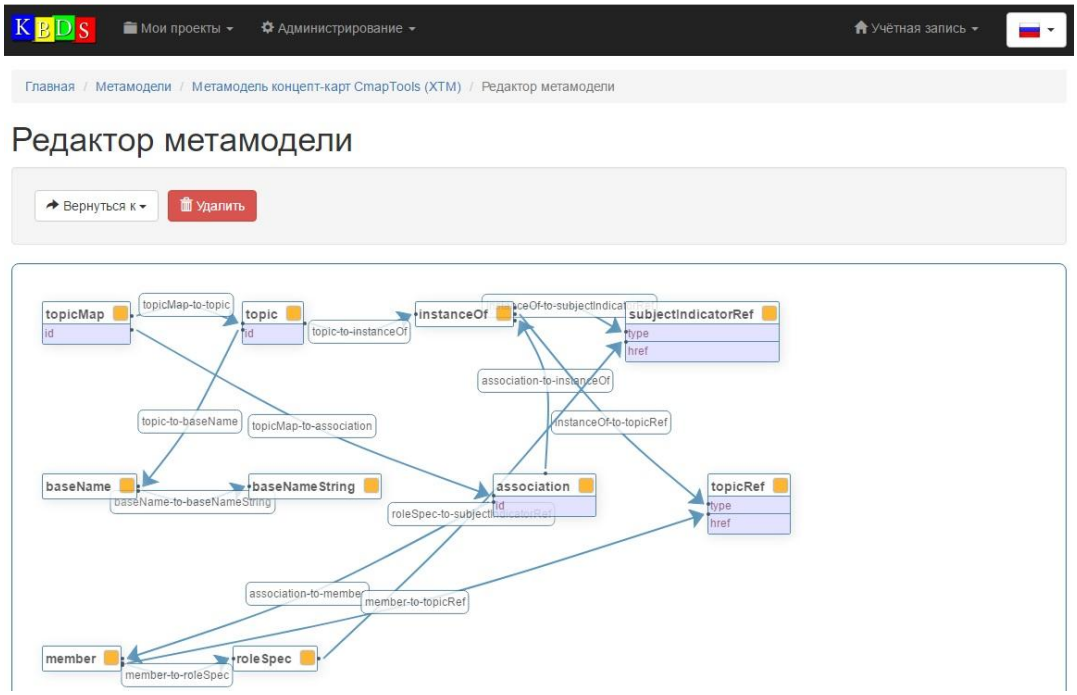

Рисунок 33. Экранная форма графического редактора метамоделей (метамодель концепт-карт XTM)

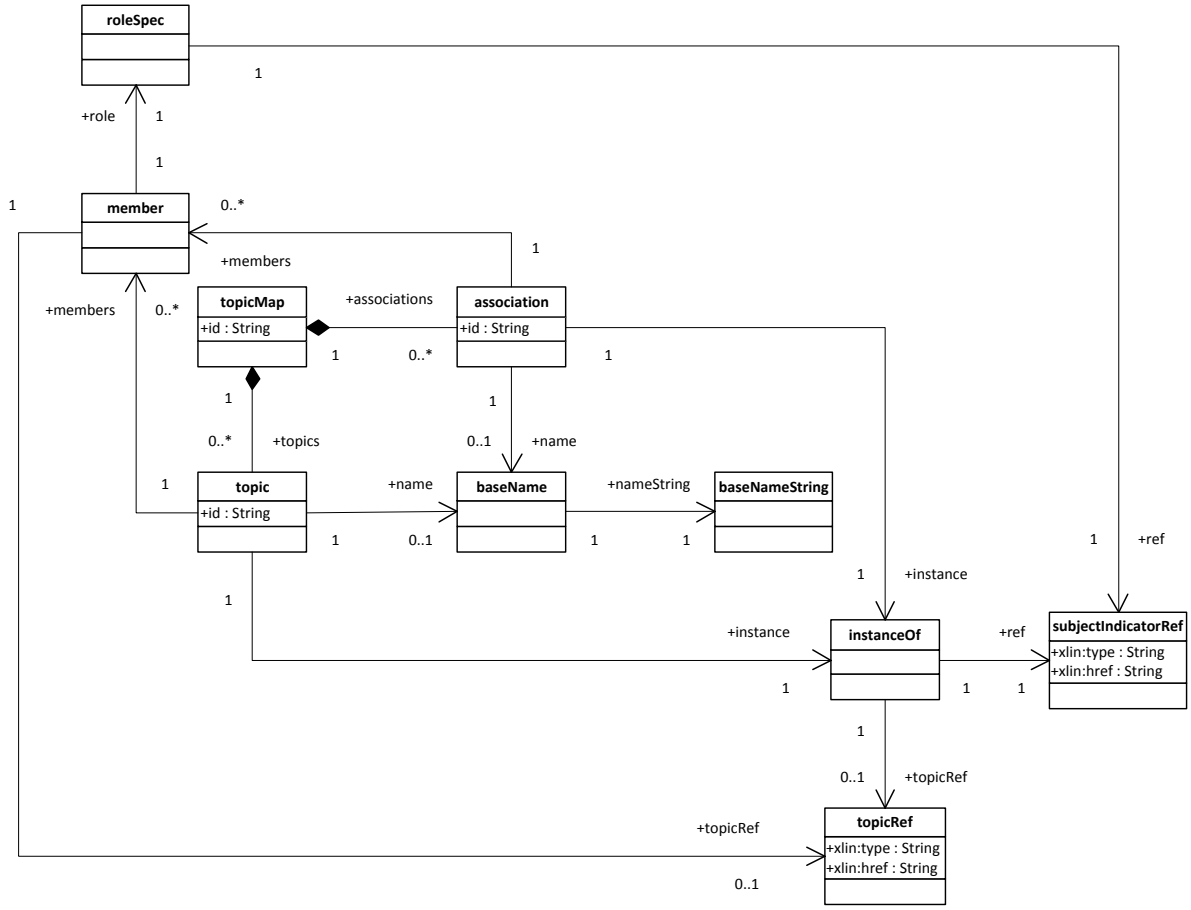

Рисунок 34. Метамодель исходной концепт-карты XTM

На Рисунке 35 показана экранная форма (веб-страница) графического редактора модели трансформации [37] с установленными соответствиями между элементами метамоделей исходной концепт-карты XTM и целевой модели продукций. Следует отметить, что если код модели трансформации и программного компонента был сгенерирован и после этого в модель трансформации внесены изменения, то статус программного компонента сменится на «устаревший». Программные компоненты с таким статусом все еще будут доступны для формирования БЗ, однако TMRL-код модели трансформации должен быть сгенерирован заново при помощи кнопки «Перегенерировать программный компонент».

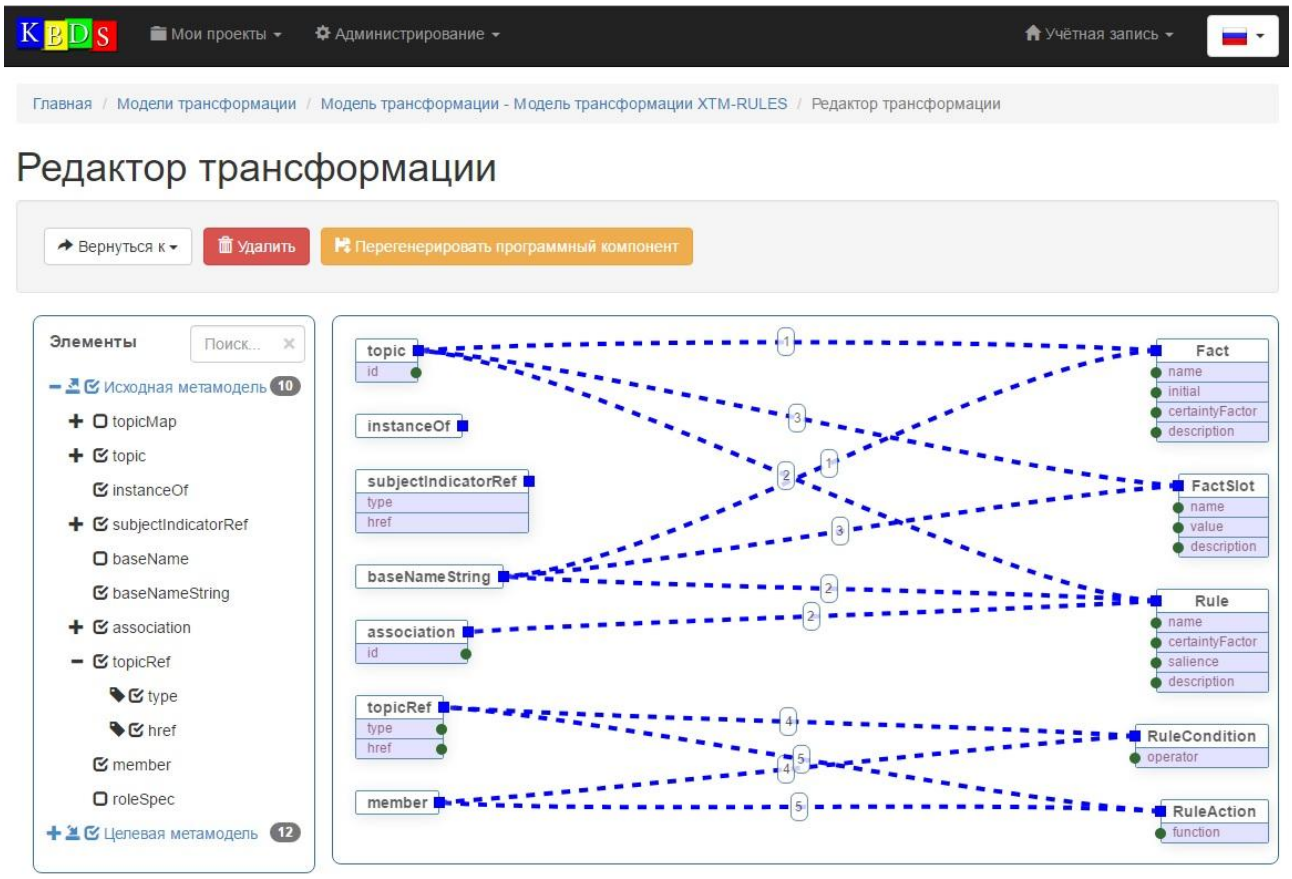

Рисунок 35. Экранная форма графического редактора модели трансформации концепт-карт XTM в модель продукций

Полный листинг уточненной модели трансформации концепт-карт XTM в модель продукций, представленной на TMRL, приводится в Приложении Е.

Таким образом, созданный программный компонент обеспечивает трансформацию исходных концепт-карт XTM в модель продукций (преобразование T_{CM-UM}). Для генерации целевой БЗ в формате CLIPS необходимо воспользоваться другим (дополнительным) уже готовым программным компонентом, обеспечивающим трансформацию модели продукций в целевой код БЗ на ЯПЗ CLIPS (преобразование T_{UM-KB}).

Для сравнения приведем пример решения данной задачи стандартным способом: путем разработки программистом специального модуля для анализа концепт-карт из файлов IHMC CmapTools (CXL-формат) и генерации на их основе продукционной модели (см. Рисунок 5). Данный модуль реализован в

среде программирования Delphi. Фрагмент листинга программного кода конвертора представлен в Приложении Ж.

На решение данной задачи было затрачено 3 часа 30 минут, включая:

- анализ структуры CXL-файла, который представляет собой структурированный текстовый файл, и выделение языковых конструкций, которые соответствуют элементам концепт-карт CmapTools – 30 мин.;
- разработку программного кода модуля – 2 часа;
- тестирование и отладку программных кодов на тестовых примерах – 1 час.

В итоге разница по времени разработки составила: 3 часа 4 минуты. Таким образом, разработка программного компонента (модуля) анализа концепт-карт CmapTools (XTM) средствами KBDS показало свою эффективность в сравнение со стандартным программированием (быстрее в 8 раз).

4.3. Разработка программного компонента анализа деревьев событий

Для разработки программного компонента анализа концептуальных моделей в форме ДС [24], применяемых в области анализа отказов и риска технических систем, также использовалась разработанная веб-ориентированная программная система (KBDS) [19, 23, 31, 34, 40, 41, 244].

Процесс разработки соответствует этапам создания программного компонента, представленным на Рисунке 11.

Этапы (алгоритм) разработки программного компонента анализа ДС совпадают с этапами разработки программного компонента анализа концептуальных моделей в форме диаграмм классов UML и концепт-карт XTM, подробно описанными в предыдущих подразделах.

Поэтому кратко приведем основные этапы и время, затраченное на разработку прикладного программного компонента анализа концептуальных моделей в форме ДС:

- создание нового проекта программного компонента (включая выбор типа) – 2 мин.;
- построение (генерация) метамодели для исходных концепт-карт XTM – 5 мин.;
- создание новой модели трансформации и привязывание ее к создаваемому программному компоненту – 2 мин.;
- формирование правил трансформации с использованием графического редактора – 10 мин.;
- генерация кода модели трансформации на TMRL – осуществляется автоматически;
- уточнение кода модели трансформации (добавление сложных правил определения) – 10 мин.;
- финальная сборка (синтез) программного компонента (настройка типового программного компонента) на основе сгенерированной модели трансформации и выбранных блоках анализатора и генератора (в соответствии с выбранным типом программного компонента) – осуществляется автоматически.

Таким образом, общее время разработки данного программного компонента составило 29 минут.

Следует отметить, что при разработке данного программного компонента анализа ДС некоторые этапы осуществлялись автоматически и практически не требовали временных затрат. Самым трудоемким этапом оказался этап формирования правил трансформации (20 мин.). В качестве наглядного примера приведем некоторые снимки экранных форм:

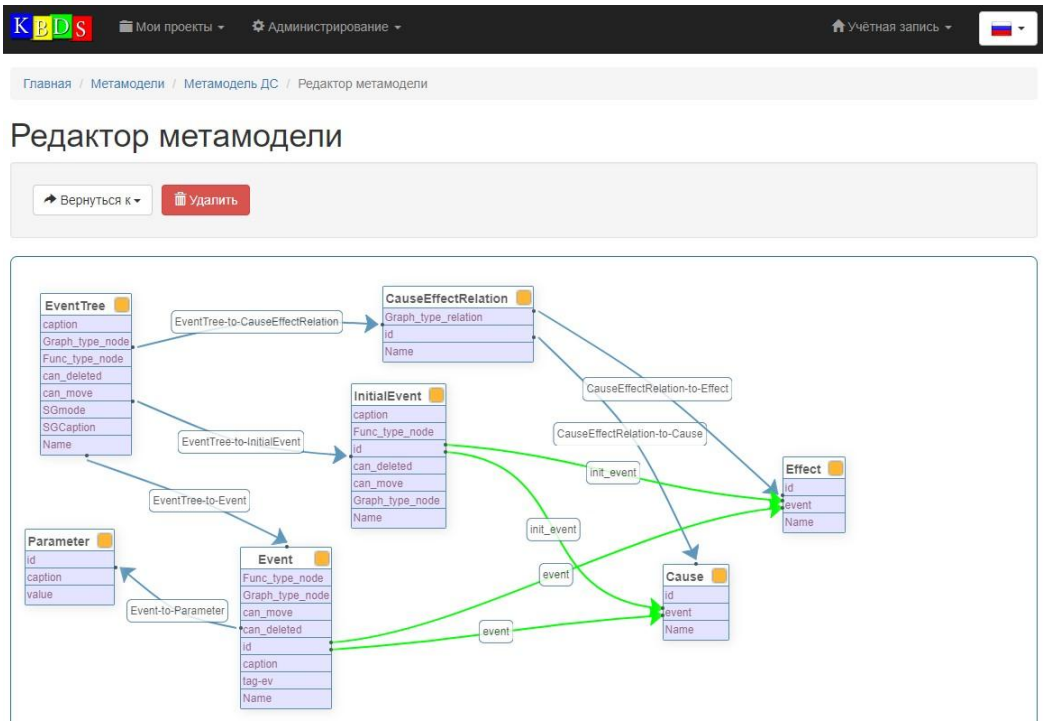

Рисунок 36. Экранная форма графического редактора метамоделей (метамодель ДС)

На Рисунке 36 показана экранная форма (веб-страница) графического редактора метамоделей после генерации метамодели на основе исходного ДС.

На Рисунке 37 показана экранная форма (веб-страница) графического редактора модели трансформации [37] с установленными соответствиями между элементами метамоделей исходного ДС и целевой модели продукций.

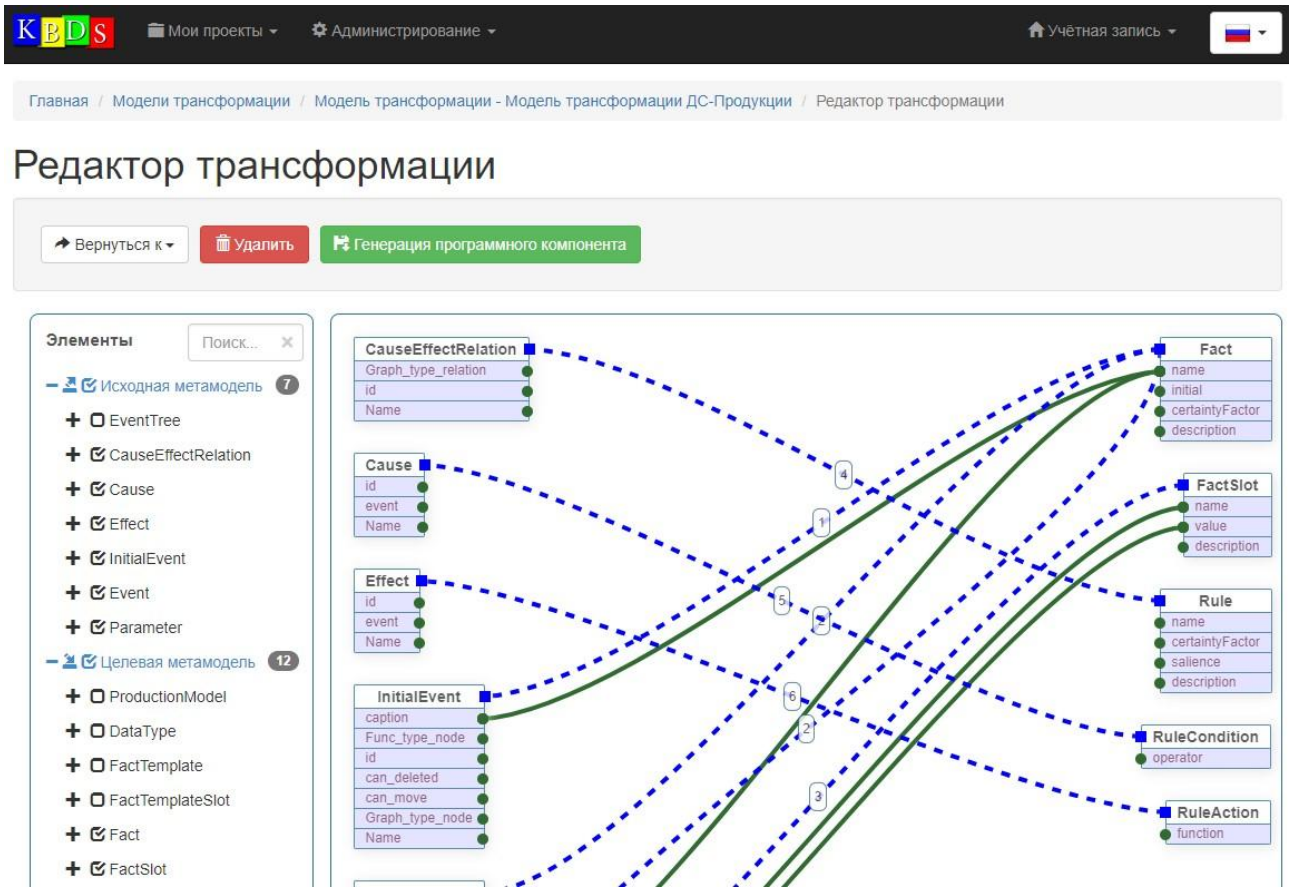

Рисунок 37. Экранная форма графического редактора модели трансформации ДС в модель продукций

Полный листинг уточненной модели трансформации ДС в модель продукций, представленной на TMRL, приводится в Приложении 3.

Таким образом, созданный программный компонент обеспечивает трансформацию исходных ДС в модель продукций (преобразование T_{CM-UM}). Для генерации целевой БЗ в формате CLIPS необходимо воспользоваться другим (дополнительным) уже готовым программным компонентом, обеспечивающим трансформацию модели продукций в целевой код БЗ на ЯПЗ CLIPS (преобразование T_{UM-KB}).

4.4. Разработка баз знаний для прогнозирования развития деградационных процессов в нефтехимии

Повышение безопасности промышленных объектов с каждым годом становится все более актуальной проблемой, так как старение оборудования во многих отраслях промышленности опережает темпы его модернизации и замены, как по объективным, так и субъективным причинам. Особенно это относится к длительно эксплуатирующемуся нефтеперерабатывающему, нефтехимическому и химическому оборудованию. В связи с этим, возникает необходимость 100% обследования такого оборудования с целью определения потенциально опасных зон и принятия соответствующих решений для исключения катастрофических отказов. Качество решения этих задач определяет величину потерь при авариях, стоимость проведения диагностирования, а также периодических и восстановительных ремонтов.

Существенное повышение надежности и безопасности нефтехимического оборудования можно обеспечить разработкой и использованием методов и средств искусственного интеллекта, в частности ЭС. ЭС могут быть применены для интерпретации условий и параметров эксплуатации, последующего обоснования программы технического диагностирования, интерпретации параметров диагностирования, которые обеспечат качественную оценку технического состояния, установление причин его изменения, объем и содержание восстановительного ремонта, а также экспертизу промышленной безопасности (ЭПБ).

В частности, процедура ЭПБ состоит из следующих основных этапов:

- планирование работ для ЭПБ;
- анализ технической документации;
- формирование карты исходных данных;
- разработка программы ЭПБ;

- техническая диагностика и анализ (в том числе и интерпретация) результатов;
- вычисление остаточного ресурса;
- принятие решений по ремонту.

Анализ данной процедуры показал, что реализация этапов «разработка программы ЭПБ», «анализ (в том числе и интерпретация) результатов», «принятие решений по ремонту» требует обработки больших объемов плохо формализованной информации. При этом эффективность данной обработки может быть повышена с помощью ЭС, которые позволят:

- интерпретировать условия и параметры функционирования;
- обосновать программу технической диагностики;
- интерпретировать параметры диагностики.

Для решения данных задач может быть использован формализм продукционных ЭС, позволяющий описать причинно-следственный комплекс (ПСК) изменения технического состояния элементов исследуемого оборудования.

ПСК [255] отражает взаимосвязь воздействующих факторов, условий эксплуатации, ошибок и несовершенств, имеющих место при проектировании, изготовлении и эксплуатации деталей, и возникающих вследствие этого деградационных процессов. ПСК обусловлен структурой перехода состояний объекта при его функционировании и представлен логической моделью. Выделены следующие классы состояний: исходная дефектность, поврежденность, разрушение, отказ. Каждое из состояний описывается набором параметров и их значений. Изменение технического состояния обусловлено деградационными процессами, протекающими в объекте исследования под влиянием совокупности воздействующих факторов, и представлено соответствующими стадиями этого процесса.

В свою очередь, полная и непротиворечивая БЗ диагностической или прогностической ЭС может быть создана лишь путем использования структурированной процедуры экспертного опроса, позволяющей получать от

эксперта всю необходимую для заполнения БЗ информацию [257]. При этом результатом этапа извлечения (получения) и структурирования экспертных знаний, как правило, является множество концептуальных моделей предметной области, представленных в различных «предметных» нотациях.

Таким образом, в данном случае, для разработки БЗ прогнозирования развития деградационных процессов аппаратов в нефтехимии, реализован экспериментальный программный компонент, обеспечивающий генерацию кода БЗ на ЯПЗ CLIPS, на основе трансформации диаграмм классов UML [23, 27, 29, 34], концепт-карт (тематических карт) XTM [43] и ДС [42].

Процесс создания БЗ определяется, исходя из этапов методики автоматизированной разработки БЗ на основе трансформации концептуальных моделей [15, 21, 22, 24-26, 28, 30, 32, 33, 35, 36], представленных на Рисунке 14.

Приведем некоторые примеры фрагментов данных концептуальных моделей, представленных в форме ДС, на основе которых разработана продукционная БЗ ЭС прогнозирования техногенных чрезвычайных ситуаций в нефтехимии.

ДС – алгоритм рассмотрения событий, исходящих от основного события (аварийной ситуации). ДС используется для определения и анализа последовательности (вариантов) развития аварии, включающей сложные взаимодействия между техническими системами обеспечения безопасности. При его построении используется прямая логика. В общем случае данный метод можно использовать и для анализа отказов, аварий и чрезвычайных ситуаций, где в качестве основного события рассматривается исходное состояние, т.е. состояние технического объекта в момент начала его эксплуатации.

Исходные ДС создавалась с использованием специального графического редактора TreeEditorET и представлены на Рисунке 38.

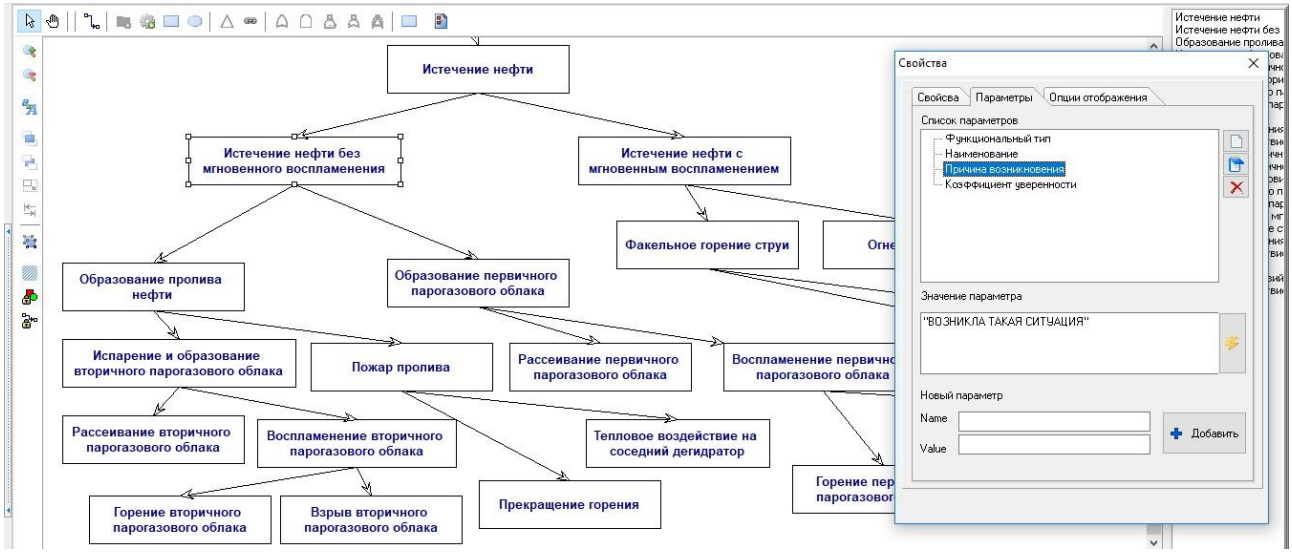

Рисунок 38. Экранная форма графического редактора ДС

Фрагмент XML-кода построенного ДС представлен в Листинге 9.

Листинг 9. Фрагмент XML-кода построенного ДС

```
<EventTree caption="Дерево событий"
  Graph_type_node="TRectangularNode"
  Func_type_node="system_EventTree"
  can_deleted="true" can_move="false" SGmode="tmEventTree"
  SGCaption="Дерево событий" Name="8">
  <InitialEvent caption="Истечение нефти"
    Func_type_node="eventET"
    id="1" can_deleted="true" can_move="true"
    Graph_type_node="TRectangularNode" Name="9"/>
  <Event Func_type_node="eventET"
    Graph_type_node="TRectangularNode" can_move="true"
    can_deleted="true" id="3" caption="Истечение нефти
    без мгновенного воспламенения" tag-ev="0" Name="10">
    <Parameter id="p1" caption="probability" value="0,95"/>
    <Parameter id="p2" caption="course" value=
      "ВОЗНИКЛА ТАКАЯ СИТУАЦИЯ"/>
    <Parameter id="p3" caption="cf" value="0,9"/>
</Event>
<CauseEffectRelation Graph_type_relation="TDirectedArrow"
  id="40" Name="32">
```

```
20    <Cause id="41" event="1" Name="33"/>
21    <Effect id="42" event="3" Name="34"/>
22 </CauseEffectRelation>
```

Фрагменты ДС, описывающие развитие аварии «Истечение нефти», представлены на Рисунке 39 и 40.

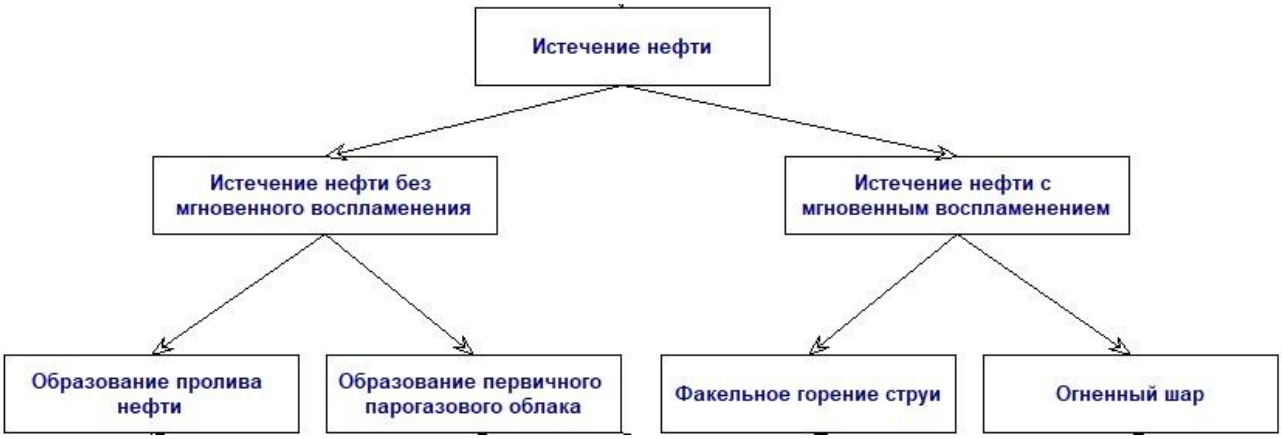

Рисунок 39. Фрагмент ДС развития аварии «истечение нефти» (1)

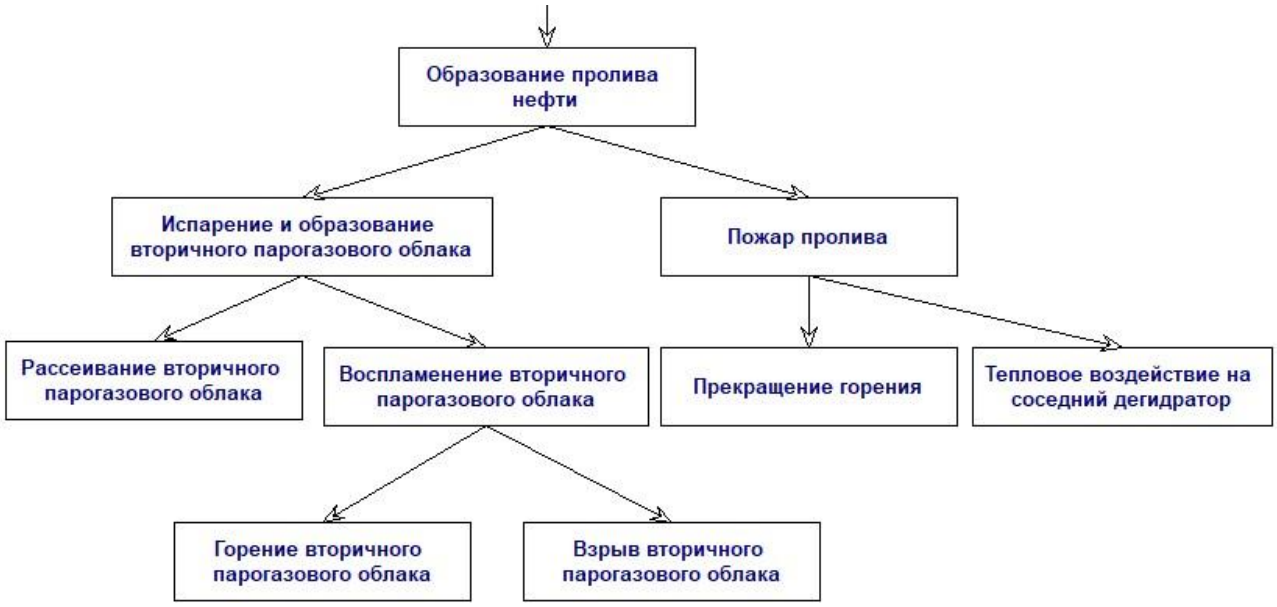

Рисунок 40. Фрагмент ДС развития аварии «истечение нефти» (2)

На основе данного ДС получен набор шаблонов фактов и правил. Примеры некоторых полученных RVML-диаграмм представлены на Рисунках 41-44.

Рисунок 41. RVML-диаграмма развития аварии «истечение нефти» (1)

Рисунок 42. RVML-диаграмма развития аварии «истечение нефти» (2)

Рисунок 43. RVML-диаграмма развития аварии «истечение нефти» (3)

Рисунок 44. RVML-диаграмма развития аварии «истечение нефти» (4)

Таким образом, на основе сформированной модели продукций получим сгенерированный код БЗ в формате CLIPS, представленный в Листинге 10.

Листинг 10. Фрагмент сгенерированного кода БЗ в формате CLIPS

```
;************************* Шаблоны ************************
(deftemplate Istechenie nefti bez mgnovennogo vosplameneniya
  (slot probability (default ""))
  (slot course (default ""))
)
(deftemplate Obrazovanie pervichnogo parogazovogo oblaka
  (slot probability (default ""))
  (slot course (default ""))
)
(deftemplate Obrazovanie proliva nefti
  (slot probability (default ""))
  (slot course (default ""))
)
;************************* правила ************************
(defrule Istechenie nefti bez mgnovennogo vosplameneniya->
  Obrazovanie proliva nefti+Obrazovanie pervichnogo
  parogazovogo oblaka
  (declare (salience 2))
  (Istechenie nefti bez mgnovennogo vosplameneniya
```

```
20       (probability "0,95")
21       (course "VOZNIKLA TAKAYa SITUACIYa")
22    )
23 =>        ;TO
24    (assert
25       (Obrazovanie pervichnogo parogazovogo oblaka
26          (probability "0,5")
27          (course "ISHOD ISTEChENIYa NEFTI")
28       )
29    )
30    (assert
31       (Obrazovanie proliva nefti
32          (probability "0,45")
33          (course "REZULTAT ISTEChENIYa NEFTI")
34       )
35    )
36 )
```

Далее рассмотрим созданную продукционную БЗ для ЭС прогнозирования развития деградационных процессов аппаратов в нефтехимии [39].

Деградационные процессы описываются в виде ПСК, представленного механизмом, кинетикой и параметрами, определив которые можно перейти к обоснованию решений для предупреждения рассмотренных деградационных процессов или снижения их скорости или обоснования диагностирования для своевременной замены деградирующих элементов для предупреждения разрушений, последующих отказов и аварий [254-256].При этом можно выделить некоторые виды деградационных процессов [254]:

- коррозионное растрескивание;
- коррозионная усталость;
- механическая усталость;
- водородное охрупчивание;
- радиационное охрупчивание;
- водородное растрескивание;

- изнашивание.

Приведем общее описание механизма деградационного процесса:

ЕСЛИ объект (свойство – материал) **И** воздействующий фактор (окружающая среда) **И** воздействующий фактор (технологическая среда) **И** воздействующий фактор (нагрузка) **ТО** механизм деградационного процесса при повреждении

При этом данное общее правило (метаправило) определяет структуру конкретного (частного) правила.

ЕСЛИ механизм деградационного процесса при повреждении **ТО** кинетика деградационного процесса при повреждении

ЕСЛИ кинетика деградационного процесса при повреждении **ТО** кинетика деградационного процесса при повреждении **И** событие 1

…

ЕСЛИ кинетика деградационного процесса при повреждении **И** событие *i* ТО кинетика деградационного процесса при повреждении **И** событие *i*+1

ЕСЛИ кинетика деградационного процесса при повреждении **ТО** признаки/параметры деградационного процесса при повреждении

ЕСЛИ механизм деградационного процесса при повреждении **И** признаки/параметры повреждения **И** объект (свойства) **ТО** механизм деградационного процесса при разрушении

ЕСЛИ механизм деградационного процесса при разрушении **И** признаки/параметры разрушения **И** объект (свойства) **ТО** механизм деградационного процесса при отказе

Таким образом, на основе данного общего описания механизма деградационного процесса разработан набор концептуальных моделей с описанием деградационных процессов в нефтехимическом оборудовании, в частности, коррозионной усталости, коррозионного растрескивания и водородного охрупчивания на стадии повреждения.

В качестве исходных концептуальных моделей использованы ДС, расширенная модель которых дополнена стадиями развития исследуемых

процессов отказов и аварий (субмикроуровень, микроуровень, мезоуровень, макроуровень) и элементами их описания (механизм и кинетика) [258]. Расширенная модель ДС (шаблон дерева), описывающая стадии, последовательность событий (кинетика) и механизмы их возникновения, представлена на Рисунке 45.

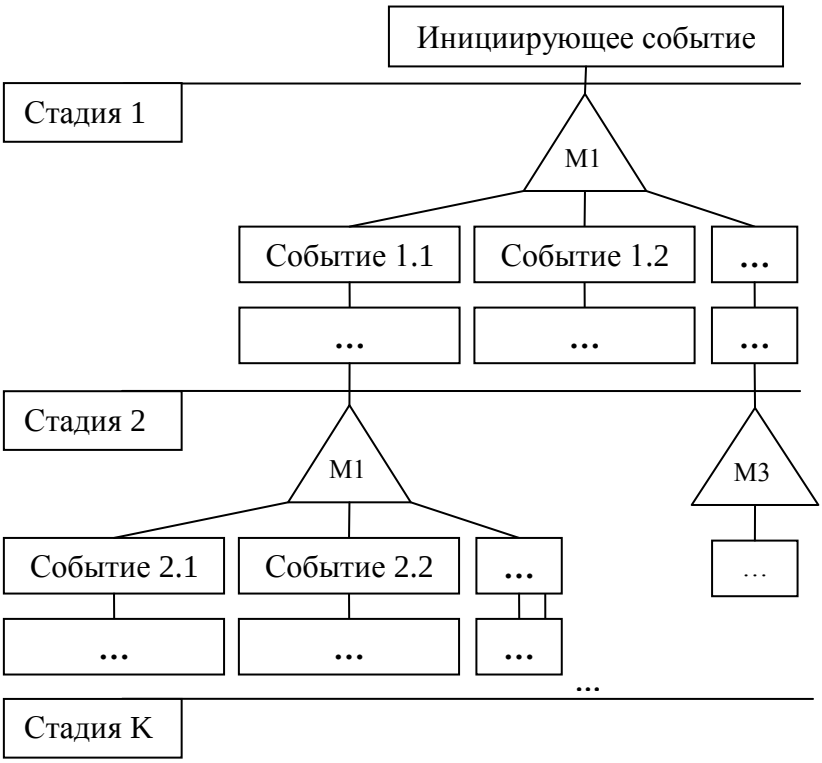

Рисунок 45. Шаблон расширенной модели ДС

Особо необходимо сказать о механизме процесса (в визуальной нотации ДС – это треугольник). Механизм процесса – это совокупность свойств объекта и факторов, воздействующих на него. Механизм может быть описан в виде продукции (правила) следующей структуры (шаблона правила) [258]:

ЕСЛИ (Свойство объекта$_1$ **И**…**И** Свойство объекта$_n$) **И**

(Воздействующий фактор$_1$ **И**…**И** Воздействующий фактор$_n$)

ТО Механизм *j*-стадии развития *i*-исследуемого процесса **И** (Событие$_1$ $\circ$…$\circ$ Событие$_n$), где $\circ$ – некоторая логическая операция, $\circ \in \{\wedge, \vee, \oplus\}$.

Метамодель (abstract syntax) расширенной модели ДС, определяющая в абстрактной форме основные концепты из которых состоит ДС, а также описание XML-формата (concrete syntax) хранения ДС приводятся в [258].

Рассмотрим разработку продукционной БЗ в формате CLIPS на примере фрагмента ДС, представленного на Рисунке 46 и описывающего стадию повреждения деградационного процесса – коррозионная усталость фланцевого соединения (Ду 6) в трубопроводе обвязки компрессора.

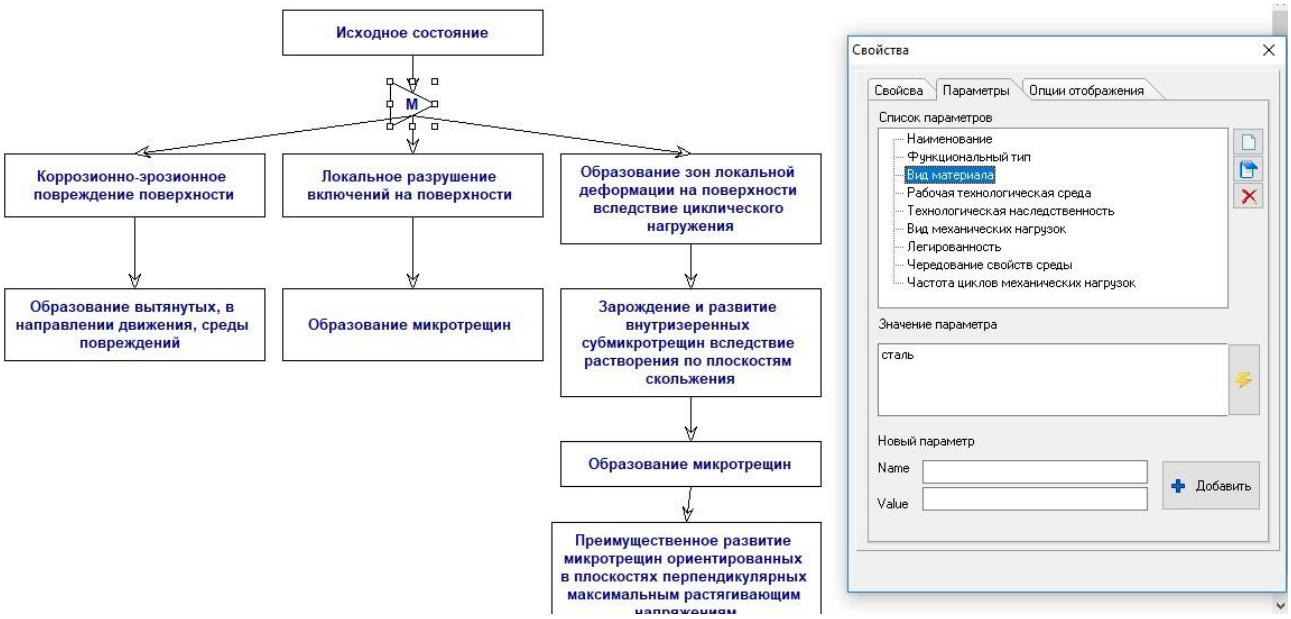

Рисунок 46. Пример фрагмента ДС коррозионной усталости

Графическому представлению ДС (см. Рисунок 46) соответствует фрагмент XML-документа (механизм деградационного процесса), представленного в Листинге 11. При этом выделены ключевые конструкции, на основе которых осуществлялось извлечение необходимых элементов ДС.

Листинг 11. Фрагмент XML-документа, описывающего механизм деградационного процесса

```
1 <Mechanism id="MEC-1" name="Механизм повреждения" event="IE-1">
2   <Operator id="OPR-1" name="AND">
3     <ObjectProperty id="OP-1" name="Вид материала"
4       value="сталь"/>
5     <ObjectProperty id="OP-2" name="Легированность"
6       value="низколегированна сталь"/>
```

```
    <ObjectProperty id="OP-3" name="Технологическая
      наследственность" val-ue="дефекты изготовления ИЛИ
      повреждаемость поверхности вследствие воздействия
      агрессивной среды"/>
    <InfluencingFactor id="IF-1" name="Рабочая технологическая
      среда" value="активная"/>
    <InfluencingFactor id="IF-2" name="Вид механических
      нагрузок" value="переменные"/>
    <InfluencingFactor id="IF-3" name="Чередование свойств
      среды" value="да"/>
    <InfluencingFactor id="IF-4" name="Частота циклов
      механических нагрузок" value="высокая (> 60 цикл/мин)"/>
  </Operator>
  <ProcessMechanism id="PM-1" name="Коррозионная усталость"/>
</Mechanism>
```

На основе извлеченных элементов ДС сгенерирована модель продукций, которая может быть представлена в нотации RVML на Рисунках 47-49 для дальнейшего уточнения полученных правил.

Рисунок 47. Пример полученной продукции в нотации RVML (1)

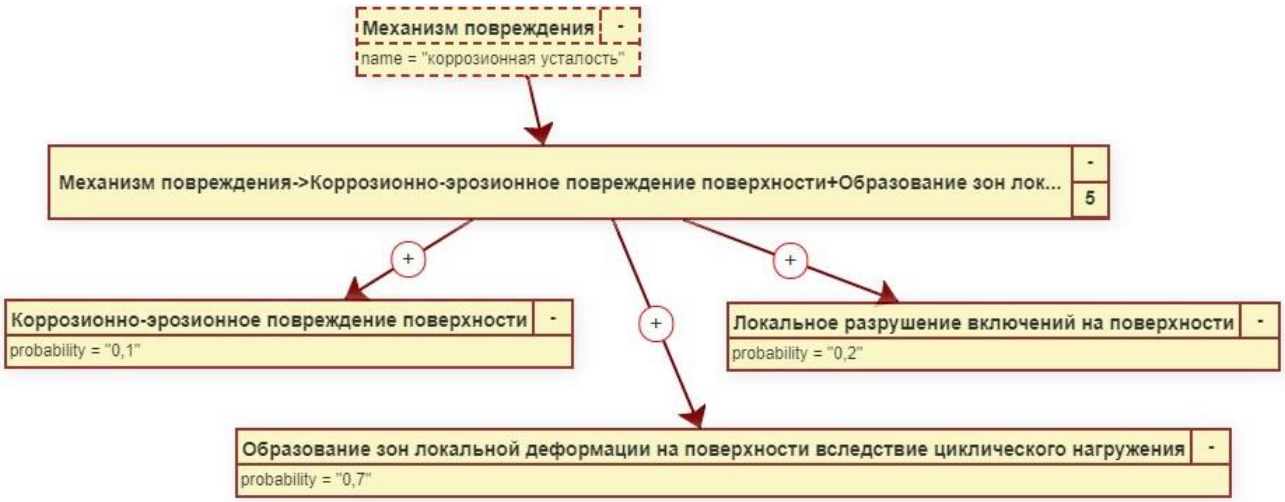

Рисунок 48. Пример полученной продукции в нотации RVML (2)

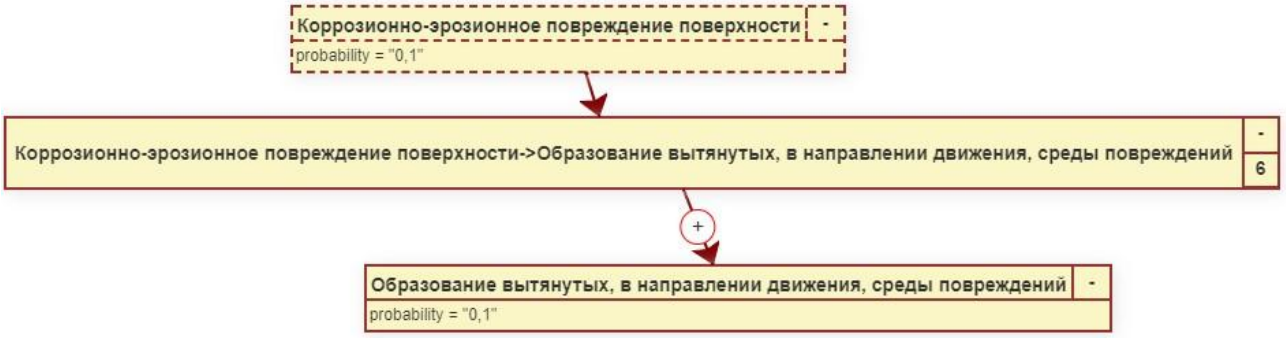

Рисунок 49. Пример полученной продукции в нотации RVML (3)

Таким образом, на основе сформированной модели продукций получим сгенерированный код БЗ в формате CLIPS (фрагмент – правило описания механизма деградационного процесса «коррозионная усталость»), представленный в Листинге 12.

Листинг 12. Фрагмент сгенерированного код БЗ в формате CLIPS (правило описания механизма деградационного процесса «коррозионная усталость»)

```
(defrule Vid materiala+Rabocha tehnologicheska
  sreda+Tehnologicheskaya nasledstvennost+Vid mehanicheskih
  nagruzok+Legirovannost+Chastota ciklov mehanicheskih
  nagruzok->Mehanizm povrezhdeniya
  (declare (salience 4))
  (Vid materiala  (name "stal")  )
  (Rabocha tehnologicheska sreda  (name "aktivnaya")  )
```

```
 8   (Tehnologicheskaya nasledstvennost  (name "defekty
 9     izgotovleniya ILI povrezhdaemost poverhnosti vsledstvie
10     vozdeystviya agressivnoy sredy")  )
11   (Vid mehanicheskih nagruzok  (name "peremennye")  )
12   (Legirovannost  (name "nizkolegirovanna stal")  )
13   (Chastota ciklov mehanicheskih nagruzok  (name "vysokaya (>
14     60 cikl/min)")  )
15   (Cheredovanie svoystv sredy  (name "da")  )
16 =>
17   (assert
18     (Mehanizm povrezhdeniya  (name "korrozionnaya ustalost")  )
19   )
20 )
```

Таким образом, разработанная продукционная модель содержит: 14 шаблонов фактов, 12 шаблонов правил, 4 начальных (initial) факта и 20 конкретных правил описывающих деградационный процесс коррозионной усталости в нефтехимическом оборудовании. На основе созданной продукционной модели сгенерирован код БЗ в формате CLIPS. Листинг сгенерированного CLP-кода БЗ приводится в Приложении И.

Разработанная БЗ может быть расширена фактами и правилами, отражающими развитие других видов деградационных процессов в нефтехимическом оборудовании, в частности, механической усталости, изнашивания, эрозии и т.д.

Другими успешными примерами практического применения разработанных моделей, методов и программного средства (KBDS) являются:

1. Разработка OWL-онтологий на основе трансформации концептуальных моделей в форме концепт-карт CmapTools (XTM) в рамках модифицированного (дополненного) модельно-ориентированного подхода (MDD/MDA) автоматизированного создания продукционных ЭС [259]. При этом онтологии используются в качестве вычислительно-независимой модели (Computation Independent Model, CIM), в терминологии MDA: был разработан экспериментальный программный компонент преобразования

исходных концепт-карт CmapTools (XTM) в целевые онтологии OWL. При этом использовался разработанный программный интерфейс (API) для взаимодействия с системой PKBD, которая обеспечивает поддержку MDA-подхода и генерацию CRUD-интерфейса для продукционных ЭС.

На Рисунках 50 и 51 представлено описание системы «ЛейнаВерке-Германия» с указанием основных составных частей (подсистем).

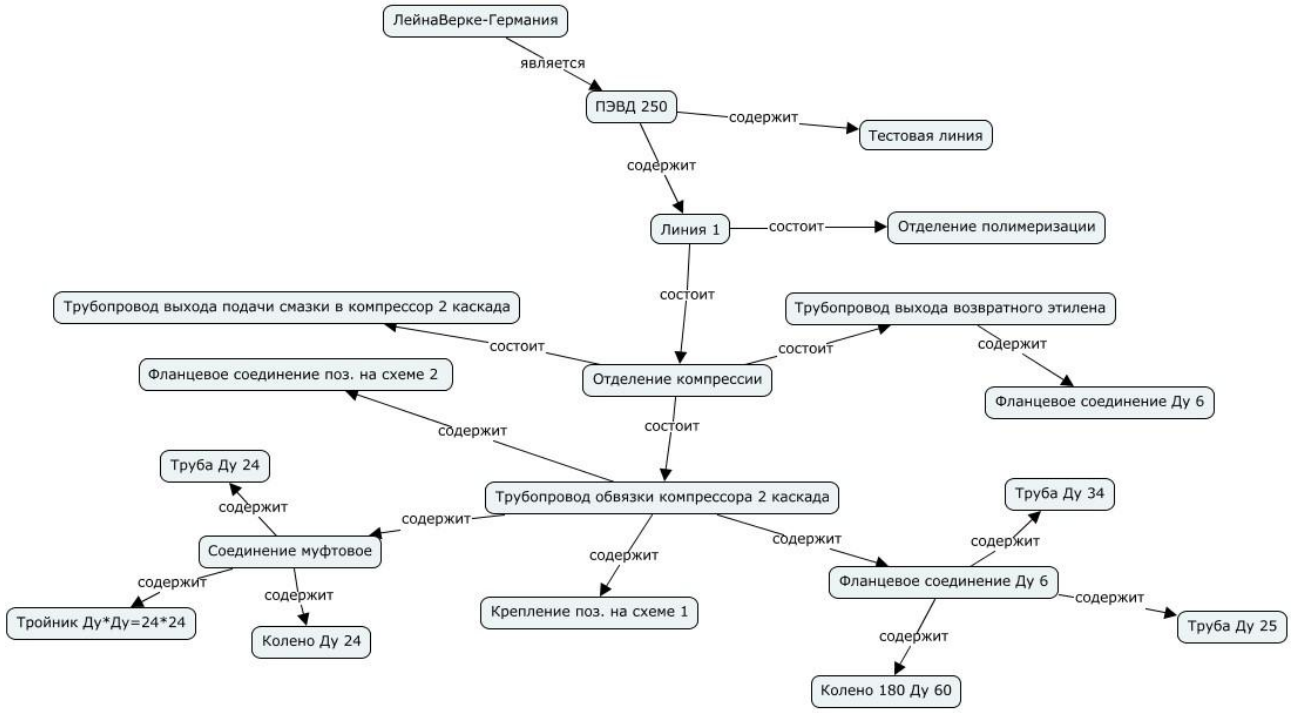

Рисунок 50. Фрагмент концепт-карты CmapTools описания системы «ЛейнаВерке-Германия» (1)

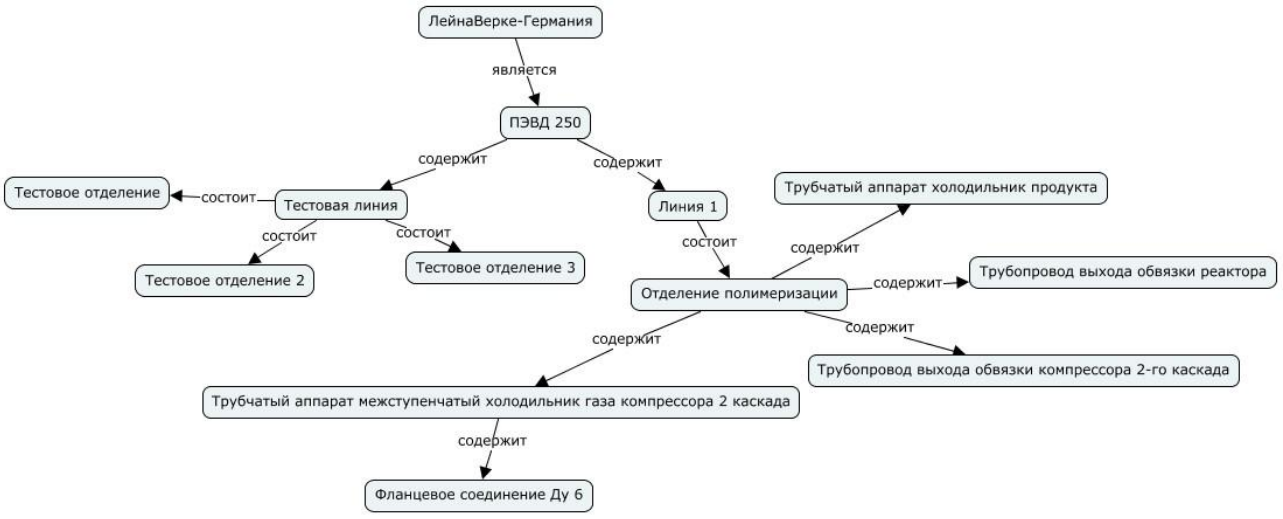

Рисунок 51. Фрагмент концепт-карты CmapTools описания системы «ЛейнаВерке-Германия» (2)

Фрагмент XML-кода полученного файла концепт-карты CmapTools в формате XTM представлен в Листинге 13.

Листинг 13. Фрагмент XML-кода полученного файла концепт-карты CmapTools в формате XTM

```
<topic id="1RRCTCDCK-1LFBB1K-L0">
  <instanceOf>
    <subjectIndicatorRef xlink:type="simple"
      xlink:href="http://cmap.coginst.uwf.edu/#concept"/>
  </instanceOf>
  <baseName>
    <baseNameString><![CDATA[ЛейнаВерке-Германия]]>
    </baseNameString>
  </baseName>
</topic>
<topic id="1RRCTCSBN-13Z58JP-LR">
  <instanceOf>
    <subjectIndicatorRef xlink:type="simple"
      xlink:href="http://cmap.coginst.uwf.edu/#concept"/>
  </instanceOf>
  <baseName>
    <baseNameString><![CDATA[ПЭВД 250]]></baseNameString>
  </baseName>
</topic>
<association id="assoc_1RRCTDKGR-1RNX40Q-N0">
  <instanceOf>
    <topicRef xlink:type="simple" xlink:href="#1RRCTDKGR
      1RNX40Q-N0"/>
  </instanceOf>
  <member>
    <roleSpec>
      <subjectIndicatorRef xlink:type="simple"
        xlink:href="http://cmap.coginst.uwf.edu/#incoming"/>
    </roleSpec>
    <topicRef xlink:type="simple" xlink:href="#1RRCTCDCK
      1LFBB1K-L0"/>
```

```
32    </member>
33    <member>
34      <roleSpec>
35        <subjectIndicatorRef xlink:type="simple"
36          xlink:href="http://cmap.coginst.uwf.edu/#outgoing"/>
37      </roleSpec>
38      <topicRef xlink:type="simple" xlink:href="#1RRCTCSBN
39        13Z58JP-LR"/>
40    </member>
41  </association>
```

На основе извлеченных элементов концепт-карты сгенерирована модель онтологии, которая может быть представлена в виде графа с использованием графического редактора онтологических моделей, входящего в состав KBDS. На Рисунке 52 представлена полученная модель онтологии.

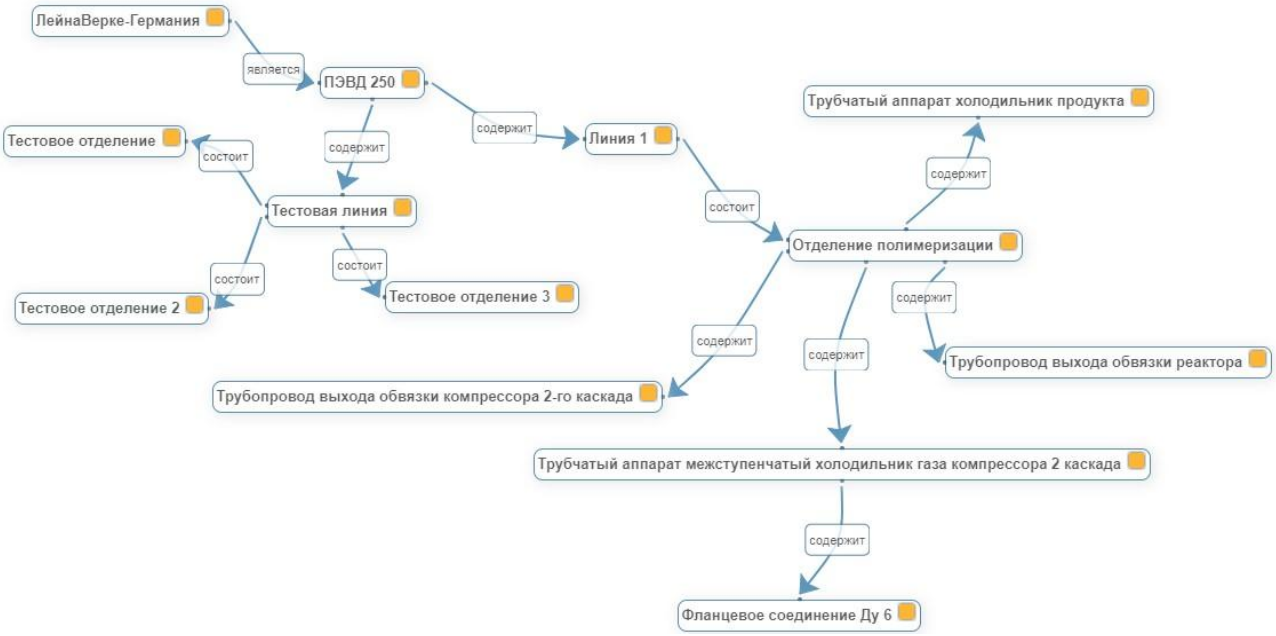

Рисунок 52. Полученная модель онтологии

Таким образом, на основе сформированной модели онтологии получим сгенерированный код БЗ в формате OWL, описывающий таксономию понятий и их отношений. Фрагмент OWL-кода полученной онтологии представлен в Листинге 14.

Листинг 14. Фрагмент OWL-кода полученной онтологии

```
<owl:Ontology rdf:about = 'ЛейнаВерке-Германия'/>
  <owl:Class rdf:ID = 'ЛейнаВеркеГермания'>
    <rdfs:label xml:lang='ru'>ЛейнаВерке-Германия</rdfs:label>
  </owl:Class>
  <owl:Class rdf:ID = 'ПЭВД250'>
    <rdfs:label xml:lang='ru'>ПЭВД 250</rdfs:label>
  </owl:Class>
  <owl:Class rdf:ID = 'Линия1'>
    <rdfs:label xml:lang='ru'>Линия 1</rdfs:label>
    <rdfs:subClassOf rdf:resource='#ПЭВД250' />
  </owl:Class>
  <owl:Class rdf:ID = 'ТестоваяЛиния'>
    <rdfs:label xml:lang='ru'>Тестовая линия</rdfs:label>
    <rdfs:subClassOf rdf:resource='#ПЭВД250' />
  </owl:Class>
  <owl:Class rdf:ID = 'ОтделениеПолимеризации'>
    <rdfs:label xml:lang='ru'>Отделение
      полимеризации</rdfs:label>
  </owl:Class>
  <owl:ObjectProperty rdf:ID='является'>
    <rdfs:domain rdf:resource='#ЛейнаВеркеГермания'/>
    <rdfs:range rdf:resource='#ПЭВД250'/>
  </owl:ObjectProperty>
```

2. Разработка OWL-онтологий на основе трансформации концептуальных моделей в рамках подхода автоматизированного создания императивного описания пространственно-временных сцен для визуализации результатов имитационного моделирования [260, 261].

Также следует отметить работы [262-269], где в той или иной мере обсуждались вопросы формирования БЗ на основе различных концептуальных моделей.

4.5. Оценка эффективности

Для оценки эффективности проведено исследование на базе ИрНИТУ [40, 41]. В исследовании приняло участие 60 студентов Института кибернетики им. Е.И. Попова и изучающих курсы «CASE-средства», «Инструментальные средства информационных технологий» и «Технологии программирования», группы АСУз-10, АСУбз-11, АСУб-12, ЭВМбзс-12. Студенты знакомы с основами проектирования программного обеспечения, моделирования концептуальных моделей (UML и концепт-карт), а также обладают базовыми знаниями в области создания ЭС и БЗ.

Основная задача исследования состояла в том, чтобы оценить трудоемкость разработки БЗ в формате CLIPS с использованием предлагаемого подхода (методики) [15, 21, 22, 24-26, 28, 30, 32, 33, 35, 36] и реализующего его программного средства (прототипа экспериментального программного компонента) [23, 27, 29, 34] (концептуальное моделирование + KBDS, введем обозначение для этого подхода – П1) путем сравнения ее с трудоемкостью разработки БЗ в других условиях:

- с использованием концептуального (визуального) моделирования и других средств разработки БЗ CLIPS (введем обозначение для этого подхода – П2);
- без использования концептуального (визуального) моделирования, но с использованием средства разработки БЗ CLIPS (введем обозначение для этого подхода – П3).

В качестве средства концептуального (визуального) моделирования для построения UML-моделей (диаграмм классов) выбрано CASE-средство – IBM Rational Rose Enterprise [252], для построения концепт-карт (карт знаний) выбран редактор – IHMC CmapTools [253]. В качестве средства создания БЗ CLIPS выбрана среда разработки – ClipsWin [270].

Студентам было предложено разработать статические ЭС для решения тестовых задач диагностирования или прогнозирования в определенной

предметной области (в зависимости от варианта задания, Табл. 2). При этом для обеспечения возможности неоднократного повторения процесса решения задач и их временной компактности выполнения были введены ограничения на объем моделей предметных областей:

- число предметных сущностей: 5-10;
- число свойств предметных сущностей: до 3;
- число связей между сущностями: 5-10;
- число причинно-следственных связей: 3-4;
- число экземпляров причинно-следственных связей (возможных правил): 10-15.

Следует отметить, что первых 4 пункта представляют ограничения, налагаемые на этап концептуального моделирования (ограничения на элементы диаграммы классов UML и концепт-карты). Последний пункт относится к этапу программирования (реализации) кода БЗ в среде разработки (ограничение на количество возможных правил в БЗ).

Таблица 2. Описание решаемых тестовых задач

Вариант	Предметные сущности	Связи	Причинно-следственные связи	Правила
1	6	5	3	10
2	5	6	3	10
3	8	5	3	10
4	5	8	4	11
5	8	7	3	12
6	9	5	3	10
7	5	6	3	14
8	8	7	4	14
9	6	5	3	15
10	7	10	3	12
11	5	6	3	11
12	5	6	3	12
13	7	7	3	14
14	8	5	3	11

15	7	6	3	18
16	6	8	3	14
17	6	5	3	11
18	8	7	3	12
19	7	8	3	10
20	5	7	3	12

Для оценки трудоемкости использовался временной критерий (затраты времени на выполнение отдельных этапов разработки БЗ ЭС). Оценка осуществлялась на следующих этапах [2, 3, 7]:

1. Концептуализация (структурирование) знаний:
    - Формулировка базовых концепции и отношений между ними, включая характеристику различных видов используемых данных, анализ информационных потоков и лежащих в их основе структур в предметной области в терминах, причинно-следственных связей, отношений частное/целое, постоянное/временное и т.п.
    - Построение концептуальной модели.

    Средняя продолжительность стадии: 2-4 недели.
2. Формализация (кодификация) знаний:
    - Перевод ключевых понятий и отношений на некоторый формальный ЯПЗ.
    - Оценка полноты и степени достоверности (неопределенности) информации и других ограничений, накладываемых на логическую интерпретацию данных, таких как зависимость от времени, надежность и полнота различных источников информации.

    Средняя продолжительность стадии: 1-2 месяца.

Основным результатом этапа концептуализации являлась концептуальная модель предметной области, представленная в форме диаграмм классов UML и концепт-карт. Основной результат этапа формализации – синтаксически корректный программный код БЗ, проверенный на адекватность и непротиворечивость.

Результаты оценки временных затрат для вариантов задач с UML-моделированием представлены в Таблице 3 и на Рисунке 53.

Результаты оценки временных затрат для тех же вариантов задач, но с моделированием концепт-карт представлены в Таблице 4 и на Рисунке 54.

При этом выделены минимальные и максимальные процентные значения относительной разницы между П1 и П3, П1 и П2.

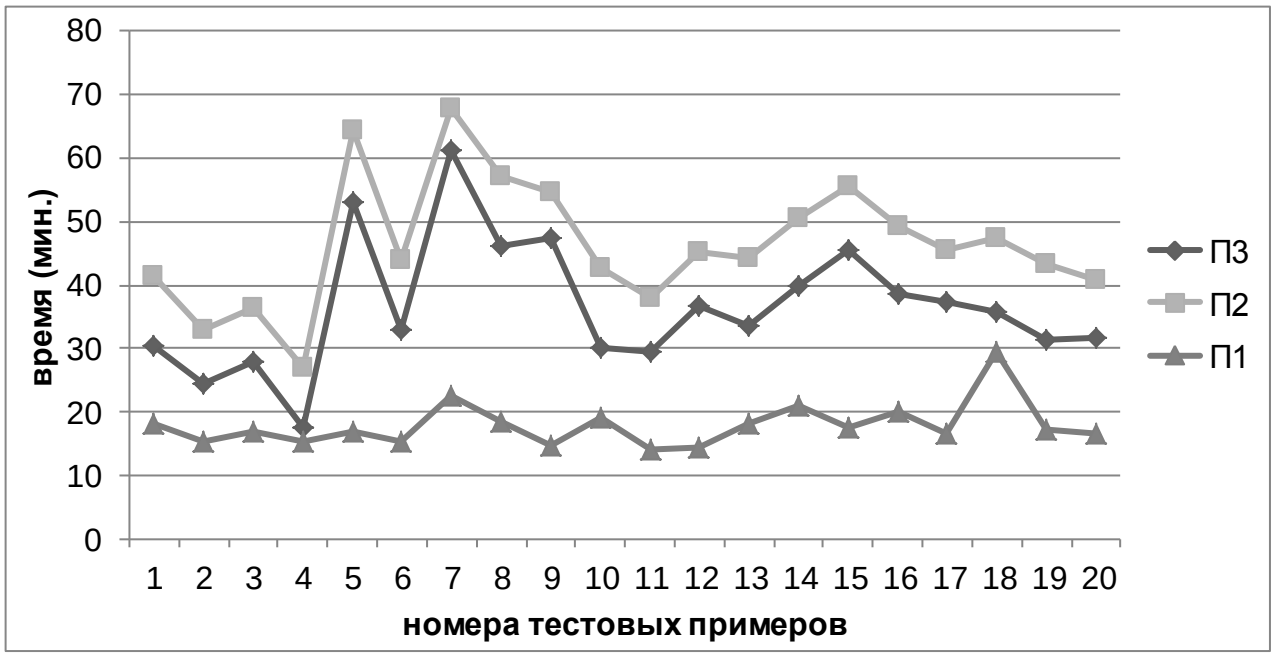

Рисунок 53. Результаты оценки временных затрат (UML-моделирование)

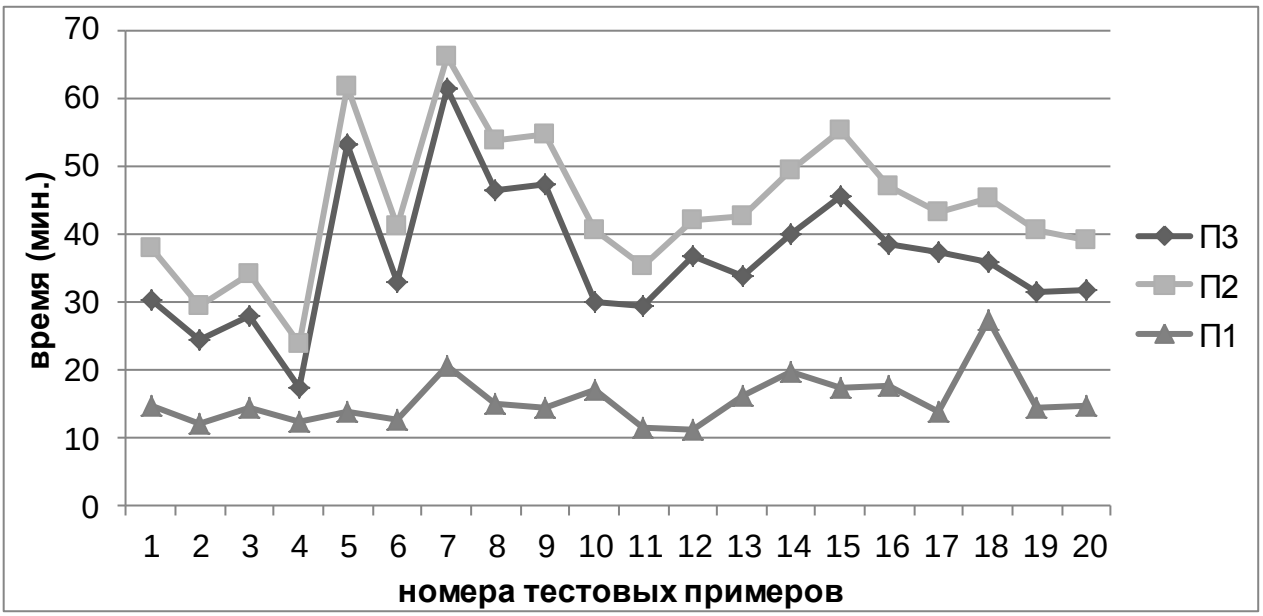

Рисунок 54. Результаты оценки временных затрат (моделирование концепт-карт)

Таблица 3. Результаты оценки временных затрат (UML-моделирование)

Вариант	Rational Rose, мин.	KBDS, мин.	П1, мин.	П2, мин.	П3, мин.	П3: Ошибки програм., шт.	Относительная разница, % П1 и П3	Относительная разница, % П1 и П2
1	10,89	7,2	18,09	41,29	30,4	2	40,49	56,19
2	8,36	7,1	15,46	32,86	24,5	0	36,89	52,95
3	8,58	8,3	16,88	36,46	27,88	1	39,45	53,70
4	9,36	5,83	15,19	26,82	17,46	0	*13,00*	43,36
5	11,25	5,52	16,77	64,41	53,16	3	68,45	**73,96**
6	10,78	4,6	15,38	43,8	33,02	1	53,42	64,89
7	6,6	15,82	22,42	68	61,4	5	63,48	67,03
8	10,95	7,56	18,51	57,23	46,28	3	60,00	67,66
9	7,37	7,2	14,57	54,71	47,34	3	**69,22**	73,37
10	12,58	6,6	19,18	42,7	30,12	0	36,32	55,08
11	8,69	5,5	14,19	38,01	29,32	0	51,60	62,67
12	8,36	6	14,36	45,22	36,86	2	61,04	68,24
13	10,64	7,42	18,06	44,31	33,67	2	46,36	59,24
14	10,66	10,23	20,89	50,57	39,91	1	47,66	58,69
15	10,01	7,56	17,57	55,5	45,49	4	61,38	68,34

16	10,92	8,96	19,88	49,42	38,5	1	48,36	59,77
17	8,14	8,36	16,5	45,59	37,45	1	55,94	63,81
18	11,55	18	29,55	47,43	35,88	3	17,64	*37,70*
19	11,85	5,2	17,05	43,28	31,43	1	45,75	60,61
20	9,12	7,44	16,56	40,95	31,83	2	47,97	59,56
Итоговое среднее значение относительной разницы:							***48,2***	***60,3***

Таблица 4. Результаты оценки временных затрат (моделирование концепт-карт)

Вариант	Cmap Tools, мин.	KBDS, мин.	П1, мин.	П2, мин.	П3, мин.	П3: Ошибки програм., шт.	Относительная разница, % П1 и П3	Относительная разница, % П1 и П2
1	7,6	7,2	14,8	38	30,4	2	51,31	61,05
2	4,95	7,1	12,05	29,45	24,5	0	50,82	59,08
3	6,14	8,3	14,44	34,02	27,88	1	48,21	57,55
4	6,4	5,83	12,23	23,86	17,46	0	29,95	48,74
5	8,42	5,52	13,94	61,58	53,16	3	**73,78**	**77,36**
6	8,12	4,6	12,72	41,14	33,02	1	61,48	69,08
7	4,68	15,82	20,5	66,08	61,4	5	66,61	68,98
8	7,38	7,56	14,94	53,66	46,28	3	67,72	72,16

9	7,28	7,2	14,48	54,62	47,34	3	69,41	73,49
10	10,36	6,6	16,96	40,48	30,12	0	43,69	58,1
11	5,84	5,5	11,34	35,16	29,32	0	61,32	67,75
12	5,07	6	11,07	41,93	36,86	2	69,97	73,6
13	8,87	7,42	16,29	42,54	33,67	2	51,62	61,71
14	9,55	10,23	19,78	49,46	39,91	1	50,44	60,01
15	9,85	7,56	17,41	55,34	45,49	4	61,73	68,54
16	8,62	8,96	17,58	47,12	38,5	1	54,34	62,69
17	5,6	8,36	13,96	43,05	37,45	1	62,72	67,57
18	9,28	18	27,28	45,16	35,88	3	*23,97*	*39,59*
19	9,25	5,2	14,45	40,68	31,43	1	54,02	64,48
20	7,24	7,44	14,68	39,07	31,83	2	53,88	62,43
Итоговое среднее значение относительной разницы:							***55,3***	***63,7***

Отметим особенности выполнения работы на различных этапах:

- П1: При разработке БЗ использовался прототип веб-ориентированной программной системы (KBDS) [23, 27, 29, 31, 34, 37, 38, 40, 41, 244], а также разработанные экспериментальные программные компоненты анализа диаграмм классов UML [23, 27, 29, 34] и концепт-карт XTM [43] (создание данных программных компонентов описано в предыдущих подразделах).
- П2: На данном этапе получены самые большие временные показатели, при том, что использовалось концептуальное моделирование предметной области средствами IBM Rational Rose Enterprise и IHMC CmapTools. Это связанно с тем, что построенные модели вручную переносились в среду разработки ClipsWin, так как у данного средства отсутствует поддержка возможности автоматической кодогенерации БЗ на основе созданных концептуальных моделей. При этом следует отметить, что другие программные средства, позволяющие синтезировать код БЗ в формате CLIPS (например, [183]), не удалось применить для этой задачи.
- П3: Функциональные ограничения ClipsWin в части редактирования водимого программного кода обусловили применение дополнительного текстового редактора (Programmer’s Notepad) при выполнении этапа кодирования. В частности, сначала осуществлялось описание кода БЗ во внешнем текстовом редакторе (используя возможности копирования и вставки отдельных блоков кода), а затем полученный код импортировался в ClipsWin, где осуществлялась проверка синтаксиса. На практике данная схема позволила снизить в 1,5 раза время на создание БЗ.

Анализ эффективности предлагаемого метода по временному критерию показал, что эффективность разработки БЗ методом П1 (с применением UML-моделирования) может быть повышена в среднем на 60.3% по сравнению с П2 (63.7% с моделированием концепт-карт) и на 48.2% по сравнению с П3 (55.3% с моделированием концепт-карт) за счет автоматической кодогенерации на основе визуальных моделей, что, в свою очередь, позволяет:

- Эффективно использовать результаты этапов концептуализации и формализации в форме диаграмм классов UML и концепт-карт XTM, рассматривая последние не как статические графические артефакты, а как основу для формирования программного кода в соответствии с идеологией модельно-управляемого подхода (MDD/MDA).
- Снизить риск ошибок проектирования за счет возможности быстрого прототипирования БЗ и получения их кода.
- Исключить ошибки программирования за счет автоматического отображения элементов концептуальной модели в языковые конструкции ЯПЗ CLIPS или OWL.

4.6. Выводы

- Произведена апробация разработанного подхода на примере создания программных компонентов трансформации концептуальных моделей в форме диаграмм классов UML, концепт-карт CmapTools (XTM) и ДС.
- Разработанные программные компоненты использованы при разработке БЗ ЭС для прогнозирования развития деградационных процессов аппаратов в нефтехимии.
- Осуществлена оценка эффективности на тестовых примерах по временному критерию в сравнении с классическим методом, который не предусматривает синтез программного кода БЗ на основе трансформации концептуальных моделей: эффективность может быть повышена, в среднем на 48.2% с применением UML-моделирования и на 55.3% с применением моделирования концепт-карт.

Заключение

Решение задач, связанных с разработкой новых методов и подходов к созданию интеллектуальных систем и их компонентов, остается перспективной областью научных исследований. В данной работе предлагаются модели и методы автоматизации проектирования и синтеза программных кодов БЗ в форме декларативных программ на основе трансформации концептуальных моделей и их программная реализация в форме инструментального программного средства для повышения эффективности обработки знаний.

Основные результаты, выносимые на защиту:

1. Впервые предложен новый метод автоматизации процесса проектирования и создания программных компонентов интеллектуальных систем, обеспечивающих синтез кода БЗ на основе трансформации концептуальных моделей.
2. Разработан новый предметно-ориентированный декларативный язык описания трансформаций – Transformation Model Representation Language (TMRL).
3. На основе предложенного метода разработано и апробировано инструментальное программное средство – Knowledge Base Development System (KBDS).
4. Создана оригинальная методика автоматизированной разработки БЗ на основе анализа концептуальных моделей, основанная на применении предлагаемых метода и средства, а также произведена оценка ее эффективности.

Список сокращений и условных обозначений

ATL – ATLAS Transformation Language

CASE – Computer Aided Software/System Engineering

CLIPS – C Language Integrated Production System

DSL – Domain Specific Language

EBNF – Extended Backus-Naur Form

EMF – Eclipse Modeling Framework

JESS – Java Expert Systems Shell

KBDS – Knowledge Base Development System

M2C – Model-To-Code

M2M – Model-To-Model

M2T – Model-To-Text

MDA – Model Driven Architecture

MDE – Model Driven Engineering

MDD – Model Driven Development

MOF – Meta-Object Facility

MTL – Model Transformation Language

ODM – Ontology Definition Metamodel

OCL – Object Constraint Language

OMG – Object Management Group

OWL – Web Ontology Language

PKBD – Personal Knowledge Base Designer

QVT – Query / View / Transformation

RDF – Resource Description Framework

RVML – Rule Visual Modeling Language

T2M – Text-To-Model

TMRL – Transformation Model Representation Language

UML – Unified Modeling Language

XMI – XML Metadata Interchange

XML – eXtensible Markup Language

XSD – XML Schema Definition

XSLT – eXtensible Stylesheet Language Transformations

XTM – XML Topic Maps

БЗ – База Знаний

ДО – Дерево Отказов

ДС – Дерево Событий

ООП – Объектно-Ориентированное Программирование

ПСК – Причинно-Следственный Комплекс

СППР – Система Поддержки Принятия Решений

ЭПБ – Экспертиза Промышленной Безопасности

ЭС – Экспертная Система

ЯПЗ – Язык Представления Знаний

Список литературы

1. Кудрявцев Д. В. Системы управления знаниями и применение онтологий: учеб. пособие. СПб.: Изд-во Политехн. ун-та, 2010. 344 с.
2. Гаврилова Т. А., Хорошевский В. Ф. Базы знаний интеллектуальных систем. СПб.: Питер, 2000. 384 с.
3. Гаврилова Т. А., Кудрявцев Д. В., Муромцев Д. И. Инженерия знаний. Модели и методы. СПб.: Лань, 2016. 324 с.
4. Джарратано Дж., Райли Г. Экспертные системы: принципы разработки и программирования, 4-е издание. М. Вильямс, 2007. 1152 с.
5. Люгер Дж. Ф. Искусственный интеллект: стратегии и методы решения сложных проблем, 4-е издание. М.: Вильямс, 2003. 864 с.
6. Рассел С., Норвиг П. Искусственный интеллект: современный подход, 2-е издание. М.: Вильямс, 2006. 1408 с.
7. Джексон П. Введение в экспертные системы. М: Вильямс, 2001. 624 с.
8. Осуга С., Саэки Ю., Судзуки Х., Кобаяси Х., Оцуки С, Китихаси Т., Танака Ю., Арикава С., Синохара Т., Мияхара Т., Харагути М. Приобретение знаний. М.: Мир, 1990. 304 с.
9. Рыбина Г. В. Основы построения интеллектуальных систем. М.: Финансы и статистика; ИНФРА-М, 2010. 432с.
10. Осипов Г. С. Методы искусственного интеллекта. М.: ФИЗМАТЛИТ, 2011. 296 с.
11. Поспелов Д. А. Инженерия знаний // Наука и жизнь. №6. 1987. С. 11–24.
12. Аверкин А. Н., Гаазе-Рапопорт М. Г., Поспелов Д. А. Толковый словарь по искусственному интеллекту. М.: Радио и связь, 1992. 256 с.
13. Юрин А. Ю. Нотация для проектирования баз знаний продукционных экспертных систем // Объектные системы. 2016. № 12. С. 48–54.

14. Бычков И. В., Дородных Н. О., Юрин А. Ю. Подход к разработке программных компонентов для формирования баз знаний на основе концептуальных моделей // Вычислительные технологии. 2016. Т. 21, № 4. С. 16–36.
15. Грищенко М. А., Дородных Н. О., Николайчук О. А., Юрин А. Ю. Применение модельно-управляемого подхода для создания продукционных экспертных систем и баз знаний // Искусственный интеллект и принятие решений. 2016. № 2. С. 16–29.
16. Дородных Н. О., Коршунов С. А., Юрин А. Ю. Концепция подхода к созданию программных компонентов генерации баз знаний на основе трансформации концептуальных моделей // Информационные и математические технологии в науке и управлении. 2016. № 2. С. 111–120.
17. Дородных Н. О., Юрин А. Ю. Разработка программных компонентов для формирования баз знаний на основе трансформации концептуальных моделей // Труды Пятнадцатой национальной конференции по искусственному интеллекту с международным участием (КИИ-2016). 2016. Т. 1. С. 33–40.
18. Дородных Н. О., Юрин А. Ю. Подход к автоматизации создания баз знаний на основе трансформации концептуальных моделей // Материалы VI Международной научно-технической конференции «Открытые семантические технологии проектирования интеллектуальных систем OSTIS-2016». 2016. С. 203–208.
19. Дородных Н. О., Коршунов С. А., Юрин А. Ю. Концепция программной системы создания веб-сервисов синтеза баз знаний на основе концептуальных моделей // Тезисы докладов XVI Всероссийской конференции молодых ученых по математическому моделированию и информационным технологиям. 2015. С. 69.
20. Дородных Н. О. Концепция технологии создания программных компонентов для анализа концептуальных моделей и синтеза баз знаний // Тезисы конференции «Ляпуновские чтения – 2015». 2015. С. 27.

21. Дородных Н. О., Юрин А. Ю. Формирование баз знаний продукционного типа на основе UML-моделей // Информатика и кибернетика. 2016. Т. 5, № 3. С. 44–50.
22. Дородных Н. О., Юрин А. Ю. Использование диаграмм классов UML для формирования продукционных баз знаний // Программная инженерия. 2015. № 4. С. 3–9.
23. Дородных Н. О., Юрин А. Ю. Web-сервис для автоматизированного формирования продукционных баз знаний на основе концептуальных моделей // Программные продукты и системы. 2014. № 4. С. 103–107.
24. Грищенко М. А., Дородных Н. О., Юрин А. Ю. Модельно-управляемый подход. Алгоритмическое и программное обеспечение для создания продукционных баз знаний и экспертных систем. LAP. 2015. 129 с.
25. Дородных Н. О., Николайчук О. А., Юрин А. Ю. Автоматизация создания продукционных баз знаний на основе концептуальных моделей // Труды Шестой международной конференции «Системный анализ и информационные технологии». 2015. Т. 1. С. 281–288.
26. Дородных Н. О., Коршунов С. А., Юрин А. Ю. Синтез баз знаний на основе концептуальных моделей // Сборник трудов XLIV Международной конференции и XIV Международной конференции молодых ученых «Информационные технологии в науке, образовании и управлении. IT + S&E`15». 2015. С. 214–221.
27. Дородных Н. О., Юрин А. Ю. Концепция облачного сервиса для поддержки процесса создания продукционных баз знаний // Тезисы докладов III Российско-монгольской конференции молодых ученых по математическому моделированию, вычислительно-информационным технологиям и управлению. 2015. С. 40.
28. Дородных Н. О., Коршунов С. А., Юрин А. Ю. Использование концептуальных моделей при автоматизированном формировании продукционных баз знаний // Материалы Международной научно-практической конференции «Фундаментальная информатика,

информационные технологии и системы управления: реалии и перспективы. FIITM-2014». 2014. С. 129–138.

29. Дородных Н. О., Юрин А. Ю. Концепция Веб-сервиса для автоматизированного формирования продукционных баз знаний // Тезисы докладов II Российско-монгольской конференции молодых ученых по математическому моделированию, вычислительно-информационным технологиям и управлению. 2013. С. 30.
30. Дородных Н. О., Юрин А. Ю. Автоматизированное формирование продукционных баз знаний на основе концептуальных моделей // Материалы XVIII Международного молодежного форума «Радиоэлектроника и молодежь в XXI веке». 2014. С. 67–68.
31. Дородных Н. О. Прототип программной системы автоматизированного создания баз знаний // Тезисы конференции «Ляпуновские чтения – 2016». 2016. С. 31.
32. Грищенко М. А., Дородных Н. О., Юрин А. Ю. Прототипирование продукционных баз знаний на основе концептуальных моделей // Материалы XVII Всероссийской конференции молодых ученых по математическому моделированию и информационным технологиям. 2016. С. 86.
33. Дородных Н. О. Автоматизированное создание баз знаний продукционного типа на основе диаграмм классов UML // Материалы III Всероссийской студенческой научно-практической конференции «Коммуникационные технологии: социально-экономические и информационные аспекты». 2015. С. 16–20.
34. Дородных Н. О., Коршунов С. А., Юрин А. Ю. Web-сервис для автоматизированного формирования продукционных баз знаний на основе концептуальных моделей // Тезисы докладов XV Всероссийской конференции молодых ученых по математическому моделированию и информационным технологиям. 2014. С. 62–63.

35. Дородных Н. О. Использование диаграмм классов UML и OWL онтологий при формировании продукционных баз знаний // Тезисы конференции «Ляпуновские чтения – 2014». 2014. С. 28.
36. Дородных Н. О., Юрин А. Ю. Формирование продукционных баз знаний CLIPS на основе концептуальных моделей // Тезисы докладов IV Всероссийской конференции «Математическое моделирование и вычислительно-информационные технологии в междисциплинарных научных исследованиях». 2014. С. 26.
37. Дородных Н. О. Web-ориентированный редактор моделей трансформаций (Web Transformation Model Editor). 2017. Свидетельство о государственной регистрации программ для ЭВМ № 2017618430 М.: Федеральная служба по интеллектуальной собственности, патентам и товарным знакам.
38. Дородных Н. О. RVML editor (Web Knowledge Base Designer). 2017. Свидетельство о государственной регистрации программ для ЭВМ № 2017618446 М.: Федеральная служба по интеллектуальной собственности, патентам и товарным знакам.
39. Berman A. F., Grishchenko M. A., Dorodnykh N. O., Nikolaychuk O. A., Yurin A. Y. A model-driven approach and a tool to support creation of rule-based expert systems for industrial safety expertise // Proceedings of the 12th International Forum on Knowledge Asset Dynamics (IFKAD-2017) – Russia, St. Petersburg: Graduate School of Management of St. Petersburg University. 2017. P. 2034–2050.
40. Дородных Н. О. Web-based software for automating development of knowledge bases on the basis of transformation of conceptual models // Материалы VII Международной научно-технической конференции «Открытые семантические технологии проектирования интеллектуальных систем OSTIS-2017». 2017. С. 145–150.
41. Дородных Н. О. Программная система автоматизации разработки web-сервисов для генерации баз знаний // Труды Седьмой международной

конференции «Системный анализ и информационные технологии». 2017. С. 222–229.

42. Дородных Н. О., Николайчук О. А., Юрин А. Ю. Автоматизированное создание продукционных баз знаний на основе деревьев событий // Информационные и математические технологии в науке и управлении. 2017. Т. 6, № 2. С. 30–41.
43. Дородных Н. О., Юрин А. Ю. Автоматизированное создание продукционных баз знаний на основе коцепт-карт CmapTools // Труды Седьмой международной конференции «Системный анализ и информационные технологии». 2017. С. 337–341.
44. Zulkafli Z., Perez K., Vitolo C., Buytaert W., Karpouzoglou T., Dewulf A., Bievre B. D., Clark J., Hannah D.M., Shaheed S. User-driven design of decision support systems for polycentric environmental resources management // Environmental Modelling & Software. 2017. Vol. 88. P. 58–73.
45. Gavrilova T. A., Leshcheva I. A. Ontology design and individual cognitive peculiarities: A pilot study // Expert Systems with Applications. 2015. Vol. 42. P. 3883–3892.
46. Starr R. R., Parente de Oliveira J. M. Concept maps as the first step in an ontology construction method // Information Systems. 2013. Vol. 38. P. 771–783.
47. Herrero-Zazo M., Segura-Bedmar I., Martínez P. Conceptual models of drug-drug interactions: A summary of recent efforts // Knowledge-Based Systems. 2016. Vol. 114. P. 99–107.
48. Wagner W. P. Trends in expert system development: A longitudinal content analysis of over thirty years of expert system case studies // Expert Systems with Applications. 2017. Vol. 76. P. 85–96.
49. Schreiber G., Akkermans H., Anjewierden A., Hoog de R., Shadbolt N. R., Velde W. V., Wielinga B. Knowledge Engineering and Management. The CommonKADS methodology // The MIT Press, Cambridge, MA. 2000.
50. Studer R., Benjamins V. R., Fensel D. Knowledge engineering: principles and methods // Data & Knowledge Engineering. 1998. Vol. 25, no. 1-2. P. 161–197.

51. Stokes M. Managing engineering knowledge: MOKA: methodology for knowledge based engineering applications (6th ed.). New York: ASME Press. 2001.
52. Protégé. 2017. URL: http://protege.stanford.edu/ (дата обращения: 2017-04-17).
53. JessTab. 2006. URL: https://protegewiki.stanford.edu/wiki/JessTab (дата обращения: 2017-04-17).
54. CLIPS: A Tool for Building Expert Systems. 2017. URL: http://clipsrules.sourceforge.net/ (дата обращения: 2017-04-14).
55. JESS: The Rule Engine for the Java Platform. 2013. URL: http://www.jessrules.com/ (дата обращения: 2017-04-14).
56. Exsys Corvid: Expert System Development Tool. 2011. URL: http://www.exsys.com/exsyscorvid.html (дата обращения: 2017-04-14).
57. Гаврилова Т. А., Гулякина Н. А. Визуальные методы работы со знаниями: попытка обзора // Искусственный интеллект и принятие решений. 2008. № 1. С. 15–21.
58. Doherty P., Lukaszewicz W., Szalas A. CAKE: A computer-aided knowledge engineering technique // Proceedings of the 15th European Conference on Artificial Intelligence (ECAI’2002). Amsterdam, IOS Press. 2002.
59. Nalepa G. J., Ligęza A. HeKatE methodology, hybrid engineering of intelligent systems // International Journal of Applied Mathematics and Computer Science. 2010. Vol. 20, no. 1. P. 35–53.
60. Nalepa G. J., Kluza K. UML representation for rule-based application models with XTT2-based business rules // International Journal of Software Engineering and Knowledge Engineering. 2012. Vol. 22, no. 4. P. 485–524.
61. Загорулько Г. Б., Загорулько Ю. А. Подход к организации комплексной поддержки процесса разработки интеллектуальных СППР в слабоформализованных предметных областях // Материалы VI международной научно-технической конференции «Открытые семантические технологии проектирования интеллектуальных систем OSTIS-2016». 2016. С. 61–64.

62. Колесников А. В., Кириков И. В., Листопад С. В. Гибридные интеллектуальные системы с самоорганизацией: координация, согласованность, спор. М.: ИПИ РАН, 2014. 189 с.
63. Колесников А. В. Гибридные интеллектуальные системы. Теория и технология разработки. СПб.: Изд-во СПбГТУ, 2001. 711 с.
64. Sahin S., Tolun M. R., Hassanpour R. Hybrid expert systems: A survey of current approaches and applications // Expert Systems with Applications. 2012. Vol. 39, no. 4. P. 4609–4617.
65. Рыбина Г. В. Инструментальные средства построения динамических интегрированных экспертных систем: развитие комплекса ат-технология // Искусственный интеллект и принятие решений. 2010. № 1. С. 41–48.
66. Аликин С. С., Жидаков В. П. Разработка платформы создания экспертных систем с применением метапрограммирования // Фундаментальные проблемы радиоэлектронного приборостроения. 2012. Т. 12, № 6. С. 80–83.
67. Ермаков А. Е., Найденова К. А. Инструментальное средство для автоматизированного создания экспертных систем // Программные продукты и системы. 2013. № 3. С. 107–114.
68. Ruiz-Mezcua B., Garcia-Crespo A., Lopez-Cuadrado J. L., Gonzalez-Carrasco I. An expert system development tool for non AI experts // Expert Systems with Applications. 2011. Vol. 38, no. 1. P. 597–609.
69. Gascueña J. M., Navarro E., Fernández-Caballero A., Martínez-Tomás R. Model-to-model and model-to-text: looking for the automation of VigilAgent // Expert Systems. 2014. Vol. 31, no. 3. P. 199–212.
70. Touzi A., Messaoud M. B. New Approach for Conception and Implementation of Object Oriented Expert System Using UML // The International Arab Journal of Information Technology. 2009. Vol. 6, no. 1. P. 99–106.
71. Abdullah M. S., Paige R., Kimble C., Benest I. A UML profile for knowledge-based systems modelling // Processing of the 5th ACIS International Conference on Software Engineering Research, Management & Applications. 2007. P. 871–878.

72. Kadhim M. A., Alam M. A., Kaur H. Design and implementation of Intelligent Agent and Diagnosis Domain Tool for Rule-based Expert System // Processing of the International Conference on Machine Intelligence Research and Advancement. 2013. P. 619–622.
73. Hatzilygeroudis I., Kovas K. A Tool for Automatic Creation of Rule-Based Expert Systems with CFs // Processing of the International Conference on Artificial Intelligence Applications and Innovations (AIAI) / Advances in Information and Communication Technology. 2010. Vol. 339. P. 195–202.
74. Baumeister J., Striffler A. Knowledge-driven systems for episodic decision support // Knowledge-Based Systems. 2015. Vol. 88. P. 45–56.
75. Гаврилова Т. А. Онтологический подход к управлению знаниями при разработке корпоративных информационных систем // Новости искусственного интеллекта. 2003. № 2. С. 24–29.
76. Дубинин В. Н., Вяткин В. В. Проектирование управляющих приложений на основе трансформации онтологий с использованием языков логического программирования // Труды Международной научно-технической конференции «Современные информационные технологии». 2012. № 16. С. 6–25.
77. Стенников В. А., Барахтенко Е. А., Соколов Д. В. Применение онтологий при реализации концепции модельно-управляемой разработки программного обеспечения для проектирования теплоснабжающих систем // Онтология проектирования. 2014. Т. 14, № 4. С. 54–68.
78. Ворожцова Т. Н., Скрипкин С. К. Онтологический подход к моделированию программного комплекса // Вестник Иркутского государственного технического университета. 2006. Т. 26, № 2. С. 72–78.
79. Голенков В. В., Гулякина Н. А. Принципы построения массовой семантической технологии компонентного проектирования интеллектуальных систем // Материалы международной научно-технической конференции «Открытые семантические технологии проектирования интеллектуальных систем OSTIS-2011». 2011. С. 21–58.

80. Грибова В. В., Клещев А. С., Крылов Д. А., Москаленко Ф. М., Смагин С. В., Тимченко В. А., Тютюнник М. Б., Шалфеева Е. А. Проект IACPaaS. Комплекс для интеллектуальных систем на основе облачных вычислений // Искусственный интеллект и принятие решений. 2011. № 1. С. 27–35.
81. Загорулько Ю. А. Семантическая технология разработки интеллектуальных систем, ориентированная на экспертов предметной области // Онтология проектирования. 2015. Т. 15, № 1. С. 30–46.
82. Массель Л. В., Копайгородский А. Н., Аршинский В. Л. Построение интеллектуальных систем для исследований энергетики на основе алгебраических сетей и онтологий: подход и реализация // Вычислительные технологии. 2008. Т. 13, спец. выпуск 1. С. 50–58.
83. Nofal M., Fouad K. M. Developing Web-Based Semantic Expert Systems // International Journal of Computer Science. 2014. Vol. 11, no. 1. P. 103–110.
84. Shue L., Chen C., Shiue W. The development of an ontology-based expert system for corporate financial rating // Expert Systems with Applications. 2009. Vol. 36, no. 2. P. 2130–2142.
85. Corsar D., Sleeman D. H. Developing Knowledge-Based Systems using the Semantic Web // Proceedings of the 2008 International Conference on Visions of Computer Science: BCS International Academic Conference. 2008. P. 29–40.
86. O'Connor M. J., Shankar R. D., Nyulas C., Tu S. W., Das A. Developing a Web-Based Application using OWL and SWRL // Proceedings of the AI Meets Business Rules and Process Management, AAAI Spring Symposium. 2008.
87. Чернецки К., Айзенекер У. Порождающее программирование: методы, инструменты, применение. СПб: Питер, 2005. 736 с.
88. Czarnecki K. Overview of generative software development // Unconventional Programming Paradigms. 2005. P. 326–341.
89. Cretu L. G., Florin D. Model-Driven Engineering of Information Systems: Principles, Techniques, and Practice. Apple Academic Press. 2014.
90. Brambilla M., Cabot J., Wimmer M. Model Driven Software Engineering in Practice. Morgan & Claypool Publishers. 2012.

91. Sami B., Book M., Gruhn V. Model-Driven Software Development. Springer. 2005.
92. Schmidt D. C. Model-Driven Engineering // IEEE Computer. 2006. Vol. 39, no. 2. P. 25–31.
93. Volter M., Stahl T., Bettin J., Haase A., Helsen S. Czarnecki K. Model-Driven Software Development: Technology, Engineering, Management. John Wiley & Sons. 2006.
94. Gašević D., Djurić D., Devedžić V. Model driven engineering and ontology development (2nd ed.). New York: Springer-Verlag. 2009.
95. Djurić D., Gašević D., Devedžić V. The Tao of Modeling Spaces // Journal of Object Technology. 2006. Vol. 5, no. 8. P. 125–147.
96. da Silva A. R. Model-driven engineering: A survey supported by the unified conceptual model // Computer Languages, Systems & Structures. 2015. Vol. 43. P. 139–155.
97. Kleppe A., Warmer J., Bast W. MDA Explained: The Model-Driven Architecture: Practice and Promise (1st ed.). Addison-Wesley Professional. 2003.
98. Frankel D. Model Driven Architecture: Applying MDA to Enterprise Computing. New York: Wiley. 2003.
99. Miguel M., Jourdan J., Salicki S. Practical Experiences in the Application of MDA // Lecture Notes in Computer Science. 2002. Vol. 2460. P. 128–139.
100. Truyen F. The Fast Guide to Model Driven Architecture: The Basics of Model Driven Architecture. Cephas Consulting Corp. 2006.
101. MDA Specifications. 2017. URL: http://www.omg.org/mda/specs.htm (дата обращения: 2017-04-14).
102. Canadas J., Palma J., Tunez S. InSCo-Gen: A MDD Tool for Web Rule-Based Applications // Lecture Notes in Computer Science. 2009. Vol. 5648. P. 523–526.
103. Chaur G. W. Modeling Rule-Based Systems with EMF. Eclipse Corner Articles. 2004. URL: http://www.eclipse.org/articles/Article-Rule%20Modeling%20With%20EMF/article.html (дата обращения: 2017-04-14).

104. Дородных Н. О., Юрин А. Ю. About the specialization of model-driven approach for creation of case-based intelligent decision support systems // Материалы VII Международной научно-технической конференции «Открытые семантические технологии проектирования интеллектуальных систем OSTIS-2017». 2017. С. 151–154.
105. Грищенко М. А., Юрин А. Ю., Павлов А. И. Разработка экспертных систем на основе трансформации информационных моделей предметной области // Программные продукты и системы. 2013. № 3. С. 143–147.
106. Djurić D., Gašević D., Devedžić V. Ontology modeling and MDA // Journal of Object technology. 2005. Vol. 4, no. 1. P. 109–128.
107. Staab S., Walter T., Gröner G., Parreiras F.S. Model Driven Engineering with Ontology Technologies // Lecture Notes in Computer Science. 2010. Vol. 6325. P. 62–98.
108. Грибачев К. Г. Delphi и Model Driven Architecture. Разработка приложений баз данных. СПб: Питер, 2004. 352 с.
109. Cherkashin E. A., Paramonov V. V., Fedorov R. K., Terehin I. N., Pozdnyak E. I., Annenkov D. V. Information Systems Framework Synthesis on the Base of a Logical Approach // E-society Journal: research and applications. 2012. Vol. 3, no. 2. P. 1–13.
110. Distante D., Pedone P., Rossi G., Canfora G. Model-Driven Development of Web Applications with UWA, MVC and JavaServer Faces // Lecture Notes in Computer Science. 2007. Vol. 4607. P. 457–472.
111. Ribarić M., Gašević D., Milanović M., Giurca A., Lukichev S., Wagner G. Model-Driven Engineering of Rules for Web Services // Lecture Notes in Computer Science. 2007. Vol. 5235. P. 377–395.
112. Dunstan N. Generating domain-specific web-based expert systems // Expert Systems with Applications. 2008. Vol. 35, no. 3. P. 686–690.
113. Ершов А. П. Предварительные соображения о лексиконе программирования // Кибернетика и вычислительная техника. 1985. № 1.

114. Касьянов В. Н., Поттосин И. В. Методы построения трансляторов. Н.: Наука, 1986. 344 с.
115. Смешанные вычисления и преобразование программ // Сб. научных трудов под ред. Котова В. Е. Новосибирск: ВЦ СО АН СССР, 1991. – 259 с.
116. Касьянов В. Н., Сабельфельд В. К. Инструментальные средства преобразования программ. Препр. ВЦ СО АН СССР №765. Новосибирск, 1987.
117. Partsch H., Steinbruggen R. Program Transformation Systems // ACM Computing Surveys. 1983. Vol. 15, no. 3. P. 199–236.
118. Ершов А. П. Научные основы доказательного программирования: Научн. сообщ. // Вестник АН СССР. 1984. № 10. С. 9–19.
119. Czarnecki K., Helsen S. Feature-based survey of model transformation approaches // IBM Systems Journal. 2006. Vol. 45, no. 3. P. 621–645.
120. Kelly S., Tolvanen, J. -P. Domain-Specific Modeling: Enabling Full Code Generation. Wiley-IEEE Computer Society Pr. 2008.
121. Tolvanen J. -P. Kelly S. Model-Driven Development Challenges and Solutions – Experiences with Domain-Specific Modelling in Industry // Processing of the 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2016). 2016. P. 711–719.
122. Kang K. C., Cohen S. G., Hess J. A., Novak W. E., Peterson A. S. Feature-Oriented Domain Analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21 ESD-90-TR-222. 1990.
123. Meta Object Facility (MOF) Core // OMG Formally Released Versions of MOF. 2016. URL: http://www.omg.org/spec/MOF/ (дата обращения: 2017-04-14).
124. Ecore. 2017. URL: http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html (дата обращения: 2017-04-14).
125. Jouault F., Bézivin J. KM3: a DSL for Metamodel Specification // Processing of the International Conference on Formal Methods for Open Object-Based

Distributed Systems / Lecture Notes in Computer Science. 2006. Vol. 4037. P. 171–185.

126. Sendall S., Kozaczynski W. Model Transformation: The Heart and Soul of Model-Driven Software Development // IEEE Software. 2003. Vol. 20, no. 5. P. 42–45.
127. Mens T., Gorp P. V. A Taxonomy of Model Transformations // Electronic Notes in Theoretical Computer Science. 2006. Vol. 152. P. 125–142.
128. Visser E. A Survey of Rewriting Strategies in Program Transformation Systems // Electronic Notes in Theoretical Computer Science. 2001. Vol. 57. P. 109–143.
129. Rozenberg G. Handbook of Graph Grammars and Computing by Graph Transformations. World Scientific Publishing Company. 1997.
130. Query/View/Transformation (QVT) Version 1.3 // OMG Document formal/2016-06-03. 2016. URL: http://www.omg.org/spec/QVT/1.3/ (дата обращения: 2017-04-17).
131. Jouault F., Allilaire F., Bézivin J., Kurtev I. ATL: A model transformation tool // Science of Computer Programming. 2008. Vol. 72, no. 1. P. 31–39.
132. Object Constraint Language (OCL) Version 2.4 // OMG Document formal/2014-02-03. 2014. URL: http://www.omg.org/spec/OCL/2.4/ (дата обращения: 2017-04-17).
133. Varro D., Balogh A. The model transformation language of the VIATRA2 framework // Science of Computer Programming. 2007. Vol. 63, no. 3. P. 214–234.
134. Balasubramanian D., Narayanan A., Buskirk C., Karsai G. The graph rewriting and transformation language: GReAT // Electronic Communications of the EASST. 2006. Vol. 1. P. 1–8.
135. Arendt T., Biermann E., Jurack S., Krause C., Taentzer G. Henshin: advanced concepts and tools for in-place EMF model transformations // Processing of the 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2010) / Lecture Notes in Computer Science, Springer Berlin Heidelberg. 2010. Vol. 6394. P. 121–135.

136. Eclipse Modeling Framework (EMF). 2017. URL: http://www.eclipse.org/modeling/emf/ (дата обращения: 2017-04-17).
137. Epsilon. 2017. URL: http://www.eclipse.org/epsilon/ (дата обращения: 2017-04-17).
138. XSL Transformations (XSLT) Version 2.0. 2007. URL: http://www.w3.org/TR/xslt20 (дата обращения: 2017-04-17).
139. Eclipse. 2017. URL: https://eclipse.org/ (дата обращения: 2017-04-17).
140. Kiko K., Atkinson C. A Detailed Comparison of UML and OWL. Germany: University of Mannheim. 2008.
141. Unified Modeling Language (UML) Version 2.5 // OMG Document formal/15-03-01. 2015. URL: http://www.omg.org/spec/UML/2.5/ (дата обращения: 2017-04-17).
142. OWL 2 Web Ontology Language Document Overview (Second Edition). 2012. URL: https://www.w3.org/TR/owl2-overview (дата обращения: 2017-04-17).
143. Milanović M., Gašević D., Giurca A., Wagner G., Devedžić V. On interchanging between OWL/SWRL and UML/OCL // Proceedings of the 6th Workshop on OCL for (Meta-) Models in Multiple Application Domains (OCLApps) at the 9th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS). 2006. P. 81–95.
144. Milanović M., Gašević D., Giurca A., Wagner G., Devedžić V. Bridging concrete and abstract syntaxes in model-driven engineering: a case of rule languages // Software: Practice and Experience. 2009. Vol. 39, no. 16. P. 1313–1346.
145. Zedlitz J., Luttenberger N. Conceptual Modelling in UML and OWL-2 // International Journal on Advances in Software. 2014. Vol. 7, no. 1. P. 182–196.
146. Zedlitz J., Jörke J., Luttenberger N. From UML to OWL 2 // Knowledge Technology. Communications in Computer and Information Science. 2012. Vol. 295. P. 154–163.
147. Belghiat A., Bourahla M. An Approach based AToM3 for the Generation of OWL Ontologies from UML Diagrams // International Journal of Computer Applications. 2012. Vol. 41, no. 3. P. 41–48.

148. Parreiras F. S., Staab S. Using ontologies with UML class-based modeling: The TwoUse approach // Data & Knowledge Engineering. 2010. Vol. 69, no. 11. P. 1194–1207.
149. Brockmans S., Colomb R. M., Haase P., Kendall E. F., Wallace E. K., Welty C., Xie G. T. A Model-Driven Approach for Building OWL DL and OWL Full Ontologies // Proceedings of the International Semantic Web Conference (ISWC 2006) / Lecture Notes in Computer Science. 2006. Vol. 4273. P. 187–200.
150. Gašević D., Djurić D., Devedžić V., Damjanović V. Converting UML to OWL ontologies // Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters. 2004. P. 488–489.
151. Na, H. -S., Choi, O -H., Lim, J. -E. A method for building domain ontologies based on the transformation of UML models // Proceedings of 4th International Conference on Software Engineering Research, Management and Applications (SERA 2006). 2006. P. 332–338.
152. Гуськов Г. Ю., Наместников А. М. Ontological Mapping for Conceptual Models of Software System // Материалы VII Международной научно-технической конференции «Открытые семантические технологии проектирования интеллектуальных систем OSTIS-2017». 2017. С. 111–116.
153. Наместников А. М., Гуськов Г. Ю. Программная система преобразования UML-диаграмм в онтологии на языке OWL // Труды Пятнадцатой национальной конференции по искусственному интеллекту с международным участием (КИИ-2016). 2016. Т. 3. С. 270–278.
154. Черняховская Л. Р., Малахова А. И. Формирование правил принятия решений в управлении проектами по результатам онтологического анализа // Материалы XV Международной научной конференции «Проблемы управления и моделирования в сложных системах». 2013. С. 343–350.
155. Xu Z., Ni Y., He W., Lin L., Yan Q. Automatic extraction of OWL ontologies from UML class diagrams: a semantics-preserving approach // World Wide Web. 2012. Vol. 15, no. 5. P. 517–545.

156. Felfernig A., Friedrich G. E., Jannach D. UML as domain specific language for the construction of knowledge-based configuration systems // International Journal of Software Engineering and Knowledge Engineering. 2000. Vol. 10, no. 4. P. 449–469.
157. Felfernig A., Friedrich G., Jannach D., Zanker M. Configuration knowledge representation using UML/OCL // Proceedings of the International Conference on the Unified Modeling Language. 2002. P. 49–62.
158. Mehrolhassani M., ELÇİ A. Developing Ontology Based Applications of Semantic Web Using UML to OWL Conversion // Proceedings of the Open Knowlege Society. A Computer Science and Information Systems Manifesto (WSKS 2008) / Communications in Computer and Information Science. 2008. Vol. 19. P. 566–577.
159. Cranefield S., Pan J. Bridging the gap between the model-driven architecture and ontology engineering // International Journal of Human-Computer Studies. 2007. Vol. 65, no. 7. P. 595–609.
160. Reynares E., Caliusco M. L., Galli M. R. A set of ontology design patterns for reengineering SBVR statements into OWL/SWRL ontologies // Expert Systems with Applications. 2015. Vol. 42, no. 5. P. 2680–2690.
161. Alberts R., Franconi E. An Integrated Method Using Conceptual Modelling to Generate an Ontology-based Query Mechanism // Proceedings of the OWL: Experiences and Directions Workshop 2012 (OWLED). 2012. Vol. 849.
162. van Assem M., Menken M. R., Schreiber G., Wielemaker J., Wielinga B. A Method for Converting Thesauri to RDF/OWL // Proceedings of the International Semantic Web Conference (ISWC 2004) / Lecture Notes in Computer Science. 2004. Vol. 3298. P. 17–31.
163. Myroshnichenko I., Murphy M. C. Mapping ER Schemas to OWL Ontologies // Proceedings of the 2009 IEEE International Conference on Semantic Computing. 2009. P. 324–329.
164. Слободюк А. А., Маторин С. И., Четвериков С. Н. О подходе к созданию онтологий на основе системно-объектных моделей предметной области //

Научные ведомости БелГУ. Сер. История. Политология. Экономика. Информатика. 2013. Т. 165, № 22. С. 186–194.

165. Simón A., Ceccaroni L., Rosete A. Generation of OWL Ontologies from Concept Maps in Shallow Domains // Proceedings of the Current Topics in Artificial Intelligence (CAEPIA 2007) / Lecture Notes in Computer Science. 2007. Vol. 4788. P. 259–267.
166. Brilhante V., Macedo G. T., Macedo S. F. Heuristic transformation of well-constructed conceptual maps into OWL preliminary domain ontologies // Proceedings of the Second Workshop on Ontologies and their Applications, CEUR-WS. 2006.
167. Extensible Markup Language (XML). 2016. URL: https://www.w3.org/XML/ (дата обращения: 2017-04-17).
168. Bohring H., Auer S. Mapping XML to OWL Ontologies // In: Jantke K, Fähnrich K, Wittig W. Marktplatz Internet: Von e-Learning bis e-Payment: 13. Leipziger Informatik-Tage (LIT2005). 2005. P. 147–156.
169. Rodrigues T., Rosa P., Cardoso J. Mapping XML to exiting OWL ontologies // Proceedings of the International Conference WWW/Internet. 2006. P. 72–77.
170. Rodrigues T., Rosa P., Cardoso J. Moving from syntactic to semantic organizations using JXML2OWL // Computers in Industry. 2008. Vol. 59, no. 8. P. 808–819.
171. O'Connor M. J., Das A. K. Acquiring OWL Ontologies from XML Documents // Proceedings of the 6th International Conference on Knowledge Capture (K-CAP'11). 2011. P. 17–24.
172. Bedini I., Matheus C., Patel-Schneider P. F., Boran A., Nguyen B. Transforming XML Schema to OWL Using Patterns // Proceedings of the 2011 IEEE Fifth International Conference on Semantic Computing. 2011. P. 102–109.
173. Thi Thu Thuy P., Lee Y. K., Lee S. Y. DTD2OWL: Automatic Transforming XML Documents into OWL Ontology // Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human. 2009. P. 125–131.

174. An Y., Borgida A., Mylopoulos J. Constructing Complex Semantic Mappings between XML Data and Ontologies // Proceedings of the International Semantic Web Conference (ISWC 2005) / Lecture Notes in Computer Science. 2005. Vol. 3729. P. 6–20.
175. Yahia N., Mokhtar S. A., Ahmed A. Automatic Generation of OWL Ontology from XML Data Source // International Journal of Computer Science Issues. 2012. Vol. 9, no. 2. P. 1–7.
176. W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. 2012. URL: https://www.w3.org/TR/xmlschema11-1/ (дата обращения: 2017-04-17).
177. W3C XML Specification DTD (“XMLspec”). 1998. URL: https://www.w3.org/XML/1998/06/xmlspec-report-19980910.htm (дата обращения: 2017-04-17).
178. Slota M., Leite J., Swift T. On updates of hybrid knowledge bases composed of ontologies and rules // Artificial Intelligence. 2015. Vol. 229. P. 33–104.
179. Загорулько Ю. А. О концепции интегрированной модели представления знаний // Известия Томского политехнического университета. Инжиниринг георесурсов. 2013. Т. 322, № 5. С. 98–103.
180. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. 2004. URL: https://www.w3.org/Submission/SWRL/ (дата обращения: 2017-04-17).
181. Avdeenko T. V., Makarova E. S. Integration of Case-based and Rule-based Reasoning Through Fuzzy Inference in Decision Support Systems // Proceedings of the XIIth International Symposium «Intelligent Systems», INTELS’16 / Procedia Computer Science. 2017. Vol. 103. P. 447–453.
182. Bassiliades N., Vlahavas I. R-DEVICE: an object-oriented knowledge base for RDF metadata // International Journal on Semantic Web and Information Systems. 2006. Vol. 2, no. 2. P. 24–90.
183. Meditskos G., Bassiliades N. CLIPS-OWL: A framework for providing object-oriented extensional ontology queries in a production rule engine // Data & Knowledge Engineering. 2011. Vol. 70. P. 661–681.

184. Verhodubs O., Grundspenkis J. Algorithm of Ontology Transformation to Concept Map for Usage in Semantic Web Expert System // Applied Computer Systems. 2013. Vol. 14, no. 1. P. 80–87.
185. Eriksson H. The JESSTAB approach to Protégé and JESS integration // Proceedings of the IFIP 17th World Computer Congress – TC12 Stream on Intelligent Information Processing. 2002. P. 237–248.
186. Corsar D., Sleeman D. Reusing JessTab rules in Protégé // Knowledge-Based Systems. 2006. Vol. 19, no. 5. P. 291–297.
187. Mei J., Bontas E. P., Lin Z. OWL2Jess: A transformational implementation of the OWL semantics // Proceedings of the Parallel and Distributed Processing and Applications – ISPA 2005 Workshops / Lecture Notes in Computer Science. 2005, Vol. 3759. P. 599–608.
188. Wang E., Kim Y. S. A Teaching Strategies Engine Using Translation from SWRL to Jess. In Proceedings of the Intelligent Tutoring Systems (ITS 2006) / Lecture Notes in Computer Science. 2006. Vol. 4053. P. 51–60.
189. SweetRules. 2005. URL: http://sweetrules.projects.semwebcentral.org/ (дата обращения: 2017-04-17).
190. Laera L., Tamma V., Bench-Capon T., Semeraro G. SweetProlog: A system to integrate ontologies and rules // Proceedings of the International Workshop on Rules and Rule Markup Languages for the Semantic Web / Lecture Notes in Computer Science. 2004. Vol. 3323. P. 188–193.
191. Akbari I., Yan B., Zhang J., Boley H. Visualizing SWRL Rules: From Unary/Binary Datalog and PSOA RuleML to Graphviz and Grailog // Proceedings of the 4th Canadian Semantic Web Symposium Montreal (CSWS) / CEUR Workshop Proceedings. 2013. Vol. 1054. P. 55–56.
192. Riveroa C. R., Hernández I., Ruiz D., Corchuelo R. Mapping RDF knowledge bases using exchange samples // Knowledge-Based Systems. 2016. Vol. 93, no. 1. P. 47–66.

193. Гадиатулин Р. А., Чуприна С. И. Rule-Mining: подход к автоматизированному извлечению онтологий // Proceedings of the XIIIth International Conference of Knowledge-Dialogue-Solution. 2007. P. 445–451.
194. Буч Г., Рамбо Д., Якобсон И. Язык UML. Руководство пользователя. 2-е изд. М.: ДМК Пресс, 2006. 496 с.
195. XML Metadata Interchange (XMI) Version 2.5.1 // OMG Document formal/2015-06-07. 2015. URL: http://www.omg.org/spec/XMI/2.5.1/ (дата обращения: 2017-04-17).
196. XML Topic Maps (XTM) 1.0. 2001. URL: http://www.topicmaps.org/xtm/ (дата обращения: 2017-04-17).
197. Deliyanni A., Kowalski R. A. Logic and Semantic Networks // Magazine Communications of the ACM. 1979. Vol. 22, no. 3. P. 184–192.
198. Минский М. Структура для представления знания // Психология машинного зрения. М.: Мир, 1978. С. 249–338.
199. Братко И. Алгоритмы искусственного интеллекта на языке Prolog. М.: Вильямс, 2004. 637 с.
200. Gruber T. R. A translation approach to portable ontologies // Knowledge Acquisition. 1993. Vol. 5, no. 2. P. 199–220.
201. Guarino N. Formal Ontology in Information Systems // Proceedings of the first international conference (FOIS'98). 1998. Vol. 46. P. 3–15.
202. Добров Б. В., Иванов В. В., Лукашевич Н. В., Соловьев В. Д. Онтологии и тезаурусы: учебно-методическое пособие. Казань: Изд-во Казанского ГУ, 2006. 156 с.
203. Berners-Lee T., Hendler J., Lassila O. The Semantic Web. Scientific American. 2001. P. 29–37.
204. Semantic Web. 2015. URL: https://www.w3.org/standards/semanticweb/ (дата обращения: 2017-04-17).
205. OWL Web Ontology Language Overview. 2004. URL: https://www.w3.org/TR/2004/REC-owl-features-20040210/ (дата обращения: 2017-04-17).

206. Baader F., Calvanese D., McGuinness D. L., Nardi D., PatelSchneider P. F. The Description Logic Handbook: Theory, Implementation, Applications. Cambridge: Cambridge University Press. 2003.
207. Resource Description Framework (RDF): Concepts and Abstract Syntax. 2004. URL: https://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ (дата обращения: 2017-04-17).
208. RDF Schema 1.1. 2014. URL: https://www.w3.org/TR/rdf-schema/ (дата обращения: 2017-04-17).
209. Лапшин В. А. Онтологии в компьютерных системах. М.: Научный мир, 2010. 224 с.
210. Grau B. C., Horrocks I., Motik B., Parsia B., Patel-Schneider P., Sattler U. OWL 2: The next step for OWL // Web Semantics: Science, Services and Agents on the World Wide Web. 2008. Vol. 6, no. 4. P. 309–322.
211. RDF/XML Syntax Specification (Revised). 2004. URL: https://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/ (дата обращения: 2017-04-17).
212. OWL 2 Web Ontology Language Conformance (Second Edition). 2012. URL: https://www.w3.org/TR/2012/REC-owl2-conformance-20121211/ (дата обращения: 2017-04-17).
213. RDF 1.1 Turtle. 2014. URL: https://www.w3.org/TR/turtle/ (дата обращения: 2017-04-17).
214. OWL 2 Web Ontology Language XML Serialization (Second Edition). 2012. URL: https://www.w3.org/TR/2012/REC-owl2-xml-serialization-20121211/ (дата обращения: 2017-04-17).
215. OWL 2 Web Ontology Language Manchester Syntax (Second Edition). 2012. URL: https://www.w3.org/TR/2012/NOTE-owl2-manchester-syntax-20121211/ (дата обращения: 2017-04-17).
216. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition). 2012. URL: https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/ (дата обращения: 2017-04-17).

217. Частиков А. П., Гаврилова Т. А., Белов Д. Л. Разработка экспертных систем. Среда CLIPS. СПб.: БХВ-Петербург, 2003. 393 с.
218. Лаврищева Е. М., Грищенко В. Н. Сборочное программирование. Основы индустрии программных продуктов: 2-изд. Дополненное и переработанное. К.: Наук. думка, 2009. 372с.
219. Лаврищева Е. М. Программная инженерия. Парадигмы, технологии и CASE-средства: Учебник. М.: Юрайт, 2016. 282 с.
220. Дал У., Дейкстра Э., Хоор К. Структурное программирование. М.: Мир, 1975. 245 с.
221. Ершов А. П. Введение в теоретическое программирование (беседы о методе). М.: Наука, 1977. 288 с.
222. Тыугу Э.Х. Концептуальное программирование. М.: Наука, 1984. 256 с.
223. Опарин Г. А. Сатурн – метасистема для построения пакетов прикладных программ // Пакеты прикладных программ. Методы и разработки. Новосибирск: Наука, 1982. С. 130–160.
224. Лаврищева Е. М. Методы программирования. Теория, инженерия, практика. К.: Наук. думка, 2006. 451с.
225. Горбунов-Посадов М. М. Расширяемые программы. М.: Полиптих, 1999. 336 с.
226. Горбунов-Посадов М. М. Облик многократно используемого компонента // Открытые системы. 1998. № 3. С. 45–49.
227. Буч Г. Объектно-ориентированный анализ. М.: Бином, 1998. 560 с.
228. Агафонов В. Н. Спецификация программ: понятийные средства и их организация. Новосибирск: Наука, 1987. 240 с.
229. Агафонов В. Н. Требования и спецификации в разработке программ. М.: Мир, 1984. 344 с.
230. Олищук А. В. Разработка Web-приложений на PHP 5. Профессиональная работа. М.: Вильямс, 2006. 352 с.
231. Graphical Modeling Framework (GMF) Tooling – Eclipse. 2014. URL: http://www.eclipse.org/gmf-tooling/ (дата обращения: 2017-04-17).

232. Greenfield J., Short K., Cook S., Kent S., Crupi J. Software factories: assembling applications with patterns, models, frameworks, and tools. Wiley Publishing. 2004.
233. Николайчук О. А. Методы, модели и инструментальное средство для исследования надежности и безопасности сложных технических систем: автореф. дис. докт. тех. наук: 05.13.01 / Николайчук Ольга Анатольевна. М., 2011. 37 с.
234. Rule Visual Modeling Language (RVML). 2016. URL: http://www.knowledge-core.ru/index.php?p=rvml (дата обращения: 2017-04-17).
235. OWL 2 Web Ontology Language MOF-Based Metamodel (Second Edition). 2010. URL: https://www.w3.org/2007/OWL/wiki/MOF-Based_Metamodel#ref-owl-2-specification (дата обращения: 2017-04-17).
236. Khan A., Sum M. Introducing Design Patterns in XML Schemas. 2006. URL: http://www.oracle.com/technetwork/java/design-patterns-142138.html (дата обращения: 2017-04-17).
237. Ахо А. В., Лам М. С., Сети Р., Ульман Дж. Д. Компиляторы: принципы, технологии и инструментарий, 2-е изд. М.: Вильямс, 2008. 1184 с.
238. Мартыненко Б. К. Языки и трансляции: Учеб. Пособие, 2-е изд., испр. и доп. СПб.: Изд-во С.-Петерб. ун-та, 2008. 257 с.
239. Extended Backus-Naur Form (EBNF). ISO-IEC 14977-1996(E). 2004. URL: http://www.iso.org/iso/catalogue_detail?csnumber=26153 (дата обращения: 2017-04-17).
240. Рыбина Г. В., Смирнов В. В. Методы и алгоритмы верификации баз знаний в интегрированных экспертных системах // Известия Российской академии наук. Теория и системы управления. 2007. № 4. С. 91–102.
241. Берман А. Ф., Грищенко М. А., Николайчук О. А., Юрин А. Ю. Проблемно-ориентированный редактор продукционных баз знаний // Программные продукты и системы. 2015. № 2. С. 13–19.
242. Дородных Н. О., Грищенко М. А., Юрин А. Ю. Система программирования продукционных баз знаний: Personal Knowledge Base Designer // Материалы

VI международной научно-технической конференции «Открытые семантические технологии проектирования интеллектуальных систем OSTIS-2016». 2016. С. 209–212.

243. Грищенко М. А., Дородных Н. О., Юрин А. Ю. Система для прототипирования продукционных баз знаний // Труды Пятнадцатой национальной конференции по искусственному интеллекту с международным участием (КИИ-2016). 2016. Т. 3. С. 254–260.
244. Knowledge Base Development System (KBDS). 2017. URL: http://kbds.knowledge-core.ru/ (дата обращения: 2017-04-17).
245. Yii2 Framework. 2017. URL: http://www.yiiframework.com/ (дата обращения: 2017-04-17).
246. Winesett J. Agile Web Application Development with Yii1.1 and PHP5. Birmingham: Packt Publishing Ltd. 2010.
247. Фримен А. ASP.NET MVC 4 с примерами на C# 5.0 для профессионалов, 4-е издание. М.: Вильямс, 2013. 688 с.
248. Фримен А. jQuery для профессионалов. М.: Вильямс, 2012. 960 с.
249. jsPlumb. 2017. URL: https://jsplumbtoolkit.com/ (дата обращения: 2017-04-17).
250. MySQL. 2017. URL: https://www.mysql.com/ (дата обращения: 2017-04-17).
251. PostgreSQL. 2017. URL: https://www.postgresql.org/ (дата обращения: 2017-04-17).
252. IBM Rational Rose Enterprise. 2017. URL: http://www-03.ibm.com/software/products/ru/enterprise/ (дата обращения: 2017-04-17).
253. IHMC CmapTools. 2017. URL: http://cmap.ihmc.us/ (дата обращения: 2017-04-17).
254. Берман А. Ф. Деградация механических систем. Новосибирск: Наука, 1998. 320 с.
255. Берман А. Ф., Николайчук О. А. Модели, знания и опыт для управления техногенной безопасностью // Проблемы управления. 2010. № 2. С. 53–60.
256. Берман А. Ф., Николайчук О. А., Юрин А. Ю., Кузнецов К. А. Поддержка принятия решений на основе продукционного подхода при проведении

экспертизы промышленной безопасности // Химическое и нефтегазовое машиностроение. 2014. № 11. С. 28–35.

257. Ларичев О. И., Мечитов А. И., Мошкович Е. М., Фуремс Е. М. Выявление экспертных знаний (процедуры и реализации). М.: Наука, 1989. 128 с.
258. Берман А. Ф., Николайчук О. А., Павлов Н. Ю., Юрин А. Ю. Методы и средства автоматизированного построения деревьев событий и отказов // Автоматизация и современные технологии. 2013. № 9. С. 8–16.
259. Дородных Н. О., Юрин А. Ю. Использование онтологий при разработке продукционных экспертных систем // Материалы VI Всероссийской конференции с международным участием «Знания-Онтологии-Теория ЗОНТ-2017». 2017. Т. 1. С. 129–138.
260. Дородных Н. О., Коршунов С. А. Концепция веб-сервиса для синтеза императивного описания пространственно-временных сцен на основе онтологий // Тезисы конференции «Ляпуновские чтения – 2016». 2016. С. 32.
261. Дородных Н. О., Коршунов С. А. Визуализация пространственно-временных сцен на основе их императивного описания и онтологий // Тезисы докладов XVI Всероссийской конференции молодых ученых по математическому моделированию и информационным технологиям. 2015. С. 78.
262. Дородных Н. О., Малтугуева Г. С., Петровский А. Б., Юрин А. Ю. Реализация многометодного подхода к обработке индивидуальных предпочтений // Материалы III Всероссийской Поспеловской конференции с международным участием (ГИСИС’2016). 2016. С. 173–181.
263. Дородных Н. О., Коршунов С. А. Программное средство визуализации результатов агентного моделирования // Тезисы докладов XV Всероссийской конференции молодых ученых по математическому моделированию и информационным технологиям. 2014. С. 67–68.
264. Дородных Н. О., Коршунов С. А., Павлов А. И., Столбов А. Б. Программное средство визуализации результатов агентного моделирования // Материалы

Международной научно-практической конференции «Фундаментальная информатика, информационные технологии и системы управления: реалии и перспективы. FIITM-2014». 2014. С. 392–398.

265. Дородных Н. О., Юрин А. Ю. Создание прецедентных экспертных систем на основе трансформации моделей // Тезисы докладов XIII Всероссийской конференции молодых ученых «Моделирование, оптимизация и информационные технологии». 2017. С. 31.
266. Дородных Н. О., Малтугуева Г. С., Николайчук О. А., Юрин А. Ю. Разработка нечетких продукционных баз знаний на основе трансформации концептуальных моделей // Тезисы конференции «Ляпуновские чтения – 2016». 2016. С. 33.
267. Дородных Н. О., Николайчук О. А., Павлов А. И. Метод поддержки извлечения знаний для решения мультидисциплинарных проблем на основе онтологии // Тезисы конференции «Ляпуновские чтения – 2016». 2016. С. 51.
268. Дородных Н. О., Коршунов С. А. Программное средство визуализации на основе библиотеки WebGL // Информационные и математические технологии в науке и управлении. 2016. № 2. С. 129–136.
269. Дородных Н. О., Малтугуева Г. С., Николайчук О. А. Метод представления и обработки экспертных знаний, извлеченных из концептуальных моделей // Труды Седьмой международной конференции «Системный анализ и информационные технологии». 2017. С. 363–368.
270. ClipsWin: CLIPS Rule Based Programming Language. 2008. URL: https://sourceforge.net/p/clipsrules/news/2008/01/clipswin-6241/ (дата обращения: 2017-04-17).

Приложение А. Описание абстрактного синтаксиса OWL2 DL

С целью описания семантики (абстрактного синтаксиса, abstract syntax) OWL2 DL [205] представим основные конструкции языка (классы и свойства) в Таблице А.1 вместе с их соответствиями в дескрипционной логике (Description Logic, DL) – sROIQ(D) [206]. Однако существует еще два дополнительных типа свойств в OWL2, которые не имеют аналогов в DL, а именно: свойства аннотаций (owl:AnnotationProperty) и свойства онтологии (owl:OntologyProperty), которые включают в себя некоторые мета-свойства онтологий.

Основным компонентом онтологий OWL2 является набор аксиом – утверждений, которые всегда истинны. Аксиомы позволяют осуществлять умозаключения в рамках онтологии. OWL2 предоставляет обширный набор различных аксиом; все они расширяют класс Axiom в структурной спецификации [216]. Аксиомы OWL2 могут быть высказываниями: о классах, об объектных свойствах, об свойствах-значений, об определении типов данных, а также сюда входят ключевые аксиомы, аксиомы об аннотациях и так называемые аксиомы-утверждения (иногда называемые – фактами).

Таблица А.2 включает в себя конструкции аксиом OWL2 DL вместе с их соответствиями в sROIQ(D).

Таблица А.1. Конструкции классов и свойств OWL2 DL

Элементы OWL2	Обозначение DL	Описание
Class (*A*)	A	Объявление некоторого класса объектов (индивидов)
Class (owl:Thing)	$\top$	Общий класс (сущность), обозначающий множество всех индивидов
Class (owl:Nothing)	$\bot$	Общий класс (ничто), обозначающий пустое множество
ObjectProperty (R_A)	R_A	Объектное свойство, определяет отношение между представителями двух классов
TopObjectProperty	U	Объектное свойство, соединяющее все возможные пары индивидов
BottomObjectProperty	$\neg U$	Объектное свойство, не связывающее ни одной пары индивидов
DatatypeProperty (*T*)	T	Свойство-значение, определяет отношение между представителями классов и RDF-литералами или типами данных, определяемых XML Schema
TopDataProperty	U_D	Свойство-значение, соединяющее все возможные индивиды со всеми литералами
BottomDataProperty	$\neg U_D$	Свойство-значение, не связывающее любого индивида с литералом
ObjectIntersectionOf (*C*, *D*)	$C \sqcap D$	Пересечение (конъюнкция) концептов (классов)
ObjectUnionOf (*C*, *D*)	$C \sqcup D$	Объединение (дизъюнкция) концептов (классов)
ObjectComplementOf (*C*)	$\neg C$	Дополнение (отрицание) концепта (класса)
ObjectAllValuesFrom (*R*, *C*)	$\forall R.C$	Локальное ограничение объектного свойства класса

ObjectSomeValuesFrom (*R*, *C*)	$\exists R.C$	Локальное ограничение объектного свойства класса
ObjectHasValue (*R*, *o*)	$\exists R.\{o\}$	Указание на существование специфических значений свойства
DataAllValuesFrom (*T*, *d*)	$\forall T.\mathbf{d}$	Локальное ограничение свойств-значений класса
DataSomeValuesFrom (*T*, *d*)	$\exists T.\mathbf{d}$	Локальное ограничение свойств-значений класса
DataHasValue (*T*, *v*)	$\exists T.\{\upsilon\}$	Указание на существование специфических значений свойства
ObjectOneOf ($o_1,\ldots,o_m$)	$\{o_1\} \sqcup \{o_2\} \sqcup \{o_m\}$	Определение класса через прямое перечисление его членов (определение диапазона класса)
ObjectMinCardinality (*n*, *S*, *C*)	$(\geq n\ S.C)$	Указание нижнего числового предела для объектного свойства
ObjectMaxCardinality (*n*, *S*, *C*)	$(\leq n\ S.C)$	Указание верхнего числового предела для объектного свойства
ObjectExactCardinality (*n*, *S*, *C*)	$(\geq n\ S.C) \sqcap (\leq n\ S.C)$	Точное указание на числовой интервал для объектного свойства
ObjectMinCardinality (*n*, *S*)	$(\geq n\ S.\top)$	Указание нижнего числового предела для объектного свойства множества всех индивидов
ObjectMaxCardinality (*n*, *S*)	$(\leq n\ S.\top)$	Указание верхнего числового предела для объектного свойства множества всех индивидов
ObjectExactCardinality (*n*, *S*)	$(\geq n\ S.\top) \sqcap (\leq n\ S.\top)$	Точное указание на числовой интервал для объектного свойства множества всех индивидов
DataMinCardinality (*n*, *T*, *d*)	$(\geq n\ T.\mathbf{d})$	Указание нижнего числового предела для свойства-значения

DataMaxCardinality (*n*, *T*, *d*)	$(\leq n\, T.\mathbf{d})$	Указание верхнего числового предела для свойства-значения множества всех индивидов
DataExactCardinality (*n*, *T*, *d*)	$(\geq n\, T.\mathbf{d}) \sqcap (\leq n\, T.\mathbf{d})$	Точное указание на числовой интервал для свойства-значения множества всех индивидов
DataMinCardinality (*n*, *T*)	$(\geq n\ T.\top)$	Указание нижнего числового предела для свойства-значения множества всех индивидов
DataMaxCardinality (*n*, *T*)	$(\leq n\ T.\top)$	Указание верхнего числового предела для свойства-значения
DataExactCardinality (*n*, *T*)	$(\geq n\ T.\top) \sqcap (\leq n\ T.\top)$	Точное указание на числовой интервал для свойства-значения
ObjectExistsSelf (*S*)	$\exists S.\mathrm{Self}$	Совокупность индивидов, которые связаны с самим собой при помощи данного объектного свойства

Таблица А.2. Конструкции аксиом OWL2 DL

Элементы OWL2	Обозначение DL	Описание
ClassAssertion (*a*, *C*)	$a:C$	Утверждение принадлежности индивида к классу
ObjectPropertyAssertion (*R*, *a*, *b*)	$(a,b):R$	Утверждение, что индивид связан объектным свойством с некоторым другим индивидом
NegativeObjectPropertyAssertion (*R*, *a*, *b*)	$(a,b):\neg R$	Утверждение, что индивид не связан объектным свойством с некоторым другим индивидом
DataPropertyAssertion (*T*, *a*, *v*)	$(a,\upsilon):T$	Утверждение, что индивид связан свойством-значением с некоторым литералом
NegativeDataPropertyAssertion (*T*, *a*, *v*)	$(a,\upsilon):\neg T$	Утверждение, что индивид не связан свойством-значением с некоторым литералом

SameIndividual ($a_1,\ldots,a_m$)	$a_i = a_j, 1 \le i \le j \le m$	Утверждение, что несколько индивидов равны друг другу
DifferentIndividuals ($a_1,\ldots,a_m$)	$a_i \ne a_j, 1 \le i \le j \le m$	Утверждение, что несколько индивидов отличаются друг от друга (семантически различны)
SubClassOf (C_1, C_2)	$C_1 \sqsubseteq C_2$	Утверждение, что каждый экземпляр одного класса также является экземпляром другого класса (построение иерархии классов)
EquivalentClasses ($C_1,\ldots,C_m$)	$C_1 \equiv \ldots \equiv C_m$	Утверждение, что несколько классов эквивалентны друг другу
DisjointClasses ($C_1,\ldots,C_m$)	$\mathrm{dis}(C_1 \equiv \ldots \equiv C_m)$	Утверждение, что несколько классов попарно не пересекаются, т.е. у них нет общих экземпляров
DisjointUnion ($C, C_1,\ldots,C_m$)	$\mathrm{disUnion}(C_1 \equiv \ldots \equiv C_m)$	Утверждение определения класса как несвязного объединения нескольких классов (выражение покрывающего ограничения)
SubObjectPropertyOf (subObjectPropertyChain ($R_1,\ldots,R_m$) R)	$R_1,\ldots,R_m \sqsubseteq R$	Аксиома включения цепочки свойств (при условии, что нет циклических включений)
SubObjectPropertyOf (R_1, R_2)	$R_1 \sqsubseteq R_2$	Утверждение, что расширение выражения одного объектного свойства входит в расширение другого выражения объектного свойства
SubDataPropertyOf (T_1, T_2)	$T_1 \sqsubseteq T_1$	Утверждение, что расширение выражения одного свойства-значения входит в расширение другого выражения свойства-значения
EquivalentObjectProperties ($R_1,\ldots,R_m$)	$R_1 \equiv \ldots \equiv R_m$	Утверждение, что расширения нескольких выражений объектных свойств являются одинаковыми

EquivalentDataProperties ($T_1,\ldots,T_m$)	$T_1 \equiv \ldots \equiv T_m$	Утверждение, что некоторые выражения свойств-значений имеют одинаковое расширение
ObjectPropertyDomain (*R*, *C*)	$\mathrm{domain}(R,C)$	Утверждения ограничения двух индивидов, связанных объектным свойством (утверждение области значений и определений)
ObjectPropertyRange (*R*, *C*)	$\mathrm{range}(R,C)$	
DataPropertyDomain (*T*, *d*)	$\mathrm{domain}(T,\mathbf{d})$	Утверждения ограничений для индивидов и литералов, связанных свойством-значения (утверждение области значений и определений)
DataPropertyRange (*T*, *d*)	$\mathrm{range}(T,\mathbf{d})$	
InverseObjectProperties (R_1, R_2)	$R_1 \equiv R_2^-$	Утверждение, что два выражения разных объектных свойств являются инверсией (обратными) друг другу
FunctionalObjectProperty (*S*)	$\mathrm{func}(S)$	Утверждение, что объектное свойство является функциональным
FunctionalDataProperty (*T*)	$\mathrm{func}(T)$	Утверждение, что свойство-значения является функциональным
InverseFunctionalObjectProperty (*S*)	$\mathrm{func}(S^-)$	Утверждение, что объектное свойство является обратно функциональным
TransitiveObjectProperty (*R*)	$\mathrm{trans}(R)$	Утверждение, что объектное свойство является транзитивным
DisjointObjectProperties (S_1, S_2)	$\mathrm{dis}(S_1,S_2)$	Утверждение, что расширения выражений нескольких объектных свойств попарно не пересекаются, т.е. у них нет общей пары связанных индивидов
DisjointDataProperties (T_1, T_2)	$\mathrm{dis}(T_1,T_2)$	Утверждение, что расширения выражений нескольких свойств-значений попарно не пересекаются, т.е. у них нет общей пары индивид-литерал

ReflexiveObjectProperty (*R*)	$\mathrm{ref}(R)$	Утверждение, что объектное свойство является рефлексивным
IrreflexiveObjectProperty (*S*)	$\mathrm{iir}(S)$	Утверждение, что объектное свойство является иррефлексивным
SymmetricObjectProperty (*R*)	$\mathrm{sym}(R)$	Утверждение, что объектное свойство является симметричным
AsymmetricObjectProperty (*S*)	$\mathrm{asy}(S)$	Утверждение, что объектное свойство является асимметричным

Приложение Б. Описание синтаксиса TMRL

С целью описания синтаксиса (concrete syntax) TMRL представим основные конструкции TMRL с использованием Расширенной Формы Бэкуса-Наура (нотация РБНФ).

Расширенная форма Бэкуса-Наура (Extended Backus-Naur Form, EBNF) – формальная система определения синтаксиса, в которой одни синтаксические категории последовательно определяются через другие. РБНФ используется для описания контекстно-свободных формальных грамматик.

Описание грамматики в РБНФ представляет собой набор правил, определяющих отношения между терминалами и нетерминалами.

Терминалы – это минимальные элементы грамматики, не имеющие собственной грамматической структуры. В РБНФ терминальные символы – это либо предопределённые идентификаторы (имена, считающиеся заданными для данного описания грамматики), либо цепочки – последовательности символов в кавычках или апострофах.

Нетерминалы – это элементы грамматики, имеющие собственные имена и структуру. Каждый нетерминальный символ состоит из одного или более терминальных и/или нетерминальных символов, сочетание которых определяется правилами грамматики. В РБНФ каждый нетерминальный символ имеет имя, которое представляет собой строку символов.

Правило в РБНФ имеет вид:

$$идентификатор = выражение. ,$$

где *идентификатор* – это наименование нетерминального символа; *выражение* – соответствующая правилам РБНФ комбинация терминальных и/или нетерминальных символов, а также специальных знаков; знак «*точка*» или «*точка с запятой*» в конце правила – это специальный символ, указывающий на завершение правила.

Описание терминальных символов нотации РБНФ, определенных в стандарте ISO/IEC 14977 [239], представлено в Таблице Б.1.

Таблица Б.1. Представление терминальных символов нотации РБНФ

Элементы нотации	Описание
=	Определение
,	Конкатенация
;	Прекращение
.	Прекращение
|	Выбор
-	Исключение
*	Повторение
[…]	Условное вхождение – обозначение необязательного элемента выражения
{…}	Повторение – обозначение конкатенации любого числа элементов (включая нуль)
(…)	Группировка – обозначение группировки элементов при формировании сложных выражений
"…"	Терминальная цепочка (литерал) – символ или группа символов
`…`	Терминальная цепочка (литерал) – символ или группа символов
(*…*)	Комментарий
?...?	Специальные последовательности

Для упрощения восприятия описания TMRL опустим некоторые нетерминальные символы, которые не несут особой смысловой нагрузки и необходимы только для точности выражения синтаксиса TMRL:

```
Пробел = ? US-ASCII character 32 ? ;
Переход на следующую строку = ? US-ASCII character 13 ? ;
```

Таким образом, описание контекстно-свободной формальной грамматики TMRL, имеет следующий вид:

```
Модель трансформации на TMRL = "Transformation Model" , Заголовок, "{" , Модель трансформации , "}" , Оператор вызова;
Модель трансформации = Исходная метамодель, Целевая метамодель, Оператор трансформации;
```

Исходная метамодель = **"Source Meta-Model"** , Заголовок, **"{"** , Тело метамодели, **"}"**;

Целевая метамодель = **"Target Meta-Model"** , Заголовок, **"{"** , Тело метамодели, **"}"**;

Оператор трансформации = **"Transformation"** , Заголовок оператора трансформации , **"{"** , Тело трансформации, **"}"**;

Заголовок = Буква , { Буква | Цифра | Символ } ;

Буква = **"A"** | **"B"** | **"C"** | **"D"** | **"E"** | **"F"** | **"G"** | **"H"** | **"I"** | **"J"** | **"K"** | **"L"** | **"M"** | **"N"** | **"O"** | **"P"** | **"Q"** | **"R"** | **"S"** | **"T"** | **"U"** | **"V"** | **"W"** | **"X"** | **"Y"** | **"Z"** | **"a"** | **"b"** | **"c"** | **"d"** | **"e"** | **"f"** | **"g"** | **"h"** | **"i"** | **"j"** | **"k"** | **"l"** | **"m"** | **"n"** | **"o"** | **"p"** | **"q"** | **"r"** | **"s"** | **"t"** | **"u"** | **"v"** | **"w"** | **"x"** | **"y"** | **"z"** | **"А"** | **"а"** | **"Б"** | **"б"** | **"В"** | **"в"** | **"Г"** | **"г"** | **"Д"** | **"д"** | **"Е"** | **"е"** | **"Ё"** | **"ё"** | **"Ж"** | **"ж"** | **"З"** | **"з"** | **"И"** | **"и"** | **"Й"** | **"й"** | **"К"** | **"к"** | **"Л"** | **"л"** | **"М"** | **"м"** | **"Н"** | **"н"** | **"О"** | **"о"** | **"П"** | **"п"** | **"Р"** | **"р"** | **"С"** | **"с"** | **"Т"** | **"т"** | **"У"** | **"у"** | **"Ф"** | **"ф"** | **"Х"** | **"х"** | **"Ц"** | **"ц"** | **"Ч"** | **"ч"** | **"Ш"** | **"ш"** | **"Щ"** | **"щ"** | **"Ъ"** | **"ъ"** | **"Ы"** | **"ы"** | **"Ь"** | **"ь"** | **"Э"** | **"э"** | **"Ю"** | **"ю"** | **"Я"** | **"я"** ;

Цифра = **"0"** | **"1"** | **"2"** | **"3"** | **"4"** | **"5"** | **"6"** | **"7"** | **"8"** | **"9"** ;

Символ = **"-"** | **"."** ;

Тело метамодели = Элементы , Связи ;

Элементы = **"Elements"** , **"["** , { Элемент } , **"]"** ;

Элемент = Наименование , [Атрибуты] , "," ;

Атрибуты = **"attributes"** , **"("** , { Атрибут } , **")"** ;

Атрибут = Наименование , "," ;

Наименование = Буква , { Буква | Цифра | Символ } ;

Связи = **"Relationships"** , **"["** , { Связь } , **"]"** ;

Связь = Ассоциация | Связь по идентификатору ;

Ассоциация = Левый элемент ассоциации , **"is associated with"** , Правый элемент ассоциации , "," ;

Левый элемент ассоциации = Наименование ;

Правый элемент ассоциации = Наименование ;

Связь по идентификатору = Левый элемент связи по идентификатору, **"is"** , Правый элемент связи по идентификатору , "," ;

Левый элемент связи по идентификатору = Наименование элемента , **"("** , Наименование атрибута , **")"** ;

Правый элемент связи по идентификатору = Наименование элемента , **"("** , Наименование атрибута , **")"** ;

Наименование элемента = Наименование ;

Наименование атрибута = Наименование ;

Заголовок оператора трансформации = Название исходной метамодели , **"to"** , Название целевой метамодели ;

Название исходной метамодели = Заголовок ;

Название целевой метамодели = Заголовок ;

Тело трансформации = { Правило } ;

Правило = Заголовок правила , **"["** , Тело правила , **"]"** ;

Заголовок правила = **"Rule"** , Исходные элементы , **"to"** , Целевые элементы , **"priority"** , Цифра ;

Исходные элементы = Единичный исходный элемент | Множество исходных элементов ;

Единичный исходный элемент = Наименование ;

Множество исходных элементов = **"("** , Исходный элемент , { Дополнительный исходный элемент } , **")"** ;

Дополнительный исходный элемент = "," , Исходный элемент ;

Исходный элемент = Наименование ;

Целевые элементы = Единичный целевой элемент |Множество целевых элементов ;

Единичный целевой элемент = Наименование ;

Множество целевых элементов = **"("** , Целевой элемент , { Дополнительный целевой элемент } , **")"** ;

Дополнительный целевой элемент = "," , Целевой элемент ;

Целевой элемент = Наименование ;

Тело правила = { Выражение определения } , { Условное выражение } ;

Выражение определения = Левый элемент выражения , **"is"** , Правый элемент выражения ;

Левый элемент выражения = Целевой элемент , **"("** , Атрибут целевого элемента , **")"** ;

Атрибут целевого элемента = Наименование ;

Правый элемент выражения = Исходный элемент , **"("** , Атрибут исходного элемента , **")"** , [Дополнительный правый элемент выражения] ;

Атрибут исходного элемента = Наименование ;

Дополнительный правый элемент выражения = { **"or"** Исходный элемент , **"("** , Атрибут исходного элемента , **")"** } ;

Условное выражение = Левый элемент выражения , **"is"** , Значение выражения , [Условие] , [Альтернативное значение выражения];

Условие = **"["** , Описание условия , **"]"**

Альтернативное значение выражения = **"or"** , Значение выражения , **"["** , Условие , **"]"** ;

Описание условия = **"if"** , **"("** , Правый элемент выражения , **"is"** , Значение выражения **")"** , [Дополнительное условие] ;

Дополнительное условие = **"and"** , **"("** , Правый элемент выражения , **"is"** , Значение выражения **")"** ;

Значение выражения = **"‟"** , { Буква | Цифра | Символы } , **"″"** ;

Символы = **"@"** | **"#"** | **"$"** | **"%"** | **"^"** | **"&"** | **"*"** | **"/"** | **"+"** | **"-"** | **"."** | **"?"** | **"!"** | **","** | **";"** | **":"** | **"′"** | **"["** | **"]"** | **"{"** | **"}"** | **"="** | **"_"** | **" "** | **"\"** ;

Оператор вызова = **"Call"** , Название программного компонента трансформации , **","** , Путь к концептуальной модели , [**","** , Путь сохранения базы знаний] ;

Название программного компонента трансформации = { Буква | Цифра | Символы } ;

Путь к концептуальной модели = { Буква | Цифра | Символы } ;

Путь сохранения базы знаний = { Буква | Цифра | Символы } ;

Приложение В. Описание методов интерфейса взаимодействия

С целью описания интерфейса взаимодействия веб-ориентированной программной системы (KBDS) с внешними программными средствами представим методы (функции) для доступа к разработанным программным компонентам по части импорта концептуальной модели и генерации кода БЗ на ЯПЗ CLIPS или OWL.

Определим значения типов и статусов программного компонента и БЗ для формирования запросов к соответствующим методам (функциям) интерфейса взаимодействия.

Значения типов программных компонентов:

- интегрированный компонент анализа (модель онтологии) = 0;
- интегрированный компонент анализа (модель продукций) = 1;
- интегрированный компонент кодогенерации (OWL) = 2;
- интегрированный компонент кодогенерации (CLIPS) = 3;
- автономный компонент кодогенерации (OWL) = 4;
- автономный компонент кодогенерации (CLIPS) = 5.

Значения статусов программных компонентов:

- черновой компонент = 0;
- сгенерированный компонент = 1;
- устаревший компонент = 2.

Значения типов БЗ:

- онтология = 0;
- продукции = 1.

Значения статусов БЗ:

- открытая = 0;
- закрытая = 1.

Далее опишем основные методы (функции) интерфейса взаимодействия:

1. getAllModulesList

Описание: получение списка всех программных компонентов созданных в KBDS.

Метод запроса: GET.

Параметры: нет.

Метод возвращает строку (если найден хоть один программный компонент):

~id=идентификатор (номер) компонента**;name=** наименование компонента**;description=**описание компонента**;type=**тип компонента**;status=**статус компонента**~**…

Где ~ – разделитель блока описания конкретного модуля,

; – разделитель информации внутри блока.

Иначе метод возвращает: false.

Пример обращения: http://kbds.knowledge-core.ru/api/get-all-modules-list.

Пример использования функции getAllModulesList в программном средстве PKBD для получения списка всех доступных программных компонентов (модулей-конверторов) представлен на Рисунке В.1.

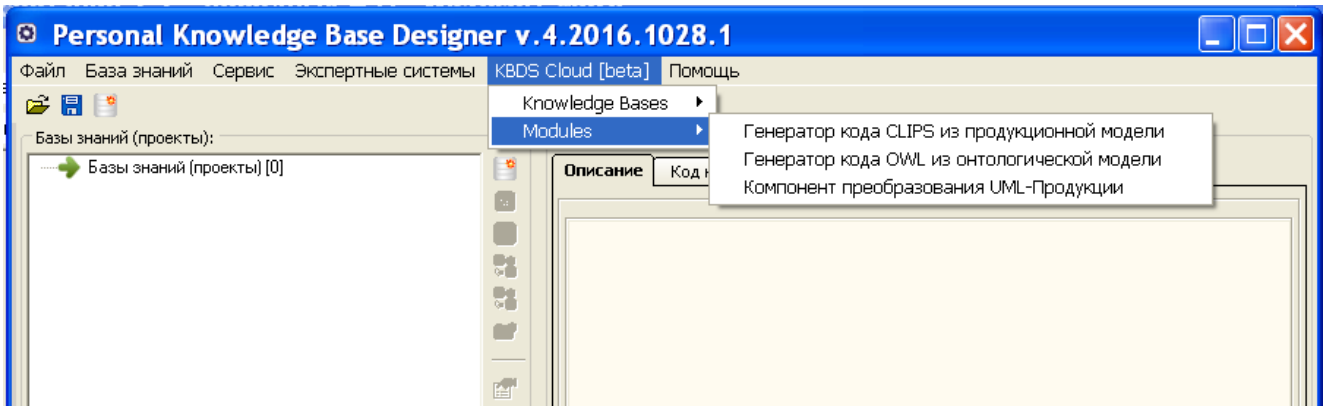

Рисунок В.1. Пример использования функции getAllModulesList в PKBD для получения списка доступных модулей

2. getModulesList

Описание: получение списка программных компонентов с определенным типом и статусом, созданных в KBDS.

Метод запроса: GET.

Параметры:

- тип программного компонента – целое числовое значение (от 0 до 5);
- статус программного компонента – целое числовое значение (от 0 до 2).

Метод возвращает строку (если найден хоть один программный компонент):

~**id**=идентификатор (номер) компонента;**name**= наименование компонента;**description**=описание компонента~…

Где ~ – разделитель блока описания конкретного модуля,

; – разделитель информации внутри блока.

Иначе метод возвращает: false.

Пример обращения: http://kbds.knowledge-core.ru/api/get-modules-list/1/2.

3. getAllKnowledgeBasesList

Описание: получение списка всех БЗ созданных в KBDS.

Метод запроса: GET.

Параметры: нет.

Метод возвращает строку (если найдена хоть одна БЗ):

~id=идентификатор (номер) базы знаний;**name=** наименование базы знаний;**description=**описание базы знаний;**subject_domain=**наименование предметной области базы знаний;type=тип базы знаний;**status=**статус базы знаний;**author=**логин автора базы знаний**~**…

Где ~ – разделитель блока описания конкретной БЗ,

; – разделитель информации внутри блока.

Иначе метод возвращает: false.

Пример обращения: http://kbds.knowledge-core.ru/api/get-all-knowledge-bases-list.

Пример использования функции getAllKnowledgeBasesList в программном средстве PKBD для получения списка всех доступных БЗ представлен на Рисунке В.2.

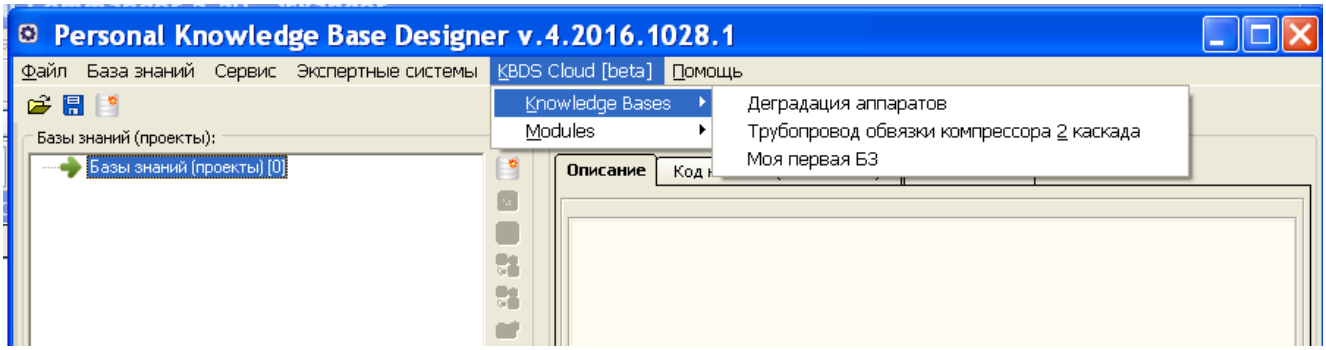

Рисунок В.2. Пример использования функции getAllKnowledgeBasesList в PKBD для получения списка доступных БЗ

4. getKnowledgeBasesList

Описание: получение списка БЗ с определенным типом и статусом, созданных в KBDS.

Метод запроса: GET.

Параметры:

- тип БЗ – целое числовое значение (от 0 до 1);
- статус БЗ – целое числовое значение (от 0 до 1).

Метод возвращает строку (если найдена хоть одна БЗ):

~id=идентификатор (номер) базы знаний**;name=** наименование базы знаний**;description=**описание базы знаний**;subject_domain=**наименование предметной области базы знаний**;author=**логин автора базы знаний**~**…

Где ~ – разделитель блока описания конкретной БЗ,

; – разделитель информации внутри блока.

Иначе метод возвращает: false.

Пример обращения: http://kbds.knowledge-core.ru/api/get-knowledge-bases-list/1/2.

5. importConceptualModel

Описание: импорт концептуальной модели и создание на ее основе БЗ в системе KBDS. Методом POST необходимо отправить переменную с наименованием «file», которая содержит XML-файл концептуальной модели.

Метод запроса: POST.

Параметры:

- идентификатор БЗ – целое числовое значение;
- идентификатор программного компонента – целое числовое значение.

Метод возвращает строку (ход выполнения импорта концептуальной модели).

Если запрос не POST, метод возвращает: false.

Пример обращения: http://kbds.knowledge-core.ru/api/import-conceptual-model/1/2.

6. exportKnowledgeBase

Описание: экспорт кода БЗ из системы KBDS.

Метод запроса: GET.

Параметры: идентификатор БЗ – целое числовое значение.

Метод возвращает файл сгенерированной БЗ в формате CLIPS или OWL.

Пример обращения: http://kbds.knowledge-core.ru/api/export-knowledge-base/1

Пример использования функции exportKnowledgeBase в программном средстве PKBD для получения содержимого БЗ «Деградация аппаратов» представлен на Рисунке В.3.

На Рисунке В.4 представлен результат выполнения функции exportKnowledgeBase.

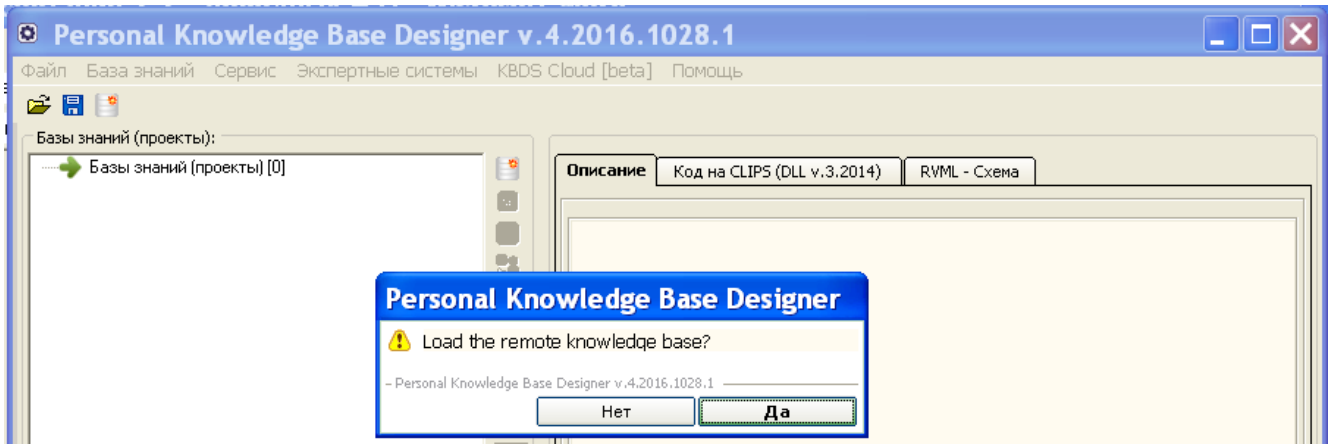

Рисунок В.3. Пример использования функции exportKnowledgeBase в PKBD для получения содержимого БЗ «Деградация аппаратов»

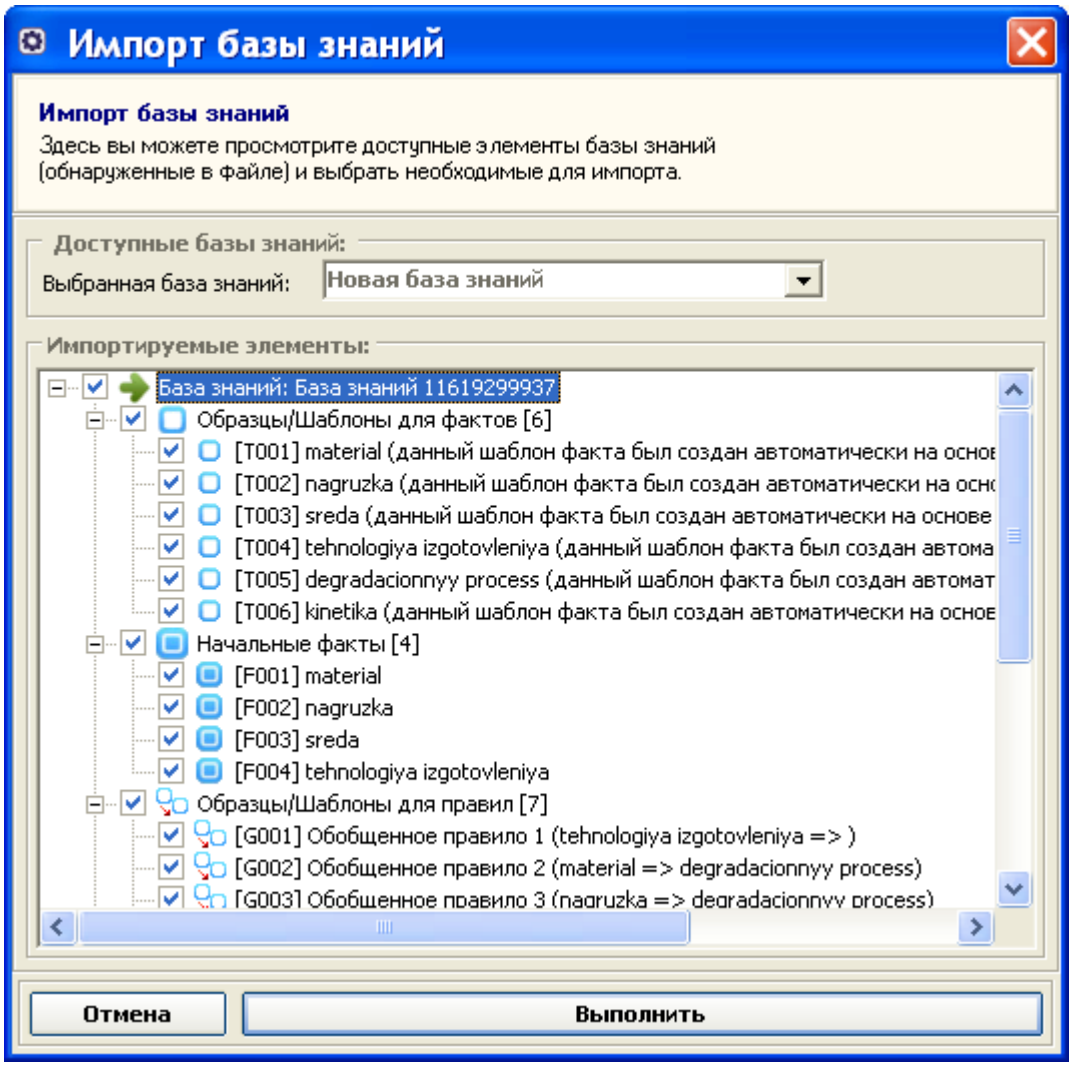

Рисунок В.4. БЗ «Деградация аппаратов» – результат выполнения функции exportKnowledgeBase в PKBD

С целью описания интерфейса взаимодействия веб-ориентированной программной системы (KBDS) с внешними программными средствами представим методы (функции) для взаимодействия с моделью онтологии в Таблице В.1 и с моделью продукций в Таблице В.2.

Таблица В.1. Методы взаимодействия с моделью онтологии

Название	Сигнатура	Описание
createOntology	name – наименование модели онтологии, description – описание модели онтологии, xml_file – файл концептуальной модели в формате XML.	Создание модели онтологии на основе импорта концептуальной модели.
generateOWLCode	file_name – наименование файла БЗ, ontology_name – наименование модели онтологии.	Генерация кода БЗ на ЯПЗ OWL, на основе модели онтологии.
addConcept	name – наименование понятия, description – описание понятия, ontology_id – идентификатор модели онтологии.	Добавление нового понятия.
updateConcept	id – идентификатор понятия, name – наименование понятия, description – описание понятия, ontology_id – идентификатор модели онтологии.	Изменение понятия.
removeConcept	id – идентификатор понятия.	Удаление понятия.
addAttribute	name – наименование атрибута, type – тип атрибута, value – значение атрибута, description – описание атрибута, concept_id – идентификатор понятия.	Добавление нового атрибута (свойства) понятия.
updateAttribute	id – идентификатор атрибута, name – наименование атрибута, type – тип атрибута, value – значение атрибута, description – описание атрибута.	Изменение атрибута (свойства) понятия.
removeAttribute	id – идентификатор атрибута.	Удаление атрибута (свойства) понятия.
getRelation	id – идентификатор связи.	Получение причинно-следственной связи между двумя понятиями по идентификатору.

addRelation	left_concept_id – идентификатор левого понятия связи, right_concept_id – идентификатор правого понятия связи, name – наименование связи, description – описание связи, obligation – обязательность связи, ontology_id – идентификатор модели онтологии.	Добавление нового причинно-следственного отношения между двумя понятиями.
updateRelation	id – идентификатор связи, left_concept_id – идентификатор левого понятия связи, right_concept_id – идентификатор правого понятия связи, name – наименование связи, description – описание связи, obligation – обязательность связи.	Изменение причинно-следственного отношения между двумя понятиями.
removeRelation	id – идентификатор связи.	Удаление причинно-следственного отношения между двумя понятиями.
getAllConcepts	ontology_id – идентификатор модели онтологии.	Получение списка всех понятий у данной модели онтологии.
getConcept	id – идентификатор понятия.	Получение понятия по идентификатору.
getAllAttributes	concept_id – идентификатор понятия.	Получение всех свойств понятия.
getAttribute	id – идентификатор свойства.	Получение свойства понятия по идентификатору.
getAllRelations	ontology_id – идентификатор модели онтологии.	Получение всех причинно-следственных связей между понятиями у данной модели онтологии.

Таблица В.2. Методы взаимодействия с моделью продукций

Название	Сигнатура	Описание
createProductionModel	name – наименование модели продукций, description – описание модели продукций, xml_file – файл концептуальной модели в формате XML.	Создание модели продукций на основе импорта концептуальной модели.
generateCLIPSCode	file_name – наименование файла БЗ, production_model_name – наименование модели продукций.	Генерация кода БЗ на ЯПЗ CLIPS, на основе модели продукций.
addFactTemplate	name – наименование шаблона факта, description – описание шаблона факта, production_model_id – идентификатор модели продукций.	Добавление нового шаблона факта.
updateFactTemplate	id – идентификатор шаблона факта, name – наименование шаблона факта, description – описание шаблона факта.	Изменение шаблона факта.
removeFactTemplate	id – идентификатор шаблона факта.	Удаление шаблона факта.
addFactTemplateSlot	name – наименование слота шаблона факта, datatype_id – идентификатор типа данных, default_value – значение по умолчанию, description – описание слота шаблона факта, fact_template_id – идентификатор шаблона факта.	Добавление нового слота шаблона факта.
updateFactTemplateSlot	id – идентификатор слота шаблона факта, name – наименование слота шаблона факта, datatype_id – идентификатор типа данных, default_value – значение по умолчанию, description – описание слота шаблона факта.	Изменение слота шаблона факта.
removeFactTemplateSlot	id – идентификатор слота шаблона факта.	Удаление слота шаблона факта.

addRuleTemplate	name – наименование шаблона правила, description – описание шаблона правила, salience – значимость шаблона правила, condition_id – идентификатор условия (шаблон факта), operator – оператор условия, action_id – идентификатор действия (шаблон факта), function – функция (команда) действия, production_model_id – идентификатор модели продукций.	Добавление нового шаблона правила.
updateRuleTemplate	id – идентификатор шаблона правила, name – наименование шаблона правила, description – описание шаблона правила, salience – значимость шаблона правила, condition_id – идентификатор условия (шаблон факта), operator – оператор условия, action_id – идентификатор действия (шаблон факта), function – функция (команда) действия.	Изменение шаблона правила.
removeRuleTemplate	id – идентификатор шаблона правила.	Удаление шаблона правила.
addFact	name – наименование факта, description – описание факта, initial – указание начального факта, certainty_factor – коэффициент уверенности, fact_template_id – идентификатор шаблона факта, production_model_id – идентификатор модели продукций.	Добавление нового факта.
updateFact	id – идентификатор факта, name – наименование факта, description – описание факта, initial – указание начального факта, certainty_factor – коэффициент уверенности.	Изменение факта.
removeFact	id – идентификатор факта.	Удаление факта.

addFactSlot	name – наименование слота факта, datatype_id – идентификатор типа данных, value – значение, description – описание слота факта, fact_ id – идентификатор факта.	Добавление нового слота факта.
updateFactSlot	id – идентификатор слота факта, name – наименование слота факта, datatype_id – идентификатор типа данных, value – значение, description – описание слота факта.	Изменение слота факта.
removeFactSlot	id – идентификатор слота факта.	Удаление слота факта.
addRule	name – наименование правила, description – описание правила, salience – значимость правила, certainty_factor – коэффициент уверенности, condition_id – идентификатор условия (факт), operator – оператор условия, action_id – идентификатор действия (факт), function – функция (команда) действия, rule_template_id – идентификатор шаблона правила, production_model_id – идентификатор модели продукций.	Добавление нового правила.
updateRule	id – идентификатор правила, name – наименование правила, description – описание правила, salience – значимость правила, certainty_factor – коэффициент уверенности, condition_id – идентификатор условия (факт), operator – оператор условия, action_id – идентификатор действия (факт), function – функция (команда) действия.	Изменение правила.
removeRule	id – идентификатор правила.	Удаление правила.
getAllFactTemplates	production_model_id – идентификатор модели продукций.	Получение списка всех шаблонов фактов у данной модели продукций.

getFactTemplate	id – идентификатор шаблона факта.	Получение шаблона факта по идентификатору.
getAllFactTemplateSlots	fact_template_id – идентификатор шаблона факта.	Получение списка всех слотов шаблона факта.
getFactTemplateSlot	id – идентификатор слота шаблона факта.	Получение слота шаблона факта по идентификатору.
getAllRuleTemplates	production_model_id – идентификатор модели продукций.	Получение списка всех шаблонов правил у данной модели продукций.
getRuleTemplate	id – идентификатор шаблона правила.	Получение шаблона правила по идентификатору.
getAllFacts	production_model_id – идентификатор модели продукций.	Получение списка всех фактов у данной модели продукций.
getFact	id – идентификатор факта.	Получение факта по идентификатору.
getAllFactSlots	fact_id – идентификатор факта.	Получение списка всех слотов факта.
getFactSlot	id – идентификатор слота факта.	Получение слота факта по идентификатору.
getAllRules	production_model_id – идентификатор модели продукций.	Получение списка всех правил у данной модели продукций.
getRule	id – идентификатор правила.	Получение правила по идентификатору.

Приложение Г. Описание модели трансформации диаграмм классов UML в модель продукций

С целью описания установленных соответствий между элементами метамоделей исходной концептуальной модели в форме диаграммы классов UML и целевой модели представления знаний (модель продукций) представим полный листинг сгенерированного кода уточненной (полной) модели трансформации на TMRL.

При этом выделенным шрифтом отмечены:

- установленные связи (связь по идентификатору R_{ID}^{cm}) в исходной метамодели диаграммы классов UML с использованием графического редактора метамоделей (см. Рисунок 24);
- сложные правила определения, установленные в текстовом редакторе кода TMRL вручную.

В Листинге Г.1 представлен TMRL-код сформированной модели трансформации.

Листинг Г.1. Модель трансформации диаграмм классов UML в модель продукций на TMRL

```
Source Meta-Model Метамодель концептуальных моделей XMI UML {
     Elements [
          XMI attributes (xmi.version, timestamp),
          XMI.header,
          XMI.documentation,
          XMI.exporter,
          XMI.exporterVersion,
          XMI.metamodel attributes (xmi.name, xmi.version),
          XMI.content,
          Model attributes (xmi.id, name, visibility,
     isSpecification, isRoot, isLeaf, isAbstract),
```

```
     Namespace.ownedElement,
     Association attributes (xmi.id, name, visibility,
isSpecification, isRoot, isLeaf, isAbstract),
     Association.connection,
     AssociationEnd attributes (xmi.id, name, visibility,
isSpecification, isNavigable, ordering, aggregation,
targetScope, changeability, type),
     AssociationEnd.multiplicity,
     Multiplicity,
     Multiplicity.range,
     MultiplicityRange attributes (xmi.id, lower, upper),
     Stereotype attributes (xmi.id, name, visibility,
isSpecification, isRoot, isLeaf, isAbstract, icon, baseClass,
extendedElement),
     Class attributes (xmi.id, name, visibility,
isSpecification, isRoot, isLeaf, isAbstract, isActive,
namespace),
     Attribute attributes (xmi.id, visibility, isSpecification,
ownerScope, changeability, targetScope, type, name),
     ModelElement.name attributes (name, description),
     StructuralFeature,
     Classifier.feature,
     StructuralFeature.multiplicity,
     Attribute.initialValue,
     Expression attributes (language, body),
     DataType attributes (xmi.id, name, visibility,
isSpecification, isRoot, isLeaf, isAbstract),
     Expression.body,
     TaggedValue attributes (xmi.id, tag, value, modelElement),
     Diagram attributes (xmi.id, name, toolName, diagramType,
style, owner),
     Diagram.element,
     DiagramElement attributes (xmi.id, geometry, style,
subject)
]
```

Relationships [

XMI is associated with XMI.header,

XMI.header is associated with XMI.documentation,

XMI.documentation is associated with XMI.exporter,

XMI.documentation is associated with XMI.exporterVersion,

XMI.header is associated with XMI.metamodel,

XMI is associated with XMI.content,

XMI.content is associated with Model,

Model is associated with Namespace.ownedElement,

Namespace.ownedElement is associated with Association,

Association is associated with Association.connection,

Association.connection is associated with AssociationEnd,

AssociationEnd is associated with AssociationEnd.multiplicity,

AssociationEnd.multiplicity is associated with Multiplicity,

Multiplicity is associated with Multiplicity.range,

Multiplicity.range is associated with MultiplicityRange,

Namespace.ownedElement is associated with Stereotype,

Namespace.ownedElement is associated with Class,

Class is associated with Attribute,

Attribute is associated with ModelElement.name,

Attribute is associated with StructuralFeature,

StructuralFeature is associated with ModelElement.name,

Class is associated with ModelElement.name,

Class is associated with Classifier.feature,

Classifier.feature is associated with Attribute,

Attribute is associated with StructuralFeature.multiplicity,

StructuralFeature.multiplicity is associated with Multiplicity,

Attribute is associated with Attribute.initialValue,

Attribute.initialValue is associated with Expression,

Namespace.ownedElement is associated with DataType,

Expression is associated with Expression.body,

```
          Classifier.feature is associated with
     StructuralFeature.multiplicity,
          XMI.content is associated with TaggedValue,
          XMI.content is associated with Diagram,
          Diagram is associated with Diagram.element,
          Diagram.element is associated with DiagramElement,
          DataType (xmi.id) is Attribute (type),
          Class (xmi.id) is AssociationEnd (type),
          Association (xmi.id) is Stereotype (extendedElement)
     ]
}
Target Meta-Model Метамодель продукций {
     Elements [
          ProductionModel attributes (name, description),
          DataType attributes (name, description),
          FactTemplate attributes (name, description),
          FactTemplateSlot attributes (name, defaultValue,
     description),
          Fact attributes (name, initial, certaintyFactor,
     description),
          FactSlot attributes (name, value, description),
          RuleTemplate attributes (name, salience, description),
          RuleTemplateCondition attributes (operator),
          RuleTemplateAction attributes (function),
          Rule attributes (name, certaintyFactor, salience,
     description),
          RuleCondition attributes (operator),
          RuleAction attributes (function)
     ]
     Relationships [
          ProductionModel is associated with FactTemplate,
          ProductionModel is associated with Fact,
          ProductionModel is associated with RuleTemplate,
          ProductionModel is associated with Rule,
          FactTemplate is associated with FactTemplateSlot,
```

```
            DataType is associated with FactTemplateSlot,
            Fact is associated with FactSlot,
            DataType is associated with FactSlot,
            FactTemplate is associated with Fact,
            RuleTemplate is associated with Rule,
            RuleTemplate is associated with RuleTemplateCondition,
            RuleTemplate is associated with RuleTemplateAction,
            FactTemplate is associated with RuleTemplateCondition,
            FactTemplate is associated with RuleTemplateAction,
            Rule is associated with RuleCondition,
            Rule is associated with RuleAction,
            Fact is associated with RuleCondition,
            Fact is associated with RuleAction
      ]
}
Transformation Метамодель концептуальных моделей XMI UML to
Метамодель продукций {
      Rule DataType to DataType priority 1 [
            DataType(name) is DataType(name)
      ]
      Rule (Class, ModelElement.name) to Fact priority 2 [
            Fact(name) is Class(name) or ModelElement.name(name)
            Fact(description) is ModelElement.name(description)
      ]
      Rule (Expression, Attribute, ModelElement.name) to FactSlot
priority 3 [
            FactSlot(value) is Expression(body)
            FactSlot(name) is Attribute(name) or
      ModelElement.name(name)
            FactSlot(description) is ModelElement.name(description)
      ]
      Rule (Association, ModelElement.name) to Rule priority 4 [
            Rule(name) is Association(name) or ModelElement.name(name)
            Rule(description) is ModelElement.name(description)
      ]
```

```
    Rule (AssociationEnd, MultiplicityRange) to RuleCondition
priority 5 [
          RuleCondition(name) is AssociationEnd
          RuleCondition(operator) is "AND" [
               if (MultiplicityRange(lower) is "1") and
          (MultiplicityRange(upper) is "-1")
          ] or "OR" [
               if (MultiplicityRange(lower) is "0") and
          (MultiplicityRange(upper) is "1")
          ] or "NOT" [
               if (MultiplicityRange(lower) is "0") and
          (MultiplicityRange(upper) is "0")
          ]
     ]
     Rule (AssociationEnd, Stereotype) to RuleAction priority 6 [
          RuleAction(name) is AssociationEnd
          RuleAction(function) is Stereotype(name)
     ]
}
```

Приложение Д. Листинг программы для анализа MDL-файлов на Delphi (фрагмент)

```
    //описание структуры продукционной модели
    TTemplate = class    //шаблон
       ID : String;
       Name : String;     //наименование
       ShortName : String;     //наименование - latin
       Description  : String;
       Slots : TList;     //слоты
       DrawParams : TStringList;
       RVMLImage : TImage;
       PackageName: string; //package name
       RootPackageName: string; //root package name
       procedure Init;
       destructor Destroy; override;
     private
       { Private declarations }
     public
       { Public declarations }
     Function GetSlotByName(Sn:String):Integer;
    end;
    TSlot = class    //слот
      Name : String;     //наименование
      ShortName : String;     //наименование - latin
      Description  : String;
      Value : String;     //значение
      DataType : String;   //тип данных: числовой или строковый
      Constraint : String; //ограничение на значение слота,
используется только в правилах
     private
       { Private declarations }
     public
```

```
        { Public declarations }
        function isFunct(F:TList):Integer;
      end;
      TFunct = class //function
        ID  : String;
        Name : String;     //наименование
        ShortName : String;     //наименование - latin
        Description  : String; //
        DataType : String;
        Body : String;
        Args : TList;  //arguments
        procedure Init;
        destructor Destroy; override;
      end;
      TFact = class    //факт
        ID : String;
        Name : String;     //наименование
        ShortName : String;     //наименование - latin
        Slots : TList;     //слоты
        Mode  :  String;   //режим - упорядоченный или
неупорядоченный
        DrawParams : TStringList;
        RVMLImage : TImage;
        PackageName: string; //package name
        RootPackageName: string; //root package name
        procedure Init;
        destructor Destroy; override;
       private
        { Private declarations }
       public
        { Public declarations }
       Function GetSlotByName(Sn:String):Integer;
      end;
      TRule = class    //правило
        ID : String;
```

```
  Name : String;     //наименование
  ShortName : String;     //наименование - latin
  Description  : String;
  Salience : String;     //важность
  Conditions : TList;     //условие
  Actions : TList;        //действие
  DrawnObjects : TList;
  RVMLImage : TImage;
  PackageName: string; //package name
  RootPackageName: string; //root package name
  procedure Init;
  destructor Destroy; override;
 private
   { Private declarations }
 public
   { Public declarations }
 Function GetRUID:string;
end;
TCondition = class    //правило
  Operator: String;  //оператор условия
  Fact : TFact;    //факт
 // destructor Destroy; override;
 private
   { Private declarations }
 public
   { Public declarations }
end;
 TRAction = class    //действие
  Operator: String;  //оператор действия
  Fact : TFact;    //факт
  //procedure Init;
  //destructor Destroy; override;
 private
   { Private declarations }
 public
```

```
    { Public declarations }
  function GetFromStr(s:String; R:TRule):Integer;
  Function GetCLIPSOperator:String;
 end;
  TKnowledgeBase = class //база знаний
   ID : string;
   Name : String;
   ShortName : String;     //наименование - latin
   Kind : Integer;  //0 - rule base; 1 - case base
   Description  : String; //
   FileName :  String;
   Vars : TList;  //variables
   Rules : TList;       //правила
   GRules : TList; //generalized rules
   Facts : TList;       //факты
   Templates : TList;   //шаблоны
   Functions  : TList;
   Tasks : TList;
   FScales : TStringList;  //fuzzy scales for slot values
   TempPackageList : TStringList;
   FactPackageList : TStringList;
   RulePackageList : TStringList;
   GRulePackageList : TStringList;
   CErrors : TStringList;
   procedure Init;
   destructor Destroy; override;
  private
    { Private declarations }
  public
    { Public declarations }
…
 end;
 TGRule = class    //generalised правило
   ID : String;
   Name : String;     //наименование
```

```
        ShortName : String;     //наименование - latin
        Description  : String;
        Conditions : TList;     //условие
        Actions : TList;        //действие
        DrawnObjects : TList;
        procedure Init;
        destructor Destroy; override;
        Function GetRUID:string;
      private
        { Private declarations }
      public
     end;
    ….
    //-------------------------------------------------------------
---------------
    // функция анализа файла Rational Rose
    Function TKnowledgeBase.LoadFromMDLFile(fName:String):Integer;
    var
      List: TStringList;
      AssociationList: TStringList;
      //nfile: string;
      i, j, r, k, position, position1: Integer;
      s, word: string;
      FS: TFileStream;
      ObjectClass: Boolean;//Если TRUE значит мы нашли класс
      ObjectAssociation: Boolean;//Если TRUE значит мы нашли связь
      FirstObjectAssociation: Boolean;//Существует ли левый объект
связи
      RuleInKB: Boolean;//Если True значит правило уже добавленно в
базу знаний
      Template, tMP, tMP1: TTemplate;
      Slot: TSlot;
      Rule, Rule1: TGRule;
    begin
     Vars.Clear;
```

```
      Rules.Clear;
      GRules.Clear;
      Facts.Clear;
      Templates.Clear;
      Functions.Clear;
       RuleInKB:=False;
       FirstObjectAssociation:=False;
       FS := TFileStream.Create(fName, fmOpenRead);
       List := TStringList.Create;
       List.LoadFromStream(FS);
       AssociationList := TStringList.Create;
       for i:=0 to List.count-1 do
         begin
           //Поиск классов
           s:=List.Strings[i];
           position := AnsiPos('object Class ', s);
           //Если нашли класс, то берем его имя
           if position <> 0 then
             begin
               ObjectClass:=True;
               Template:=TTemplate.Create;
               Template.Init;

Template.Name:=Copy(s,Pos('"',s)+1,LastDelimiter('"',s)-1-
Pos('"',s));

Template.ShortName:=Translit.Trans(Template.Name,Translit.FL);
               Template.ID:=NewID('T');
               Templates.Add(Template);
             end;
           //Дальше проверяем есть ли у класса атрибуты
           position := AnsiPos('(object ClassAttribute', s);
           if (position <> 0) and (ObjectClass = True) then
             begin
               Slot:=TSlot.Create;
```

```
              Slot.Name:=Copy(s,Pos('"',s)+1,LastDelimiter('"',s)-
1-Pos('"',s));

Slot.ShortName:=Translit.Trans(Slot.Name,Translit.FL);
              Slot.DataType:='String';
              Template.Slots.Add(Slot);
            end;
          //Ищем закрытие класса, будет значить, что класс закрыт и
больше нет атрибутов
          position := AnsiPos('))))', s);
          if (position <> 0) and (ObjectClass = True) then
            begin
              ObjectClass:=False;
              ObjectAssociation:=False;
            end;
          //Теперь ищем связи
          position := AnsiPos('object Association ', s);
          if position <> 0 then
            begin
              ObjectAssociation:=True;
            end;
          //Сторим StringList со связями
          position := AnsiPos('supplier ', s);
          position1 := AnsiPos('Logical View', s);
          if (ObjectAssociation = True) and (position <> 0) and
(position1 <> 0) then
            begin
              position1 := LastDelimiter('"',s);
              position := AnsiPos('::', s);
              position1 := position1-position-2;
              word:=Copy(s,position+2,position1);
              //Если False значит это первое значение связи
              if FirstObjectAssociation = False then
              begin
                FirstObjectAssociation:=True;
```

```
                for j:=0 to Templates.Count-1 do
                begin
                  tMP:=Template.Create;
                  tMP:=Templates[j];
                  //Если tMP.Name=word значит мы нашли совпадение
НАИМЕНОВАНИЕ ШАБЛОНА в связи и НАИМЕНОВАНИЕ ШАБЛОНА В БЗ
                  if tMP.Name=word then
                  begin
                    //KnowledgeBase.GRules.Count>0 значит правила
уже есть и надо проверять
                    //есть ли правила с такими же действиями, если
да, то надо их объеденять
                    for r:=0 to GRules.Count-1 do
                    begin
                      Rule:=TGRule.Create;
                      Rule.Init;
                      Rule:=GRules[r];
                      //Rule.Actions.Count>0 значит действия уже
есть и надо проверять
                      //есть ли действия с такими же шаблонами,
если да, то надо их заменять
                      for k:=0 to Rule.Actions.Count-1 do
                      begin
                        tMP1:=TTemplate.Create;
                        tMP1.Init;
                        tMP1:=Rule.Actions[k];
                        if tMP1.Name=tMP.Name then
                        begin
                          Rule:=Rule;
                          RuleInKB:=True;
                        end;
                      end;
                    end;
                    if RuleInKB=False then
                    begin
```

```
                    Rule:=TGRule.Create;
                    Rule.Init;
                    Rule.Name:='NewRule'+IntToStr(i);
    //                  Rule.Name:=Rule.GetRUID;

Rule.ShortName:=Translit.Trans(Rule.Name,Translit.FL);
                    Rule.ID:=NewID('G');
                    Rule.Actions.Add(tMP);
                  end;
                end;
              end;
            end
            else
              begin
                FirstObjectAssociation:=False;
                for j:=0 to Templates.Count-1 do
                  begin
                    tMP:=Template.Create;
                    tMP:=Templates[j];
                    if tMP.Name=word then
                      begin
                        Rule.Conditions.Add(tMP);
                        Break;
                      end;
                  end;
                if RuleInKB=False then
                begin
                  GRules.Add(Rule);
                end;
                RuleInKB:=False;
              end;
          end;
        end;
    //    Result:=KnowledgeBase;
      for i := 0 to GRules.Count-1 do
```

```
        try

TGRule(GRules.Items[i]).Name:=TGRule(GRules.Items[i]).GetRUID;
         TGRule(GRules.Items[i]).ShortName:=Translit.Trans(
          TGRule(GRules.Items[i]).Name,Translit.FL);
        except
        end;
         FS.Free;
     end;
```

Приложение Е. Описание модели трансформации концепт-карт XTM в модель продукций

С целью описания установленных соответствий между элементами метамоделей исходной концептуальной модели в форме концепт-карт XTM и целевой модели представления знаний (модель продукций) представим полный листинг сгенерированного кода уточненной (полной) модели трансформации на TMRL. При этом выделенным шрифтом отмечены сложные правила определения, установленные в текстовом редакторе кода TMRL вручную.

В Листинге Е.1 представлен TMRL-код сформированной модели трансформации.

Листинг Е.1. Модель трансформации концепт-карт XTM в модель продукций на TMRL

```
Source Meta-Model Метамодель концепт-карт CmapTools (XTM) {
     Elements [
          topicMap attributes (id),
          topic attributes (id),
          instanceOf,
          subjectIndicatorRef attributes (type, href),
          baseName,
          baseNameString,
          association attributes (id),
          topicRef attributes (type, href),
          member,
          roleSpec
     ]
     Relationships [
          topicMap is associated with topic,
          topic is associated with instanceOf,
          instanceOf is associated with subjectIndicatorRef,
          topic is associated with baseName,
```

```
            baseName is associated with baseNameString,
            topicMap is associated with association,
            association is associated with instanceOf,
            instanceOf is associated with topicRef,
            association is associated with member,
            member is associated with roleSpec,
            roleSpec is associated with subjectIndicatorRef,
            member is associated with topicRef
      ]
}
Target Meta-Model Метамодель продукций {
      Elements [
            ProductionModel attributes (name, description),
            DataType attributes (name, description),
            FactTemplate attributes (name, description),
            FactTemplateSlot attributes (name, defaultValue,
      description),
            Fact attributes (name, initial, certaintyFactor,
      description),
            FactSlot attributes (name, value, description),
            RuleTemplate attributes (name, salience, description),
            RuleTemplateCondition attributes (operator),
            RuleTemplateAction attributes (function),
            Rule attributes (name, certaintyFactor, salience,
      description),
            RuleCondition attributes (operator),
            RuleAction attributes (function)
      ]
      Relationships [
            ProductionModel is associated with FactTemplate,
            ProductionModel is associated with Fact,
            ProductionModel is associated with RuleTemplate,
            ProductionModel is associated with Rule,
            FactTemplate is associated with FactTemplateSlot,
            DataType is associated with FactTemplateSlot,
```

```
          Fact is associated with FactSlot,
          DataType is associated with FactSlot,
          FactTemplate is associated with Fact,
          RuleTemplate is associated with Rule,
          RuleTemplate is associated with RuleTemplateCondition,
          RuleTemplate is associated with RuleTemplateAction,
          FactTemplate is associated with RuleTemplateCondition,
          FactTemplate is associated with RuleTemplateAction,
          Rule is associated with RuleCondition,
          Rule is associated with RuleAction,
          Fact is associated with RuleCondition,
          Fact is associated with RuleAction
     ]
}
Transformation Метамодель концепт-карт CmapTools (XTM) to Метамодель
продукций {
     Rule (topic, baseNameString) to Fact priority 1 [
          Fact(name) is topic or baseNameString
     ]
     Rule (topic, baseNameString) to Rule priority 2 [
          Rule(name) is topic or baseNameString
     ]
     Rule (topic, baseNameString) to FactSlot priority 3 [
          FactSlot(name) is topic or baseNameString
     ]
     Rule (member, topicRef) to RuleCondition priority 4 [
          RuleCondition(name) is member or topicRef
          RuleCondition(operator) is “AND”
     ]
     Rule (member, topicRef) to RuleAction priority 5 [
          RuleAction(name) is member or topicRef
          RuleCondition(function) is “assert”

     ]
}
```

Приложение Ж. Листинг программы для анализа CXL-файлов на Delphi (фрагмент)

```
unit Unit1;
interface
Function DllInfo:WideString; stdcall;
Function Execute(FN:ShortString;gF:Integer):TObject; stdcall;
Function About:WideString; stdcall;
implementation
uses UPKBClass, uTrans, Windows, SysUtils, Classes;
//---------------------------------------------------------------
Function About:WideString; stdcall;
begin
 Result:='CmapTools (cxl) importer';
end;
//---------------------------------------------------------------
Function DllInfo:WideString; stdcall;
begin
 Result:='1.2017';
end;
//---------------------------------------------------------------
Function Execute(FN:ShortString;gF:Integer):TObject;
//gF - flag for complexing the grules
//0 - none (just a simple grules)
//1 - group lhs
//2 - group rhs
//3 - group lhs and rhs
var
 i,j,k,k2 : Integer;
 tmTs,tmTs1,Concepts,Links,Links1 : TStringList;
 captital_letters : ShortString;
 tmT : TTemplate;
 tmSl : TSlot;
```

```
      tmGR : TGRule;
     begin
       tmTs:=TStringList.Create;
       tmTs1:=TStringList.Create;
       Concepts:=TStringList.Create;
       Links:=TStringList.Create;
       Links1:=TStringList.Create;
     captital_letters:='"`АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫБЭЮЯABCDEFGHI
JKLMNOPQRSTUVWXYZ'+#39;
       //load and preprocessing
       tmTs.LoadFromFile(FN);
       tmTs.Text:=StringReplace(tmTs.Text,'<concept
id',#10+'<concept id',[rfReplaceAll]);
       tmTs.Text:=StringReplace(tmTs.Text,'<connection
id',#10+'<connection id',[rfReplaceAll]);
       tmTs.Text:=StringReplace(tmTs.Text,'<linking-phrase
id',#10+'<linking-phrase id',[rfReplaceAll]);

tmTs.Text:=StringReplace(tmTs.Text,'/>',#10+'/>',[rfReplaceAll]);
       //processing
       for i:=0 to tmTs.Count-1 do
        begin

tmTs1.Text:=StringReplace(Trim(tmTs.Strings[i]),'"',#10,[rfReplaceAl
l]);
         if tmTs1.Count>3 then //get concepts
          begin
          if tmTs1.Strings[0]='<concept id=' then
           begin
            Concepts.Add(
             Trim(tmTs1.Strings[1])+'='+
             Trim(tmTs1.Strings[3])
             );
           end;
          if tmTs1.Strings[0]='<linking-phrase id=' then
```

```
    begin
     Links1.Add(
      Trim(tmTs1.Strings[1])
      );
    end;
   if tmTs1.Strings[0]='<connection id=' then
    begin
     Links.Add(
      Trim(tmTs1.Strings[3])+'='+
      Trim(tmTs1.Strings[5])
      )
    end;
   end;
  end;
//process linking phrase
for i:=0 to Links1.Count-1 do
 begin
  j:=0;
  while j<Links.Count do  //look for lhs
   begin
//    for j:=0 to Links.Count-1 do
     if Links1.Strings[i]=Links.ValueFromIndex[j] then
      begin
       //look for rhs
       k:=Links.IndexOfName(Links1.Strings[i]);
       if k<>-1 then
        begin
         Links.ValueFromIndex[j]:=Links.ValueFromIndex[k];
         Links.Delete(k);
        end;
      end;
    Inc(j);
   end;
 end;
Translit:=TTranslit.Create;
```

```
        Translit.Init;
        TKnowledgeBase(Result):=TKnowledgeBase.Create;
        TKnowledgeBase(Result).Init;
        //first stage - create templates
        for i:=0 to Concepts.Count-1 do
         begin
           //check the first letter
          if Concepts.ValueFromIndex[i]<>'' then
            if pos(Concepts.ValueFromIndex[i][1],captital_letters)>0
then
             begin //it is a template
              tmT:=TTemplate.Create;
                 tmT.Init;

tmT.Name:=StringReplace(Concepts.ValueFromIndex[i],'''',''',[rfReplac
eAll]);

tmT.Name:=StringReplace(tmT.Name,'"','',[rfReplaceAll]);

tmT.Name:=StringReplace(tmT.Name,'`','',[rfReplaceAll]);
              tmT.ShortName:=Translit.Trans(tmT.Name,Translit.FL);
              tmT.ID:=Concepts.Names[i];
              TKnowledgeBase(Result).Templates.Add(tmT);
             end
         end;

       //second stage - create properties
        for i:=0 to Concepts.Count-1 do
         begin
           //check the first letter
          if Concepts.ValueFromIndex[i]<>'' then
           begin
            if pos(Concepts.ValueFromIndex[i][1],captital_letters)=0
then
             begin  //it is a property
```

```
              //get the linked template
              //look out in rhs
              for j:=0 to Links.Count-1 do
               if Concepts.Names[i]=Links.ValueFromIndex[j] then
                begin

k:=TKnowledgeBase(Result).IndexOfTemplateByID(Links.Names[j]);
                 if k>-1 then
                  begin
                   tmSl:=TSlot.Create;
                   tmSl.Name:=Concepts.ValueFromIndex[i];
                   tmSl.DataType:='String';

tmSl.ShortName:=Translit.Trans(tmSl.Name,Translit.FL);

TTemplate(TKnowledgeBase(Result).Templates.Items[k]).Slots.Add(tmSl)
;
                  end;
                end;
              //look out in lhs
              j:=Links.IndexOfName(Concepts.Names[i]);
              if j>-1 then
               begin  //add the propery as a slot

k:=TKnowledgeBase(Result).IndexOfTemplateByID(Links.ValueFromIndex[j
]);
                if k>-1 then
                 begin
                  tmSl:=TSlot.Create;
                  tmSl.Name:=Concepts.ValueFromIndex[i];
                  tmSl.DataType:='String';

tmSl.ShortName:=Translit.Trans(tmSl.Name,Translit.FL);
```

```
TTemplate(TKnowledgeBase(Result).Templates.Items[k]).Slots.Add(tmSl)
;
                end;
              end;
            end;
          end;
        end;
       //third stage - create general rules
       for i:=Links.Count-1 downto 0 do
        begin

j:=TKnowledgeBase(Result).IndexOfTemplateByID(Links.ValueFromIndex[i
]);

k:=TKnowledgeBase(Result).IndexOfTemplateByID(Links.Names[i]);
         if (j=-1)or(k=-1) then Links.Delete(i);  //delete
properties links
        end;
       case gF of
       0:begin //simple
         for i:=0 to Links.Count-1 do
          begin

j:=TKnowledgeBase(Result).IndexOfTemplateByID(Links.ValueFromIndex[i
]);

k:=TKnowledgeBase(Result).IndexOfTemplateByID(Links.Names[i]);
           if (j<>-1)and(k<>-1) then
            begin  //create grule
             tmGR:=TGRule.Create;
             tmGR.Init;

tmGR.ID:=IntToStr(TKnowledgeBase(Result).GRules.Count+1);
             tmGR.PackageName:='';
```

```
              tmGR.RootPackageName:='';
              tmGR.Name:='';
              //condition
              tmT:=TTemplate.Create;
              tmT.Init;

tmT.MakeACloneFrom(1,TTemplate(TKnowledgeBase(Result).Templates.Item
s[k]));
              tmGR.Conditions.Add(tmT);
              //action
              tmT:=TTemplate.Create;
              tmT.Init;

tmT.MakeACloneFrom(1,TTemplate(TKnowledgeBase(Result).Templates.Item
s[j]));
              tmGR.Actions.Add(tmT);
              TKnowledgeBase(Result).GRules.Add(tmGR);
             end;
           end;
         end;  //end 0
         1:begin //lhs
          for i:=0 to Links.Count-1 do
           begin

j:=TKnowledgeBase(Result).IndexOfTemplateByID(Links.ValueFromIndex[i
]); //rhs

k:=TKnowledgeBase(Result).IndexOfTemplateByID(Links.Names[i]); //lhs
            if (j<>-1)and(k<>-1) then
             begin  //create grule
              //look for grule with this lhs
              k2:=TKnowledgeBase(Result).IndexOfGRuleWithTemplate(

TTemplate(TKnowledgeBase(Result).Templates.Items[k]).ShortName
               ,0);
```

```
              if k2=-1 then
               begin
                tmGR:=TGRule.Create;
                tmGR.Init;

tmGR.ID:=IntToStr(TKnowledgeBase(Result).GRules.Count+1);
                tmGR.PackageName:='';
                tmGR.RootPackageName:='';
                //condition
                tmT:=TTemplate.Create;
                tmT.Init;

tmT.MakeACloneFrom(1,TTemplate(TKnowledgeBase(Result).Templates.Item
s[k]));
                tmGR.Conditions.Add(tmT);
                //action
                tmT:=TTemplate.Create;
                tmT.Init;

tmT.MakeACloneFrom(1,TTemplate(TKnowledgeBase(Result).Templates.Item
s[j]));
                tmGR.Actions.Add(tmT);
                TKnowledgeBase(Result).GRules.Add(tmGR);
              end
               else
                begin   //for rhs
                  if
TGRule(TKnowledgeBase(Result).GRules.Items[k2]).IndexOfComponentByNa
me(

1,TTemplate(TKnowledgeBase(Result).Templates.Items[j]).ShortName)=-1
then
                    begin
                      tmT:=TTemplate.Create;
                      tmT.Init;
```

```
tmT.MakeACloneFrom(1,TTemplate(TKnowledgeBase(Result).Templates.Item
s[j]));

TGRule(TKnowledgeBase(Result).GRules.Items[k2]).Actions.Add(tmT);
                    end;
                end;
              end;
            end;
           end; //end 1
          2:begin //rhs
           for i:=0 to Links.Count-1 do
           begin

j:=TKnowledgeBase(Result).IndexOfTemplateByID(Links.ValueFromIndex[i
]); //rhs

k:=TKnowledgeBase(Result).IndexOfTemplateByID(Links.Names[i]); //lhs
             if (j<>-1)and(k<>-1) then
              begin  //create grule
               //look for grule with this rhs
               k2:=TKnowledgeBase(Result).IndexOfGRuleWithTemplate(

TTemplate(TKnowledgeBase(Result).Templates.Items[j]).ShortName
                ,1);
               if k2=-1 then
                begin
                 tmGR:=TGRule.Create;
                 tmGR.Init;

tmGR.ID:=IntToStr(TKnowledgeBase(Result).GRules.Count+1);
                 tmGR.PackageName:='';
                 tmGR.RootPackageName:='';
                 //condition
                 tmT:=TTemplate.Create;
```

```
                tmT.Init;

tmT.MakeACloneFrom(1,TTemplate(TKnowledgeBase(Result).Templates.Item
s[k]));
                tmGR.Conditions.Add(tmT);
                //action
                tmT:=TTemplate.Create;
                tmT.Init;

tmT.MakeACloneFrom(1,TTemplate(TKnowledgeBase(Result).Templates.Item
s[j]));
                tmGR.Actions.Add(tmT);
                TKnowledgeBase(Result).GRules.Add(tmGR);
              end
               else
                begin   //for lhs
                  if
TGRule(TKnowledgeBase(Result).GRules.Items[k2]).IndexOfComponentByNa
me(

0,TTemplate(TKnowledgeBase(Result).Templates.Items[k]).ShortName)=-1
then
                    begin
                      tmT:=TTemplate.Create;
                      tmT.Init;

tmT.MakeACloneFrom(1,TTemplate(TKnowledgeBase(Result).Templates.Item
s[k]));

TGRule(TKnowledgeBase(Result).GRules.Items[k2]).Conditions.Add(tmT);
                    end;
                end;
              end;
            end;
          end; //end 2
```

```
        3:begin //lhs+rhs
        for i:=0 to Links.Count-1 do
          begin

j:=TKnowledgeBase(Result).IndexOfTemplateByID(Links.ValueFromIndex[i
]); //rhs

k:=TKnowledgeBase(Result).IndexOfTemplateByID(Links.Names[i]); //lhs
           if (j<>-1)and(k<>-1) then
            begin  //create grule
             //look for grule with this lhs
             k2:=TKnowledgeBase(Result).IndexOfGRuleWithTemplate(

TTemplate(TKnowledgeBase(Result).Templates.Items[k]).ShortName
              ,0);
             if k2=-1 then
              begin
               k2:=TKnowledgeBase(Result).IndexOfGRuleWithTemplate(

TTemplate(TKnowledgeBase(Result).Templates.Items[j]).ShortName
                ,1);
               if k2=-1 then
                begin
                 tmGR:=TGRule.Create;
                 tmGR.Init;

tmGR.ID:=IntToStr(TKnowledgeBase(Result).GRules.Count+1);
              tmGR.PackageName:='';
                 tmGR.RootPackageName:='';
                 //condition
                 tmT:=TTemplate.Create;
                 tmT.Init;

tmT.MakeACloneFrom(1,TTemplate(TKnowledgeBase(Result).Templates.Item
s[k]));
```

```
                tmGR.Conditions.Add(tmT);
                //action
                tmT:=TTemplate.Create;
                tmT.Init;

tmT.MakeACloneFrom(1,TTemplate(TKnowledgeBase(Result).Templates.Item
s[j]));
                tmGR.Actions.Add(tmT);
                TKnowledgeBase(Result).GRules.Add(tmGR);
               end
             else
              begin  //for lhs
                if
TGRule(TKnowledgeBase(Result).GRules.Items[k2]).IndexOfComponentByNa
me(

0,TTemplate(TKnowledgeBase(Result).Templates.Items[k]).ShortName)=-1
then
                  begin
                    tmT:=TTemplate.Create;
                    tmT.Init;

tmT.MakeACloneFrom(1,TTemplate(TKnowledgeBase(Result).Templates.Item
s[k]));

TGRule(TKnowledgeBase(Result).GRules.Items[k2]).Conditions.Add(tmT);
                  end;
              end;
            end
             else
              begin   //for rhs
                if
TGRule(TKnowledgeBase(Result).GRules.Items[k2]).IndexOfComponentByNa
me(
```

```
1,TTemplate(TKnowledgeBase(Result).Templates.Items[j]).ShortName)=-1
then
                  begin
                    tmT:=TTemplate.Create;
                    tmT.Init;

tmT.MakeACloneFrom(1,TTemplate(TKnowledgeBase(Result).Templates.Item
s[j]));

TGRule(TKnowledgeBase(Result).GRules.Items[k2]).Actions.Add(tmT);
                  end;
               end;
             end;
           end;
         end; //end 3
         end; //end case gF
        //make grules names
        for i := 0 to TKnowledgeBase(Result).GRules.Count-1 do
         begin
          TGRule(TKnowledgeBase(Result).GRules.Items[i]).Name:='';
          for j := 0 to
TGRule(TKnowledgeBase(Result).GRules.Items[i]).Conditions.Count-1 do
           begin
            if
TGRule(TKnowledgeBase(Result).GRules.Items[i]).Name<>'' then
             TGRule(TKnowledgeBase(Result).GRules.Items[i]).Name:=

TGRule(TKnowledgeBase(Result).GRules.Items[i]).Name+'+';
             TGRule(TKnowledgeBase(Result).GRules.Items[i]).Name:=
              TGRule(TKnowledgeBase(Result).GRules.Items[i]).Name+

TTemplate(TGRule(TKnowledgeBase(Result).GRules.Items[i]).Conditions.
Items[j]).Name;
           end;
```

```
          TGRule(TKnowledgeBase(Result).GRules.Items[i]).Name:=
           TGRule(TKnowledgeBase(Result).GRules.Items[i]).Name+'->';
          for j := 0 to
TGRule(TKnowledgeBase(Result).GRules.Items[i]).Actions.Count-1 do
           begin
            if j<>0 then
             TGRule(TKnowledgeBase(Result).GRules.Items[i]).Name:=

TGRule(TKnowledgeBase(Result).GRules.Items[i]).Name+'+';
              TGRule(TKnowledgeBase(Result).GRules.Items[i]).Name:=
               TGRule(TKnowledgeBase(Result).GRules.Items[i]).Name+

TTemplate(TGRule(TKnowledgeBase(Result).GRules.Items[i]).Actions.Ite
ms[j]).Name;
           end;
          TGRule(TKnowledgeBase(Result).GRules.Items[i]).ShortName:=

Translit.Trans(TGRule(TKnowledgeBase(Result).GRules.Items[i]).Name,T
ranslit.FL);
         end;
        tmTs.Free;
        tmTs1.Free;
        Concepts.Free;
        Links.Free;
      end;
      end.
```

Приложение 3. Описание модели трансформации деревьев событий в модель продукций

С целью описания установленных соответствий между элементами метамоделей исходной концептуальной модели в форме ДС и целевой модели представления знаний (модель продукций) представим полный листинг сгенерированного кода уточненной (полной) модели трансформации на TMRL. При этом выделенным шрифтом отмечены сложные правила определения, установленные в текстовом редакторе кода TMRL вручную.

В Листинге 3.1 представлен TMRL-код сформированной модели трансформации.

Листинг 3.1. Модель трансформации ДС в модель продукций на TMRL

```
Source Meta-Model Метамодель ДС {
     Elements [
          EventTree attributes (caption, Graph_type_node,
Func_type_node, can_deleted, can_move, SGmode, SGCaption, Name),
          CauseEffectRelation attributes (Graph_type_relation,
id, Name),
          Cause attributes (id, event, Name),
          Effect attributes (id, event, Name),
          InitialEvent attributes (caption, Func_type_node, id,
can_deleted, can_move, Graph_type_node, Name),
          Event attributes (Func_type_node, Graph_type_node,
can_move, can_deleted, id, caption, tag-ev, Name),
          Parameter attributes (id, caption, value)
     ]
     Relationships [
          EventTree is associated with CauseEffectRelation,
          CauseEffectRelation is associated with Cause,
          CauseEffectRelation is associated with Effect,
          EventTree is associated with InitialEvent,
```

```
                EventTree is associated with Event,
                Event is associated with Parameter,
                Event (id) is associated with Cause (event),
                Event (id) is associated with Effect (event),
                InitialEvent (id) is associated with Cause (event),
                     InitialEvent (id) is associated with Effect
(event)
          ]
     }
     Target Meta-Model Метамодель продукций {
          Elements [
                ProductionModel attributes (name, description),
                DataType attributes (name, description),
                FactTemplate attributes (name, description),
                FactTemplateSlot attributes (name, defaultValue,
description),
                Fact attributes (name, initial, certaintyFactor,
description),
                FactSlot attributes (name, value, description),
                RuleTemplate attributes (name, salience, description),
                RuleTemplateCondition attributes (operator),
                RuleTemplateAction attributes (function),
                Rule attributes (name, certaintyFactor, salience,
description),
                RuleCondition attributes (operator),
                RuleAction attributes (function)
          ]
          Relationships [
                ProductionModel is associated with FactTemplate,
                ProductionModel is associated with Fact,
                ProductionModel is associated with RuleTemplate,
                ProductionModel is associated with Rule,
                FactTemplate is associated with FactTemplateSlot,
                DataType is associated with FactTemplateSlot,
                Fact is associated with FactSlot,
```

```
          DataType is associated with FactSlot,
          FactTemplate is associated with Fact,
          RuleTemplate is associated with Rule,
          RuleTemplate is associated with RuleTemplateCondition,
          RuleTemplate is associated with RuleTemplateAction,
          FactTemplate is associated with RuleTemplateCondition,
          FactTemplate is associated with RuleTemplateAction,
          Rule is associated with RuleCondition,
          Rule is associated with RuleAction,
          Fact is associated with RuleCondition,
          Fact is associated with RuleAction
     ]
}
Transformation Метамодель ДС to Метамодель продукций {
     Rule InitialEvent to Fact priority 1 [
          Fact(name) is InitialEvent(caption)
     ]
     Rule (Parameter, Event) to Fact priority 2 [
          Fact(name) is Event(caption)
          Fact(certaintyFactor) is Parameter(value) [
               if (Parameter(caption) is “cf”)
          ]
     ]
     Rule Parameter to FactSlot priority 3 [
          FactSlot(name) is Parameter(caption)
          FactSlot(value) is Parameter(value)
     ]
     Rule CauseEffectRelation to Rule priority 4 [
          Rule(name) is CauseEffectRelation
     ]
     Rule Cause to RuleCondition priority 5 [
          RuleCondition(name) is Cause
     ]
     Rule Effect to RuleAction priority 6 [
          RuleAction(name) is Effect
```

```
            RuleCondition(function) is "assert"
        ]
    }
```

Приложение И. Листинг кода БЗ в формате CLIPS с описаним деградационного процесса коррозионной усталости

```
;*****************************************************************
;База знаний:
;*****************************************************************
;[Наименование] - Деградационные процессы аппаратов в
нефтехимии
;[Описание] - Деградационные процессы аппаратов в нефтехимии.
;********************* Шаблоны ******************************
(deftemplate incident-object ;object of the incident
    (slot cf (default "1")) ;certainty factor
    (slot caption (default "НЕТ ДАННЫХ")) ;name of the object
)
(deftemplate object-properties ;property of the object
    (slot cf (default "1")) ;certainty factor
    (slot caption) ;name of the object
    (slot caption-incident-object) ;ref to the object
)
(deftemplate material ;material of the object
    (slot cf) ;certainty factor
    (slot caption) ;наименование
    (slot type (default "СТАЛЬ")) ;вид
    (slot mechanical-prop-strength-limit) ;strength limit
    (slot mechanical-prop-yield-limit) ;свойства стойкости
    (slot resistance-prop-corrosion) ;Corrosion resistance
    (slot resistance-prop-temperature) ;temperature resistance
    (slot resistance-prop-wear) ;chemical properties
    (slot chemical-prop-alloying) ;structure
    (slot structure-prop-class) ;structural class / martensitic
/ ferritic/
)
```

```
    (deftemplate technological-heredity ;технологическая
наследственность
        (slot cf (default 1)) ;коэффициент уверенности
        (slot caption) ;наименование
    )
    (deftemplate making-defects ;дефекты изготовления
         (slot caption-technological-heredity) ;ссылка на
     технологическую наследственность
        (slot type) ;вид /микротрещины/дефекты изготовления/
        (slot cause) ;причина /сварка/
        (slot location) ;месторасположение /зона терм-го
   воздействия/
        (slot cf (default 1)) ;коэффициент уверенности
    )
    (deftemplate mechanical-stress-const ;механические нагрузки -
статические/постоянные/
         (slot stress-const-type (default "HIGH")) ;вид статических
     нагрузок/внутреннее давление/мпа//сосредоточенная нагрузка
     /мн//распределенная нагрузка/
        (slot stress-value (default 0)) ;величина нагрузок
        (slot tension-type) ;вид напряжения /растягивающие /
   сжимающие / сдвига/
        (slot tension-value) ;величина напряжения
        (slot cycle-amplitude) ;амплитуда цикла
        (slot cycle-frequency) ;частота цикла
        (slot cycle-asymmetry) ;коэффициент асимметрии цикла
        (slot cycle-average-value) ;среднее значение цикла
        (slot speed) ;скорость
        (slot speed-up) ;ускорение
        (slot max-stress-value) ;максимальное значение нагрузок
        (slot cf (default 1)) ;коэффициент уверенности
    )
    (deftemplate technological-environment ;/рабочая/
технологическая среда
```

```
        (slot contents-molecular-hydrogen (default "ACTIVE"))
    ;содержание молекулярного водорода /меньше 10% об/от 10% до 50%
    об/больше 50 % об/
        (slot ph) ;водородный показатель /меньше 0-7 /рабочая среда
    кислая, кислотность увеличивается к нулю/, = 7 /рабочая среда
    нейтральная/, больше 7-15 /рабочая среда щелочная, щелочные
    свойства увеличиваются к 15//
        (slot radiation) ;радиация /термическая / электромагнитная
    / ионизирующая/
        (slot properties-alternation) ;чередование свойств среды
  /да/нет/
        (slot environment-humidity) ;влажность среды
        (slot environment-flash) ;температура вспышки паров среды
        (slot cf (default 1)) ;коэффициент уверенности
    )
    (deftemplate heat-exchange-technological-environment
;теплообменная технологическая среда
        (slot environment-type) ;вид /оборотная вода
    /слабокоррозионная/ / химически-очищенная вода /не
    коррозионная/ / химически-очищенная вода /слабощелочная//
        (slot ph (default "НЕЙТРАЛЬНАЯ")) ;водородный показатель
        (slot cf (default 1)) ;коэффициент уверенности
    )
    (deftemplate exist-event ;наблюдаемое событие кинетики
        (slot cf) ;коэффициент уверенности
        (slot caption) ;наименование события
        (slot caption-kin) ;ссылка на кинетику
        (slot probabilityrel (default "0")) ;вероятность
  возникновения
        (slot id (default "0"))
    )
    (deftemplate exist-dam ;наблюдаемые повреждения
        (slot id-dam (default "CRACK"))
        (slot caption-dam) ;наименование повреждения
```

```
     (slot dam-type) ;тип /в частном случает тип
трещинообразования/
     (slot dam-stress-type) ;тип напряжения
     (slot dam-mesto-zarozdenia) ;место зарождения /например,
трещины/
     (slot dam-istochnik (default "CORROSION FATIGUE")) ;источник
     (slot dam-mestopolozenie) ;местоположение
     (slot dam-orientacia) ;ориентировка /например, трещины/
     (slot dam-glubina) ;глубина
     (slot dam-dlina) ;длина
     (slot dam-diametr) ;диаметр
     (slot dam-forma) ;форма в изломе
     (slot dam-kolichestvo) ;количество повреждений близких по
размеру
     (slot dam-velichina-raskritia) ;велична раскрытия
     (slot dam-tech-pered-razrusheniem) ;критерий течь перед
разрушением
     (slot dam-napravlenie) ;направление по отношению к главной
оси
     (slot caption-def)
     (slot id-def)
     (slot caption-meh)
     (slot dam-cf (default 1)) ;коэффициент уверенности
 )
 (deftemplate exist-des ;наблюдаемые разрушения
     (slot id-des (default "BRITTLE FAILURE"))
     (slot caption-des) ;наименование
     (slot des-istochnik) ;источник
     (slot des-orientacia) ;ориентировка /например, трещины/
     (slot des-napravlenie) ;направление по отношению к главной
оси
     (slot des-forma) ;форма
     (slot des-kolichestvo) ;количество
     (slot des-koncentracia-naprazheniy)
     (slot des-mestopolozenie (default "CRACK")) ;местоположение
```

```
    (slot caption-dam)
    (slot id-dam)
    (slot caption-meh)
    (slot des-cf (default 1)) ;коэффициент уверенности
 )
 (deftemplate exist-def ;наблюдаемые дефекты
    (slot id-def)
    (slot caption-def) ;наименование
    (slot caption-meh)
    (slot def-cf (default 1)) ;коэффициент уверенности
 )
 (deftemplate exist-meh ;наблюдаемая механизм
    (slot caption-meh (default "CORROSION FATIGUE"))
;наименование
    (slot meh-cf (default 1)) ;коэффициент уверенности
 )
 (deftemplate exist-kin ;наблюдаемая кинетика отказа
    (slot caption (default "KINETICS 'CORROSION FATIGUE'"))
;наименование
    (slot caption-meh (default "CORROSION FATIGUE")) ;ссылка на
механизм
    (slot cf (default 1)) ;выявления событий -
 )
 ;************************ Факты ******************************
 (deffacts initial-settings
    (mechanical-stress-const ;mechanical-stress-const
        (stress-value 0)
        (cycle-frequency "HIGH")
        (cf 1)
    )
    (technological-environment ;technological-environment
        (ph "ACTIVE")
        (properties-alternation "YES")
        (cf 1)
    )
```

```
        (incident-object ;incident-object
            (cf "1")
            (caption "PIPE INTO PIPE")
        )
        (material ;material
            (cf "1")
            (type "STEEL")
            (chemical-prop-alloying "LOW-ALLOY STEEL")
        )
    )
    ;************************* Правила ***************************
    (defrule fai-mechanism-ky-1001 "правило выявления /механизма/:
если имеются механические нагрузки высокой частоты и активная
технологическая среда с чередующимися свойствами и материал
низколегированная сталь и имеются дефекты изготовления то может
возникнуть механизм коррозионная усталость"
    (mechanical-stress-const ;механические нагрузки
    (cycle-frequency "HIGH") ;-----------
    )
    (technological-environment ;технологическая среда
    (ph "ACTIVE") ;водородный показатель
    (properties-alternation "YES") ;чередование свойств среды
/да/нет/
    )
    (material ;материал
    (type "STEEL") ;вид
    (chemical-prop-alloying "LOW-ALLOY STEEL") ;легированность
    )
    (making-defects ;дефекты изготовления
    (caption-technological-heredity ?id-th-m1) ;ссылка на
технологическую наследственность
    )
    =>
    (assert
    (exist-event ;exist-event
```

```
     (caption "MECHANISM 'CORROSION FATIGUE'") ;может возникнуть
/механизм/
     (cf "0,9") ;ку
     ))
     (assert
     (exist-meh ;exist-meh
     (caption-meh "CORROSION FATIGUE") ;может возникнуть /механизм/
     (meh-cf "0,9") ;ку
     ))
     )
     (defrule fai-kinatics-ky-1001 "правило выявления /кинетики/:
если наблюдается механизм коррозионная усталость то может возникнуть
кинетика коррозионная усталость"
     (exist-event ;exist-event
     (caption "MECHANISM 'CORROSION FATIGUE'") ;наблюдается
/механизм/
     (cf ?x)
     )
     =>
     (assert
     (exist-kin ;exist-kin
     (caption "KINETICS 'CORROSION FATIGUE'") ;может возникнуть
/кинетика/
     (caption-meh "CORROSION FATIGUE")
     (cf ?x)
     ))
     )
     (defrule fai-kinatic-events-ky-1001 "правило выявления
/параметры/события кинетики/: если наблюдается кинетика коррозионная
усталость то могут наблюдаться события: утечка рабочей среды через
сквозные трещины и полный выброс рабочей среды"
     (exist-kin ;exist-kin
     (caption "KINETICS 'CORROSION FATIGUE'") ;наблюдается
/кинетика/
     (cf ?x)
```

```
     )
     =>
     (assert
     (exist-event ;exist-event
     (caption "LEAKAGE OF THE WORKING ENVIRONMENT THROUGH THE
THROUGH CRACKS") ;может возникнуть /событие/
     (probabilityrel "1")
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
     ))
     (assert
     (exist-event ;exist-event
     (caption "COMPLETE RESET OF THE WORKING ENVIRONMENT") ;может
возникнуть /событие/
     (probabilityrel "1")
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
     ))
     )
     (defrule fai-kinatic-events-ky-1002 "правило выявления
/последующее событие кинетики/: если наблюдается кинетика
коррозионная усталость и событие утечка рабочей среды через сквозные
трещины то может наблюдаться событие образование сквозной трещины"
     (exist-kin ;exist-kin
     (caption "KINETICS 'CORROSION FATIGUE'") ;наблюдается
/кинетика/
     )
     (exist-event ;exist-event
     (caption "LEAKAGE OF THE WORKING ENVIRONMENT THROUGH THE
THROUGH CRACKS") ;/событие/
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
     )
     =>
     (assert
```

```
     (exist-event ;exist-event
     (caption "FORMATION OF THE THROUGH CRACKS") ;может возникнуть
/событие/
     (probabilityrel "1")
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
     ))
     )
     (defrule fai-kinatic-events-ky-1003 "правило выявления
/последующее событие кинетики/: если наблюдает-ся кинетика
коррозионная усталость и событие полный выброс рабочей среды то
может наблюдаться событие хрупкое разрушение"
     (exist-kin ;exist-kin
     (caption "KINETICS 'CORROSION FATIGUE'") ;наблюдается
/кинетика/
     )
     (exist-event ;exist-event
     (caption "COMPLETE RESET OF THE WORKING ENVIRONMENT")
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
     )
     =>
     (assert
     (exist-event ;exist-event
     (caption "BRITTLE FAILURE") ;может возникнуть /событие/
     (probabilityrel "1")
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
     ))
     )
     (defrule des-mechanism-ky-1001 "Описание правила: des-
mechanism-ky-1001"
     (exist-dam ;exist-dam
     (caption-dam "CRACK")
     (dam-orientacia "PERPENDICULARLY")
```

```
     (dam-napravlenie "LONGITUDINAL")
     (caption-meh "CORROSION FATIGUE")
     (dam-cf ?x)
     (id-dam ?id)
     )
     =>
     (assert
     (exist-des ;exist-des
     (des-cf ?x)
     (caption-des "MACROCRACK")
     (des-istochnik "SURFACE DAMAGE")
     (des-orientacia "PERPENDICULARLY")
     (des-napravlenie "LONGITUDINAL")
     (des-kolichestvo "SINGLE")
     (caption-meh "CORROSION FATIGUE")
     (caption-dam "CRACK")
     (id-dam ?id)
     (id-des ?id)
     ))
     (assert
     (exist-event ;exist-event
     (caption "FORMATION OF THE MACROCRACKS OF THE UNACCEPTABLE
SIZE") ;может возникнуть /событие/
     (probabilityrel "1")
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
     ))
     )
     (defrule des-mechanism-ky-1002 "Описание правила: des-
mechanism-ky-1002"
     (exist-dam ;exist-dam
     (caption-dam "CRACK")
     (dam-istochnik "CONSTRUCTIVE STRESS CONCENTRATOR FORMED BY THE
INTERSECTION OF HOLES")
     (dam-orientacia "PERPENDICULARLY")
```

```
(dam-napravlenie "LONGITUDINAL")
(dam-forma "SEMI-ELLIPTIC")
(caption-meh "CORROSION FATIGUE")
(dam-cf ?x)
(id-dam ?id)
)
(exist-event ;exist-event
(caption "FORMATION OF THE THROUGH CRACKS")
)
=>
(assert
(exist-des ;exist-des
(des-cf ?x)
(caption-des "MACROCRACK")
(des-istochnik "HOLEHOLE")
(des-orientacia "PERPENDICULARLY")
(des-napravlenie "LONGITUDINAL")
(des-kolichestvo "SINGLE")
(des-koncentracia-naprazheniy "LESS THAN 2")
(caption-meh "CORROSION FATIGUE")
(caption-dam "CRACK")
(id-dam ?id)
(id-des ?id)
))
(assert
(exist-event ;exist-event
(caption "FORMATION OF THE MACROCRACKS OF THE UNACCEPTABLE
SIZE") ;может возникнуть /событие/
(probabilityrel "1")
(caption-kin "KINETICS 'CORROSION FATIGUE'")
(cf ?x)
))
)
(defrule des-mechanism-ky-1003 "Описание правила: des-
mechanism-ky-1003"
```

```
     (exist-event ;exist-event
     (caption "BRITTLE FAILURE")
     )
     (exist-des ;exist-des
     (caption-des "BRITTLE FAILURE")
     (des-istochnik "CONSTRUCTIVE STRESS CONCENTRATOR FORMED BY THE
INTERSECTION OF HOLES")
     (des-orientacia "PERPENDICULARLY")
     (des-napravlenie "LONGITUDINAL")
     (des-forma "SEMI-ELLIPTIC")
     (caption-meh "CORROSION FATIGUE")
     (id-dam ?id)
     (id-des ?id2)
     (des-cf ?x)
     )
     =>
     (assert
     (exist-des ;exist-des
     (des-cf ?x)
     (caption-des "MACROCRACK")
     (des-istochnik "HOLE")
     (des-orientacia "PERPENDICULARLY")
     (des-napravlenie "LONGITUDINAL")
     (des-kolichestvo "SINGLE")
     (des-koncentracia-naprazheniy "LESS THAN 2")
     (caption-meh "CORROSION FATIGUE")
     (caption-dam "CRACK")
     (id-dam ?id)
     (id-des ?id2)
     ))
     (assert
     (exist-event ;exist-event
     (caption "FORMATION OF THE MACROCRACKS OF THE UNACCEPTABLE
SIZE") ;может возникнуть /событие/
     (probabilityrel "1")
```

```
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
     ))
     )
     (defrule des-kinatic-events-ky-1004 "Описание правила: des-
kinatic-events-ky-1004"
     (exist-dam ;exist-dam
     (caption-dam "CRACK")
     (dam-istochnik "SURFACE DAMAGE")
     (dam-orientacia "PERPENDICULARLY")
     (dam-napravlenie "LONGITUDINAL")
     (dam-kolichestvo "SINGLE")
     (caption-meh "CORROSION FATIGUE")
     )
     (exist-event ;exist-event
     (caption "FORMATION OF THE MACROCRACKS OF THE UNACCEPTABLE
SIZE")
     (caption-kin "KINETICS OF DESTRUCTION 'CORROSION FATIGUE 1'")
     (cf ?x)
     )
     =>
     (assert
     (exist-event ;exist-event
     (caption "MACROCRACKS DEVELOPMENT OF STRUCTURAL DAMAGE IN ZONES
OF STRESS CONCENTRATORS 'FROM THE HOLE IN THE SAME PLANE IN TWO
DIRECTIONS'") ;может возникнуть /событие/
     (probabilityrel "1")
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
     ))
     )
     (defrule des-kinatic-events-ky-1005 "Описание правила: des-
kinatic-events-ky-1005"
     (exist-dam ;exist-dam
     (caption-dam "CRACK")
```

```
     (dam-istochnik "HOLE")
     (dam-orientacia "PERPENDICULARLY")
     (dam-napravlenie "LONGITUDINAL")
     (dam-kolichestvo "SINGLE")
     (caption-meh "CORROSION FATIGUE")
     )
     (exist-event ;exist-event
     (caption "FORMATION OF THE MACROCRACKS OF THE UNACCEPTABLE
SIZE")
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
     )
     =>
     (assert
     (exist-event ;exist-event
     (caption "MACROCRACKS DEVELOPMENT OF STRUCTURAL DAMAGE IN ZONES
OF STRESS CONCENTRATORS 'FROM THE HOLE IN THE SAME PLANE IN TWO
DIRECTIONS'") ;может возникнуть /событие/
     (probabilityrel "1")
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
     ))
     )
     (defrule des-kinatic-events-ky-1006 "Описание правила: des-
kinatic-events-ky-1006"
     (exist-dam ;exist-dam
     (caption-dam "CRACK")
     (dam-istochnik "HOLE")
     (dam-orientacia "PERPENDICULARLY")
     (dam-napravlenie "LONGITUDINAL")
     (dam-kolichestvo "SINGLE")
     (caption-meh "CORROSION FATIGUE")
     )
     (exist-event ;exist-event
```

```
     (caption "FORMATION OF THE MACROCRACKS OF THE UNACCEPTABLE
SIZE")
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
     )
     =>
     (assert
     (exist-event ;exist-event
     (caption "MACROCRACKS DEVELOPMENT OF STRUCTURAL DAMAGE IN ZONES
OF STRESS CONCENTRATORS 'FROM THE HOLE IN THE SAME PLANE IN TWO
DIRECTIONS'") ;может возникнуть /событие/
     (probabilityrel "1")
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
     ))
     )
     (defrule dam-mechanism-ky-1001 "правило выявления /механизма/"
     (mechanical-stress-const ;механические нагрузки
     (cycle-frequency "HIGH") ;-----------
     )
     (technological-environment ;технологическая среда
     (ph "ACTIVE") ;водородный показатель
     (properties-alternation "YES") ;чередование свойств среды
/да/нет/
     )
     (incident-object ;объект инцидента
     (caption ?id-inc-obj) ;код
     )
     (material ;материал
     (type "STEEL") ;вид
     (chemical-prop-alloying "LOW-ALLOY STEEL") ;легированность
     )
     =>
     (assert
     (exist-event ;exist-event
```

```
     (caption "MECHANISM 'CORROSION FATIGUE'") ;может возникнуть
/механизм/
     (cf "0,9") ;ку
     ))
     )
     (defrule dam-mechanism-ky-1002 "правило выявления /механизма/"
     (mechanical-stress-const ;механические нагрузки
     (cycle-frequency "HIGH") ;-----------
     )
     (technological-environment ;технологическая среда
     (ph "ACTIVE") ;водородный показатель
     (properties-alternation "YES") ;чередование свойств среды
/да/нет/
     )
     (incident-object ;объект инцидента
     (caption ?id-inc-obj) ;код
     )
     (material ;материал
     (type "STEEL") ;вид
     (chemical-prop-alloying "LOW-ALLOY STEEL") ;легированность
     )
     (exist-event ;exist-event
     (caption "MACROCRACKS DEVELOPMENT OF STRUCTURAL DAMAGE IN ZONES
OF STRESS CONCENTRATORS 'FROM THE HOLE IN THE SAME PLANE IN TWO
DIRECTIONS'")
     (cf ?x)
     )
     =>
     (assert
     (exist-event ;exist-event
     (caption "MECHANISM 'CORROSION FATIGUE'") ;может возникнуть
/механизм/
     (cf "0,9") ;ку
     ))
     )
```

```
(defrule dam-kinatics-ky-1001 "правило выявления /кинетики/"
(exist-event ;exist-event
(caption "MECHANISM 'CORROSION FATIGUE'") ;наблюдается
/механизм/
(cf ?x)
)
=>
(assert
(exist-kin ;exist-kin
(caption "KINETICS 'CORROSION FATIGUE'") ;может возникнуть
/кинетика/
(cf ?x)
))
)
(defrule dam-kinatic-events-ky-1001 "правило возникновения
повреждения: если наблюдается кинетика коррозионная усталость то
может возникнуть повреждение трещина, количество - одиночные,
продольные, ориентированные перпендикулярно"
(exist-kin ;exist-kin
(caption "KINETICS 'CORROSION FATIGUE'") ;наблюдается
/кинетика/
(cf ?x)
)
=>
(assert
(exist-dam ;exist-dam
(dam-cf ?x)
(caption-dam "CRACK")
(dam-orientacia "PERPENDICULARLY")
(dam-napravlenie "LONGITUDINAL")
(dam-kolichestvo "SINGLE")
(caption-meh "CORROSION FATIGUE")
(id-dam "КУ-1")
))
(assert
```

```
     (exist-event ;exist-event
     (caption "MACROCRACKS DEVELOPMENT OF STRUCTURAL DAMAGE IN ZONES
OF STRESS CONCENTRATORS 'FROM THE HOLE IN THE SAME PLANE IN TWO
DIRECTIONS'") ;может возникнуть /событие/
     (probabilityrel "1")
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
     ))
     )
     (defrule dam-kinatic-events-ky-1002 "Описание правила: dam-
kinatic-events-ky-1002"
     (exist-kin ;exist-kin
     (caption "KINETICS 'CORROSION FATIGUE'") ;наблюдается
/кинетика/ /
     )
     (exist-dam ;exist-dam
     (caption-dam "CRACK")
     (dam-orientacia "PERPENDICULARLY")
     (dam-napravlenie "LONGITUDINAL")
     (dam-kolichestvo "SINGLE")
     (caption-meh "CORROSION FATIGUE")
     )
     (exist-event ;exist-event
     (caption "FORMATION OF THE MACROCRACKS OF THE UNACCEPTABLE
SIZE")
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
     )
     =>
     (assert
     (exist-event ;exist-event
     (caption "MERGING MICROCRACKS") ;может возникнуть /событие/
     (probabilityrel "1")
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
```

```
     ))
     )
     (defrule dam-kinatic-events-ky-1003 "Описание правила: dam-
kinatic-events-ky-1003"
     (exist-kin ;exist-kin
     (caption "KINETICS 'CORROSION FATIGUE'") ;наблюдается
/кинетика/ /
     )
     (exist-event ;exist-event
     (caption "MERGING MICROCRACKS")
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
     )
     =>
     (assert
     (exist-event ;exist-event
     (caption "PREFERENTIAL DEVELOPMENT OF MICROCRACKS ORIENTED IN
PLANES PERPENDICULAR TO THE MAXIMUM TENSILE STRESS") ;может
возникнуть /событие/
     (probabilityrel "1")
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
     ))
     )
     (defrule dam-kinatic-events-ky-1004 "Описание правила: dam-
kinatic-events-ky-1004"
     (exist-kin ;exist-kin
     (caption "KINETICS 'CORROSION FATIGUE'") ;наблюдается
/кинетика/ /
     )
     (exist-event ;exist-event
     (caption "PREFERENTIAL DEVELOPMENT OF MICROCRACKS ORIENTED IN
PLANES PERPENDICULAR TO THE MAXIMUM TENSILE STRESS")
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
```

```
     )
     =>
     (assert
     (exist-event ;exist-event
     (caption "FORMATION OF THE MICROCRACKS") ;может возникнуть
/событие/
     (probabilityrel "1")
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
     ))
     )
     (defrule dam-kinatic-events-ky-1005 "Описание правила: dam-
kinatic-events-ky-1005"
     (exist-kin ;exist-kin
     (caption "KINETICS 'CORROSION FATIGUE'") ;наблюдается
/кинетика/ /
     )
     (exist-event ;exist-event
     (caption "FORMATION OF THE MICROCRACKS")
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
     )
     =>
     (assert
     (exist-event ;exist-event
     (caption "DEVELOPMENT OF INTRAGRANULAR SUBMICROCRACKS DUE TO
THE DISSOLUTION IN THE SLIP PLANES") ;может возникнуть /событие/
     (probabilityrel "1")
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
     ))
     )
     (defrule dam-kinatic-events-ky-1006 "Описание правила: dam-
kinatic-events-ky-1006"
     (exist-kin ;exist-kin
```

```
     (caption "KINETICS 'CORROSION FATIGUE'") ;наблюдается
/кинетика/ /
     )
     (exist-event ;exist-event
     (caption "DEVELOPMENT OF INTRAGRANULAR SUBMICROCRACKS DUE TO
THE DISSOLUTION IN THE SLIP PLANES")
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
     )
     =>
     (assert
     (exist-event ;exist-event
     (caption "FORMATION OF LOCAL DEFORMATION ZONES IN THE SURFACE
DUE TO CYCLIC LOADING") ;может возникнуть /событие/
     (probabilityrel "1")
     (caption-kin "KINETICS 'CORROSION FATIGUE'")
     (cf ?x)
     ))
     )
```

Приложение К. Свидетельства о государственной регистрации программ для ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2017618430

Web-ориентированный редактор моделей трансформаций

Правообладатель: *Федеральное государственное бюджетное учреждение науки Институт динамики систем и теории управления имени В.М. Матросова Сибирского отделения Российской академии наук (ИДСТУ СО РАН) (RU)*

Автор: *Дородных Никита Олегович (RU)*

Заявка № 2017615215
Дата поступления 01 июня 2017 г.
Дата государственной регистрации
в Реестре программ для ЭВМ *01 августа 2017 г.*

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2017618446

RVML editor

Правообладатель: ***Федеральное государственное бюджетное учреждение науки Институт динамики систем и теории управления имени В.М. Матросова Сибирского отделения Российской академии наук (ИДСТУ СО РАН) (RU)***

Автор: ***Дородных Никита Олегович (RU)***

Заявка № **2017615217**
Дата поступления **01 июня 2017 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***01 августа 2017 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

Г.П. Ивлиев

Приложение Л. Акты о внедрении результатов диссертационного исследования

Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

664074 Россия, Иркутск, ул. Лермонтова, 83
телефон: +7(3952)405-000, факс: +7(3952)405-100
E-mail: info@istu.edu
ОКПО 02068249, ОГРН 1023801756120
ИНН/КПП 3812014066/381201001

_____ № _____

на № _____ от _____

УТВЕРЖДАЮ

Проректор по учебной работе

ФГБОУ ВО «ИРНИТУ»

_____ **Д.В. Огнев**

15 сентября 2017 г.

м.п.

АКТ о внедрении программ для ЭВМ

Настоящим Актом подтверждается, что программы для ЭВМ « Web-ориентированный редактор моделей трансформаций (Web Transformation Model Editor)» (автор Дородных Никита Олегович Свидетельство о государственной регистрации программ для ЭВМ № 2017618430 от 01.08.2017. М.: Федеральная служба по интеллектуальной собственности, патентам и товарным знакам, 2017) и . «RVML editor (Web Knowledge Base Designer)» (автор Дородных Никита Олегович Свидетельство о государственной регистрации программ для ЭВМ № 2017618446 от 01.08.2017. М.: Федеральная служба по интеллектуальной собственности, патентам и товарным знакам, 2017) используются при преподавании дисциплин «CASE-средства» и «Инструментальные средства информационных систем», для автоматизированного создания баз знаний экспертных систем на основе анализа концептуальных моделей на кафедре «Автоматизированных систем» ФГБОУ ВО «Иркутский национальный исследовательский технический университет» (ФГБОУ ВО «ИРНИТУ»)

Заведующий кафедрой

«Автоматизированных систем» ФГБОУ ВО «ИРНИТУ»

к.т.н. доцент _____ **С.В. Бахвалов**

Акционерное общество
«Иркутский научно-исследовательский и конструкторский институт
химического и нефтяного машиностроения»
(АО «ИркутскНИИхиммаш»)
Академика Курчатова ул., д. 3, г. Иркутск, 664074
Тел.: (395-2) 41-04-34 Факс: (395-2) 41-05-10 **E-mail:** himmash@irk.ru http://himmash.irk.ru
ОКПО 00220227; ОГРН 1023801748596; ИНН/ КПП 3812010128/381201001
р/сч. 40702810518350102572 в Байкальском банке СБ РФ ОСБ 8586 г. Иркутск кор/сч. 30101810900000000607 БИК 042520607; ИНН банка 3810008677

УТВЕРЖДАЮ
Генеральный директор
Кузнецов Анатолий Макарович
_____
«16» 10 2017 г.

АКТ
об использования результатов научных исследований

Результаты диссертационной работы Дородных Никиты Олеговича «Метод и программное средство разработки баз знаний на основе трансформации концептуальных моделей» используются для автоматизированного создания баз знаний экспертных систем определения причин инцидентов и аварий нефтехимического оборудования при проведении экспертизы промышленной безопасности. В частности, используется программное обеспечение формирования баз знаний на основе концептуальных моделей деревьев событий, отражающих знания о процессах формирования инцидентов и аварий.

Использование указанных результатов позволяет повысить качество проведения технического диагностирования объектов экспертизы промышленной безопасности.

Результаты используются в рамках продолжения НИР, осуществляемых по договору № 052013НИР от 19 сентября 2013 г. «Разработка проблемно-ориентированного редактора продукционных баз знаний для задач оценки технического состояния и остаточного ресурса».

Заведующий научно-исследовательским и
конструкторским отделом оборудования
для химической, нефтехимической,
нефтегазоперерабатывающей и других
отраслей промышленности,
кандидат технических наук Г.М. Мордина