

Федеральное государственное бюджетное учреждение науки
Институт проблем информатики РАН (ИПИ РАН)

на правах рукописи

Вовченко Алексей Евгеньевич

**Распределенная реализация приложений в среде
предметных посредников**

05.13.11. - математическое и программное обеспечение
вычислительных машин, комплексов и компьютерных сетей

ДИССЕРТАЦИЯ

на соискание ученой степени
кандидата технических наук

Научный руководитель

Доктор физико-математических наук, профессор

Калиниченко Л.А.

МОСКВА

2012

Оглавление

Введение.....	6
ГЛАВА 1. Организация решения задач над неоднородными распределенными информационными ресурсами в среде предметных посредников	17
1.1. Концепция предметных посредников.....	17
1.2. Принципы построения сред предметных посредников	19
1.3. Обобщенная архитектура среды предметных посредников.....	22
1.4. Обобщенная архитектура исполнительного слоя среды предметных посредников для решения задач	24
1.5. Пример решения задачи в среде предметных посредников.....	28
1.5.1. Описание задачи определения вторичных стандартов.....	28
1.5.2. Варианты оптимизации алгоритма решения задачи	35
1.6. Выводы по главе	37
ГЛАВА 2. Рассредоточение реализации алгоритма решения задачи в распределенной среде предметных посредников	38
2.1. Постановка задачи рассредоточения в среде предметных посредников.....	38
2.2. Описание языка правил предметных посредников и языка спецификации взглядов.....	42
2.3. Функции компонентов среды предметных посредников	46
2.4. Представление графа зависимостей функциональных операций и модели рассредоточения	49
2.5. Семантика графа зависимостей функциональных операций и модели рассредоточения	52

2.6.	Обобщение задачи рассредоточения	54
2.7.	Методы построения эффективного рассредоточения	55
2.8.	Перестановка операций при построении рассредоточения.....	60
2.9.	Экспертные правила	64
2.10.	Обзор существующих подходов.....	70
2.11.	Выводы по главе	72
ГЛАВА 3. Сопряжение языков программирования с декларативным языком правил предметных посредников		
75		
3.1.	Краткая характеристика задачи сопряжения	75
3.2.	Описание проблемы несоответствия импеданса при сопряжении языков запросов к базам данных и языков программирования	76
3.3.	Характеризация сопряжений ЯП с базами данных	78
3.4.	Подход к сопряжению языков программирования с предметными посредниками	85
3.4.1.	Статический подход	86
3.4.2.	Динамический подход.....	91
3.4.3.	Долговечные и транзитные объекты.....	92
3.5.	Реализация сопряжения предметных посредников с языками программирования	93
3.6.	Обзор существующих подходов.....	97
3.7.	Выводы по главе	102
ГЛАВА 4. Конструирование адаптеров информационных ресурсов		
103		
4.1.	Краткая характеристика адаптеров	103
4.2.	Архитектура простого адаптера	104
4.3.	Подход к конструированию адаптеров.....	107

4.4.	Требования к адаптеру для поддержки эффективного выполнения рассредоточенной программы	110
4.5.	Программируемый адаптер.....	112
4.6.	Основные особенности реализации адаптеров для задачи рассредоточения.....	114
4.7.	Описание реализации реляционного адаптера	116
4.8.	Обзор существующих подходов.....	120
4.9.	Выводы по главе	121
ГЛАВА 5. Практическое применение и тестирование системы построения рассредоточений.....		122
5.1.	Описание программной реализации системы построения рассредоточений.....	122
5.2.	Пример применения системы построения эффективного рассредоточения для научной задачи	124
5.3.	Описание процесса тестирования алгоритмов построения эффективного рассредоточения.....	147
5.3.1.	Описание тестовых примеров	147
5.3.2.	Набор тестов.....	148
5.3.3.	Результаты тестирования	149
5.4.	Выводы по главе	152
Заключение		153
Литература		155
Приложение А Грамматика декларативного языка правил предметных посредников (язык Syfs)		168

Приложение Б Алгебраическая форма языка правил предметных посредников – язык Asyfs.....	173
Приложение В Статическое связывание языка СИНТЕЗ и языка программирования Java	180
Приложение Г Динамическое связывание языка СИНТЕЗ и языка программирования Java	190
Приложение Д Спецификация адаптера.....	197
Приложение Е Спецификация синтетического тестовой задачи	214

Введение

В различных областях науки наблюдается экспоненциальный рост объема получаемых экспериментальных данных. Например, в астрономии текущий и ожидаемый темп роста данных от наземных и космических телескопов удваивается в течение периода от шести месяцев до одного года. Сложность использования таких данных увеличивается еще и вследствие их структурной и модельной разнородности. Число организаций, получающих данные наблюдений в отдельных областях науки, велико. Разнообразие и информационная несогласованность получаемой информации вызывается не только большим числом независимых организаций, производящих наблюдения, но и разнообразием объектов наблюдения. Вместе с тем непрерывно совершенствуются и техники наблюдений, вызывающие адекватные изменения структуры и содержания накапливаемой информации. Это приводит к необходимости использования неоднородной, распределенной информации, накопленной в течение значительного периода наблюдений технологически различными инструментами.

Основной идеей в инфраструктуре доступа к множественным неоднородным информационным ресурсам является введение промежуточного слоя между ресурсами и потребителями информации. Базовыми компонентами промежуточного слоя являются предметные посредники [1], определяемые независимо от информационных ресурсов. Применение среды предметных посредников для решения задач над множеством распределенных неоднородных информационных ресурсов обеспечивает их интеграцию в контексте предметных областей приложений. Такой контекст для класса приложений определяется декларативной спецификацией посредников.

В лаборатории «Композиционных методов и средств построения информационных систем» в Институте проблем информатики РАН (ИПИ

РАН) разработаны средства поддержки среды предметных посредников для решения задач над неоднородными информационными ресурсами [1]. Полученный опыт использования среды предметных посредников для решения разнообразных научных задач в области астрономии показывает необходимость разработки подходов для повышения эффективности реализации задач в таких средах. Проблема эффективной реализации (задача рассредоточения) алгоритма решения задачи в среде предметных посредников заключается в том, что каждый компонент среды обладает широкими возможностями, которые во многом пересекаются, что приводит к неоднозначности выбора конкретной реализации. Например, какая-то часть алгоритма решения задачи (далее просто задачи) может быть реализована как программа на языке программирования (ЯП), либо как программа посредника. Существует ряд видов компонентов среды предметных посредников, между которыми может быть рассредоточена реализация задачи:

- системы программирования (СП);
- предметные посредники;
- средства поддержки отображений классов ресурсов в классы посредников;
- адаптеры информационных ресурсов;
- конкретные информационные ресурсы.

Таким образом, многоязычная спецификация алгоритма решения задачи представляет собой совокупность спецификаций, заданных на языках программирования соответствующих компонентов. Языки программирования фрагментов алгоритма решения задач могут сильно различаться. Например, императивные языки, языки правил, языки представления взглядов.

Множество возможных реализаций задачи образует пространство вариантов, называемое моделью рассредоточения. Каждый вариант рассредоточения (состояние модели рассредоточения) характеризуется назначениями для всех операций алгоритма решения задачи. Назначение

определяет компонент, в котором данная операция будет реализована. В качестве назначений могут выступать перечисленные выше компоненты среды предметных посредников, между которыми может быть рассредоточена реализация задачи. Задача рассредоточения заключается в построении эффективного рассредоточения, т.е. такого рассредоточения, для которого время выполнения минимально или близко к таковому. Время выполнения эффективного рассредоточения и начального рассредоточения могут существенно отличаться. Особенно остро проблема построения эффективного рассредоточения встает в тех случаях, когда требуются многократные прогоны решаемой задачи для различных наборов параметров. Например, в астрономии некоторая задача была сформулирована для площадки размером 1 квадратный градус. Тогда, чтобы прогнать решаемую задачу по всему небу для полосы шириной 1 градус, потребуется 360 прогонов. Для того чтобы прогнать решаемую задачу для всего неба потребуется $360 \cdot 180 = 64800$ прогонов. При таких условиях даже выигрыш во времени выполнения одного прогона в 1 минуту экономит полтора месяца вычислений.

Как уже было сказано, выполнение частей рассредоточенной программы может происходить на ресурсах, управляемых адаптерами, на самих адаптерах, а также в системах программирования. В связи с задачей рассредоточения возникают вспомогательные задачи. Одна из них заключается в разработке архитектуры программируемого адаптера, поддерживающего реализацию рассредоточения и позволяющего осуществлять эффективное выполнение операций на ресурсах, адаптерах и в посреднике. Другая задача заключается в разработке хорошо обоснованного и эффективного сопряжения среды предметных посредников с языками программирования.

В диссертационной работе разрабатываются и исследуются подходы построения эффективного рассредоточения для реализации алгоритма решения задачи в среде предметных посредников. В работе исследуются подходы сопряжения среды предметных посредников с системами программирования.

Наконец в работе разработана архитектура программируемых адаптеров информационных ресурсов, согласованная с требованиями средств рассредоточения, а также подходы к конструированию подобных адаптеров.

Объект и предмет исследования

Объектом исследования являются инфраструктуры и методы решения задач над множеством неоднородных распределенных ресурсов на основе парадигмы предметных посредников. Предметом исследования являются методы и средства повышения эффективности решения задач в подобных инфраструктурах, а также согласованные с ними методы и средства сопряжения императивных языков и систем программирования с декларативными языками посредников, архитектуры адаптеров информационных ресурсов и подходы к их реализации.

Цели и задачи работы

Целью работы является разработка подхода к построению эффективного рассредоточения для реализации алгоритма решения задач в среде предметных посредников. Для достижения поставленной цели необходимо решить следующие задачи:

1. Разработка методов и средств представления рассредоточений и манипулирования ими, а также методов и средств оценки эффективности рассредоточений.
2. Разработка и реализация алгоритмов построения эффективного рассредоточения в среде предметных посредников.
3. Создание адекватных задаче рассредоточения методов и средств сопряжения систем программирования с декларативным языком предметных посредников.

4. Разработка архитектуры программируемых адаптеров, обеспечивающих эффективное рассредоточение реализации программ посредников над ресурсами, а также методов конструирования адаптеров. Создание адаптеров для конкретных классов информационных ресурсов.
5. Создание средств оценки эффективности различных вариантов рассредоточения реализации задач, полученных в результате применения алгоритмов построения рассредоточений.

Методы исследования

При решении поставленных в работе задач использовались методы объектного анализа и проектирования, концептуального моделирования, теории баз данных, теории дедуктивных баз данных, теории множеств, теории отображения и уточнения спецификаций посредника и ресурсов.

Научная новизна

В диссертационной работе получены следующие новые научные результаты:

- разработан подход к построению эффективного рассредоточения реализации алгоритма решения задачи в среде предметных посредников, позволяющий генерировать варианты рассредоточений, а также оценивать их эффективность;
- разработан подход к сопряжению систем программирования с предметными посредниками, на основании которого созданы реализации статического и динамического сопряжения;
- разработана архитектура программируемых адаптеров информационных ресурсов, обеспечивающая эффективное выполнение рассредоточения над ресурсами, а также подход к конструированию подобных адаптеров.

Достоверность полученных результатов

Обоснованность и достоверность научных положений, выводов и практических результатов подтверждается результатами анализа существующих подходов и систем в исследуемых областях; накопленным опытом решения задач в среде предметных посредников; результатами практического применения разработанных подходов для реализации системы построения эффективного рассредоточения; результатом применения сопряжения декларативного языка правил предметных посредников с языком программирования Java для задачи рассредоточения; применением разнообразных адаптеров для задачи рассредоточения; результатами тестирования алгоритмов построения эффективного рассредоточения при решении реальных задач.

Практическая значимость

Предложенный подход рассредоточения реализации приложений в среде предметных посредников может быть применен при решении задач в различных прикладных областях, например, в астрономии, в биологии, в науках о Земле, и в любой другой области, требующей решения задач над множеством неоднородных распределенных информационных ресурсов.

Разработанный подход к конструированию адаптеров может быть применен в системах интеграции неоднородных ресурсов. Например, в области виртуальных обсерваторий, для обеспечения интегрированного доступа к распределенным ресурсам в информационных системах.

Предлагаемый в работе подход сопряжения среды предметных посредников с системами программирования может быть использован при реализации доступа к базам данных из языков программирования. В разработанном подходе удалось решить проблему несоответствия импеданса. Тем самым обеспечивается возможность повышения эффективности и

надежности разрабатываемых систем, а также уменьшения времени, затрачиваемого на отладку и тестирование систем.

Все предложенные в работе подходы реализованы и прошли проверку при решении практических задач в области астрономии.

Результаты диссертационной работы использованы в проектах, выполняемых по планам ИПИ РАН, в проектах РФФИ 05-07-90413-в, 06-07-89188-а, 10-07-00342-а и 10-07-00640-а, а также в проекте 4.2 Программы фундаментальных исследований Президиума РАН №15.

Реализация результатов исследования

На основании предложенного в работе подхода к построению эффективного рассредоточения для реализации алгоритмов решения задач была разработана система построения рассредоточений. Система была применена для повышения эффективности решения задач в области астрономии. Эффективный алгоритм решения задачи, связанной с наблюдением гамма-всплесков, составляет основу программной системы, используемой в ИКИ РАН для практического решения задачи. Разработанный в работе подход построения сопряжения декларативного языка правил предметных посредников с языками программирования лег в основу программной реализации сопряжения, используемого в исполнительной среде предметных посредников. Подход к конструированию адаптеров, соответствующих требованиям системы построения рассредоточений, применен для создания ряда адаптеров: для реляционных СУБД, для объектно-реляционных СУБД, для слабоструктурированных данных (XML), для реестров системы Астрогрид, для ресурсов DSA системы Астрогрид, для веб-сервисов, для астрономического ресурса SDSS, для информационного грида Vizier, для потоковых данных. Реализация подтверждается практическим опытом решения задач, а также свидетельствами о регистрации четырех программных продуктов.

Апробация работы

Основные результаты диссертации докладывались на Международных конференциях: «15th East European conference on advances in databases and information systems» ADBIS 2011 (Austria, Vienna 2011), «Distributed Computing and Grid-technologies in Science and Education» (Дубна 2008, Дубна 2010), «Современные информационные технологии и ИТ-образование» (Москва 2009, Москва 2011); на Российских конференциях по электронным библиотекам RCDL (Дубна 2008, Петрозаводск 2009, Казань 2010, Воронеж 2011); на семинаре по Российской Виртуальной Обсерватории (Москва 2007); на семинаре Московской секции ACM SIGMOD (Москва 2009); на научных семинарах по проекту СИНТЕЗ лаборатории Композиционных методов проектирования информационных систем Института проблем информатики РАН.

На защиту выносятся следующие, полученные автором результаты:

- подход к построению эффективного рассредоточения реализации алгоритма решения задач в среде предметных посредников, а также программный инструментарий системы построения рассредоточений;
- подход к сопряжению систем программирования с декларативным языком предметных посредников, а также программные средства, обеспечивающие статическое и динамическое связывание предметных посредников и объектно-ориентированного языка (Java).
- архитектура программируемых адаптеров информационных ресурсов, обеспечивающая эффективное выполнение рассредоточения над ресурсами, подход к конструированию подобных адаптеров, а также все разработанные в работе адаптеры.

Публикации по теме диссертации

Результаты диссертации опубликованы в 13 печатных работах, в том числе имеются 2 публикации в научных журналах, входящих в перечень изданий, рекомендованных ВАК. Основные результаты подтверждаются четырьмя свидетельствами о регистрации программ.

Структура работы

Текст диссертации включает введение, пять глав, заключение, список литературы и 6 приложений.

В первой главе представлена обобщенная архитектура среды предметных посредников для решения задач над неоднородными информационными ресурсами, в этой же главе приводятся обоснования необходимости построения эффективного рассредоточения для реализации алгоритма решения задач.

Во второй главе представлен подход построения эффективного рассредоточений для реализации алгоритма решения задач в среде предметных посредников. В главе дано описание языка правил предметных посредников и приведены функциональные возможности компонентов среды предметных посредников. Для представления множества вариантов рассредоточений используется модель рассредоточения. Подробное описание семантики модели, а также графического представления также приведено в главе. Для построения эффективного рассредоточения используется алгоритм позволяющий генерировать варианты рассредоточений, а также оценивать их эффективность. При описании методов, формально определены понятия модели рассредоточения, эффективного рассредоточения, а также понятие оценки эффективности рассредоточения. Дан анализ возможности перестановок различных операций в модели рассредоточения, на основании которого выработан набор экспертных правил. Правила используются для принятия

решения о перестановке операций. Также в главе описаны два алгоритма автоматического построения эффективного рассредоточения.

В третьей главе представлен подход сопряжения предметных посредников с языками программирования. В главе представлена характеристика возможных подходов, а также определяется то, какими характеристиками должен обладать подход для решения проблем несоответствия импеданса. Данный подход называется статическим подходом. Также в главе приводится описание противоположного подхода – динамического, направленно на предоставление максимального широких возможностей пользователю. В работе предлагается одновременно реализовать как статический, так и динамический подходы, обеспечивая максимальный уровень надежности (решение проблем несоответствия импеданса) вместе с широкими возможностями для специалистов. В главе также особое внимание уделено проблеме долговечных объектов и верификации отображений.

В четвертой главе представлен подход к полуавтоматическому созданию адаптеров для предметных посредников. В главе приводится общая архитектура адаптеров, особенностью которой являются широкие возможности по сбору и оценки статистики выполнения запросов, необходимой для эффективного планирования выполнения программы. В главе описывается подход полуавтоматического создания адаптеров. Главная идея автоматизации создания адаптеров заключается в выделении в архитектуре адаптера три группы компонентов: создаваемые автоматически; генерируемые полуавтоматически (средствами унификатора моделей); автоматизация создания которых не представляется возможной. В главе также описываются разработанные с помощью подхода адаптеры. Наконец в главе описываются программируемые адаптеры, необходимые для эффективного рассредоточения реализации.

В пятой главе представлено описание системы построения рассредоточений, тестирование способностей системы по построению эффективного рассредоточения.

В заключении приводятся основные результаты, полученные в рамках данной работы.

В приложении А содержится грамматика языка правил посредников, язык Syfs, встраиваемая в язык Java.

В приложении Б содержится описания алгебраической формы языка правил посредников – языка Asyfs.

В приложении В содержится описание статического связывания декларативного языка правил предметных посредников и языка программирования Java.

В приложении Г содержится описание динамического связывания декларативного языка правил предметных посредников и языка программирования Java.

В приложении Д содержится подробная спецификация адаптера.

В приложение Е содержится спецификация тестового синтетического примера.

ГЛАВА 1. Организация решения задач над неоднородными распределенными информационными ресурсами в среде предметных посредников

1.1. Концепция предметных посредников

Основной идеей в инфраструктуре доступа к множественным неоднородным информационным источникам является введение промежуточного слоя между информационными ресурсами и потребителями информации. Основными компонентами среды предметных посредников являются посредники [1], существующие независимо от информационных ресурсов. Уровень предметных посредников вводится как часть информационных систем, создаваемых для решения научных задач. Каждый предметный посредник задает спецификацию предметной области для решения некоторого класса задач. Спецификация предметной области включает онтологию предметной области, схему посредника, описание процессов, и др. Схема посредника включает определения структур данных, которые планируется использовать для решения задачи и определения функций, необходимых для обработки получаемых данных. Спецификация предметной области использует каноническую информационную модель [2] для унифицированного представления как предметной области, так и разнообразных видов моделей информационных ресурсов. Каноническая модель служит в качестве некоторого унифицированного языка при интеграции неоднородных источников данных и сервисов.

Существует два принципиально различных подхода к проблеме интегрированного представления описания предметной области задачи по отношению к множеству релевантных задаче информационных ресурсов. В первом подходе «движимом ресурсами», схема посредника формируется как интегрированная схема множества ресурсов независимо от приложения. Во

втором подходе «движимом приложением», описание предметной области приложения образуется независимо от ресурсов в терминах понятий, структур данных, функций, процессов, а затем уже релевантные приложению ресурсы отображаются в это описание.

Подход «движимый ресурсами» является не масштабируемым по отношению к числу ресурсов, не дает возможности достижения семантической интеграции ресурсов в контексте конкретного приложения.

Подход «движимый приложениями» предполагает создание предметного посредника, который поддерживает взаимодействие между приложением и ресурсом на основе определения прикладной области (определения посредника). Второй подход имеет очевидные преимущества по отношению к подходу, движимому информационными ресурсами. Процесс регистрации неоднородных информационных ресурсов в предметном посреднике в подходе, движимом приложениями, основан на технике GLAV [3], комбинирующей два подхода: LAV (Local As View) [4], при котором схемы регистрируемых ресурсов рассматриваются как материализованные взгляды над виртуальными классами посредника; и GAV (Global As View) [4, 5], при котором глобальная схема посредника является взглядом над схемами ресурсов. В составе GLAV спецификаций GAV взгляды служат для разрешения различных конфликтов между спецификациями ресурсов и посредника. Подобная техника регистрации обеспечивает стабильность спецификации приложения при изменении конкретных информационных ресурсов и их фактического присутствия (удаление ресурса, добавление новых ресурсов), а также масштабируемость посредников по отношению к числу регистрируемых ресурсов. Настоящая работа основана главным образом на подходе, движимом приложениями и технике GLAV.

1.2. Принципы построения сред предметных посредников

Выбор проектов для сравнения, учитывая их число и разнообразие, является непростой задачей. Прежде всего, в данной работе интересны те проекты, которые можно было бы достаточно содержательно сопоставлять с проектом СИНТЕЗ [1-3, 7-9], в рамках которого выполняется данная работа. С другой стороны, интерес представляют и отличные от проекта СИНТЕЗ работы, интересные оригинальными решениями. Были рассмотрены следующие проекты: LAV – Agora [10, 11], Infomaster [12, 13], SIRUP [14 - 16], PICSEL [17-19], Information Manifold [20]; GAV – MedMaker [21], MOMIS [22, 23], TSIMMIS [24, 25]; BAV – AutoMed [26- 29]; GLAV – СИНТЕЗ. Рассмотрены также работы по их сравнению [30, 31, 32]. Основное внимание было уделено проектам, приводимым к LAV (LAV, BAV, GLAV). В Институте проблем информатики РАН были сформулированы основные принципы построения сред предметных посредников. Все рассмотренные проекты в той или иной степени удовлетворяют этим принципам.

Основополагающим принципом является использование **подхода движимого приложениями**. Как было отмечено ранее, существуют два подхода: «движимый ресурсами» и «движимый приложением». Используя подход, ориентированный на проблему («движимый приложением»), специалист формулирует задачу, описывая базовые сущности и понятия предметной области, в рамках которой предполагается решать задачу. Среди прочего, специфицируются: структуры данных, понятия предметной области, функции, процессы и пр. Спецификация предметной области, представляет собой спецификацию предметного посредника, для решения класса задач. Сущности и понятия предметной области, определенные таким образом, не зависят от существующих информационных ресурсов.

Вторым принципом является **принцип расширяемости канонической модели данных**. При интеграции неоднородных ресурсов в посреднике нужно

уметь семантически отождествлять объекты, представленные в различных информационных моделях, и семантически правильно отображать схемы интегрируемых ресурсов в схему посредника. При интеграции неоднородных ресурсов (представленных в различных моделях) для однородного представления их семантики требуется приведение информационных моделей ресурсов к унифицированному виду в рамках некоторой информационной модели, которая называется *канонической*.

Для унификации разнородных спецификаций, прежде всего, требуется умение сопоставлять спецификации различных ресурсов друг с другом так, чтобы можно было отвечать на вопрос, можно ли при реализации посредника использовать спецификацию существующего ресурса вместо фрагмента спецификации посредника. Для этого достаточно доказать, что рассматриваемые спецификации находятся в отношении уточнения. Говорят, что спецификация А уточняет спецификацию D, если А можно использовать вместо D так, что пользователь D не будет замечать этой замены. Средства доказательства факта уточнения, реализуемые на основе теоретико-модельных нотаций и соответствующего инструментария [33-35], составляют фундамент применяемых методов конструирования унифицирующих (канонических) моделей представления информации в посредниках. Каноническая информационная модель служит в качестве общего языка, эсперанто, для адекватного выражения семантики разнородных моделей представления информации, используемых в разнообразных информационных ресурсах. Методы отображения информационных моделей и синтеза расширяемых канонических информационных моделей для среды предметных посредников подробно рассмотрены в [2].

Третьим принципом является **принцип семантической интеграции** релевантных неоднородных информационных ресурсов в спецификации посредника. Процесс семантической интеграции ресурсов в спецификации посредника называется регистрацией ресурсов. Регистрация релевантных

посреднику ресурсов рассматривается как задача композиционного проектирования систем [36, 37]. Регистрация ресурсов есть процесс целенаправленной трансформации спецификаций. Регистрация включает: декомпозицию спецификаций посредника на непротиворечивые фрагменты, поиск среди спецификаций релевантных ресурсов подходящих типов данных — кандидатов для уточнения ими спецификаций типов посредника, построение выражений, определяющих классы ресурсов в виде композиции классов посредника. Результатом процесса регистрации ресурсов в спецификации посредника являются семантические отображения классов ресурсов в классы посредника, называемые также взглядами [3].

Последним принципом является **принцип расширяемости архитектуры**. В настоящий период существуют множество инфраструктур решения задач: веб-сервисы, семантический-веб, виртуальные обсерватории, грид-инфраструктуры, облачные инфраструктуры. Вместе с тем растет как число новых информационных ресурсов, так и их разнообразие. Ресурсы реляционных и объектно-реляционных данных, веб-сервисы, грид-сервисы, программы, онтологии, XML-ресурсы – это далеко не полный список разнообразия информационных ресурсов. Для технической унификации разнообразных информационных ресурсов используются адаптеры. Адаптер [38] – элемент архитектуры посредника, который обеспечивает отображение модели информационного ресурса в каноническую модель. Это отображение заключается в отображении схем ресурса в канонические схемы и преобразование операторов языка манипулирования данными канонической модели в язык запросов источника. Адаптеры призваны обеспечивать унифицированный доступ к разнородным информационным источникам. Важным является и вопрос автоматизации конструирования новых адаптеров, т.к. разнообразие ресурсов увеличивается.

1.3. Обобщенная архитектура среды предметных посредников

Исходя из принципов построения сред предметных посредников, в лаборатории ИПИ РАН разработана обобщенная архитектура среды предметных посредников для решения задач, представленная на рисунке 1.1.

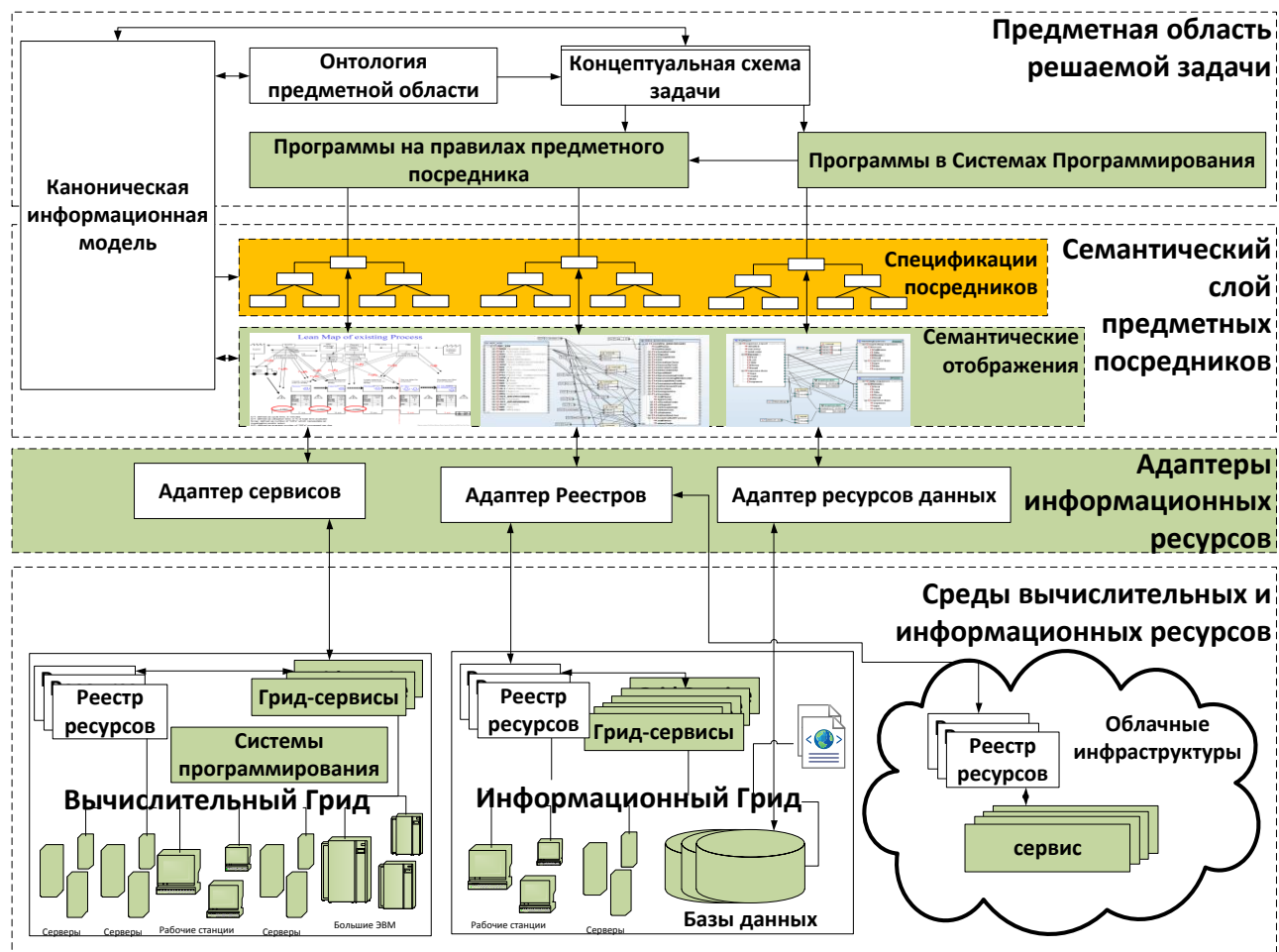


Рисунок 1.1. Обобщенная архитектура среды предметных посредников для решения задач

Архитектура предметных посредников включает:

- уровень информационных ресурсов, включающий базы данных, сервисы, средства программирования и другие средства, организованные в рамках грид, облачных или других инфраструктур;
- уровень адаптеров, обеспечивающий интероперабельность ресурсов благодаря технической унификации их интерфейсов и введению дистанционных механизмов обращения к ресурсам (адаптеры,

осуществляющие преобразование запросов, выраженных в канонической информационной модели посредников, в их представление в информационной модели ресурса);

- уровень предметных посредников, каждый из которых создает спецификацию предметной области для решения некоторого класса задач, используя каноническую информационную модель («эсперанто») для представления семантики предметной области и унифицированного отображения разнообразных видов информационных моделей ресурсов (моделей данных, сервисных моделей, онтологических моделей, процессных моделей);
- уровень задач (приложений), формулируемых в терминах одного или нескольких посредников.

Для решения задач используется метод, движимый приложениями. Отправляясь от задачи, определяется онтология предметной области (понятия и связи между ними), затем строится концептуальная схема, содержащая информационные структуры и методы, необходимые для решения задачи. Таким образом, образуется семантическая спецификация решения задачи, независимая от ресурсов. В терминах концептуальной схемы формулируются программы для решения задачи на языке правил посредника и на языках программирования. После этого определяются инфраструктуры содержащие ресурсы, необходимые для решения задачи. Далее, идентифицируются ресурсы, релевантные задаче, используя реестры доступных инфраструктур. Релевантные задаче ресурсы регистрируются в предметных посредниках, образуя тем самым семантические отображения классов ресурсов в классы посредника (взгляды). Для всех ресурсов релевантных задаче конструируются адаптеры, а также задаются операционные возможности адаптеров и ресурсов.

1.4. Обобщенная архитектура исполнительного слоя среды предметных посредников для решения задач

В предыдущем разделе дана обобщённая архитектура среды предметных посредников, в рамках которой возможно задание алгоритма решения задачи. На рисунке 1.2. представлена архитектура исполнительного слоя среды предметных посредников для решения задач.

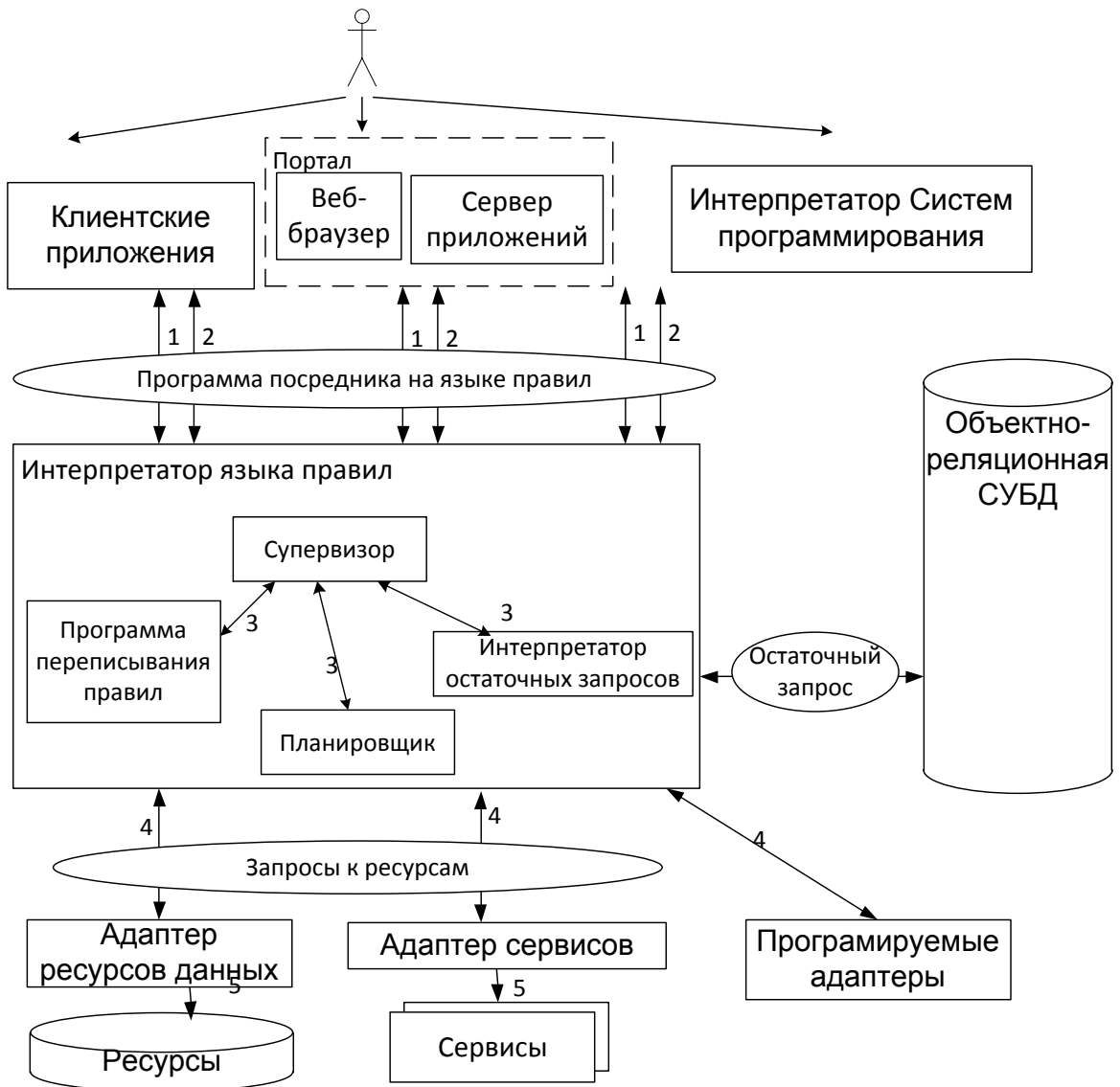


Рисунок 1.2. Архитектура исполнительного слоя среды предметных посредников для решения задач

Алгоритм решения задачи задаётся в виде программы на языке программирования (ЯП), программы на языке правил посредника, спецификацией семантических отображений (взглядов), спецификациями

адаптеров задающих операционные возможности ресурсов. Архитектура, представленная на рисунке 1.1., была бы не полной без описания исполнительных механизмов, позволяющих выполнять алгоритм решения задачи, заданный в виде спецификаций.

Программа заданная на языке программирования выполняется интерпретатором системы программирования (ИСП). Программа посредника формулируется в терминах схемы посредника и представляет собой совокупность правил. Программа на правилах переписывается из терминов посредника в термины информационных ресурсов. Переписанные правила представлены в унифицированном виде в канонической модели данных. Для переписывания правил используются взгляды, определенные при регистрации ресурсов в посреднике. Переписанная программа в терминах информационных ресурсов планировщиком разбивается на отдельные подзапросы к различным адаптерам. Место выполнения отдельных подзапросов определяется планировщиком, но зависит от операционных возможностей адаптеров и ресурсов.

Язык правил посредника представляет собой объектный (типизированный) вариант языка Datalog со стратифицированной семантикой. Программа посредника состоит из набора правил. Правила выполняются над базой фактов. Подробно семантика языка рассматривается в следующей главе, в разделе 2.2. *Интерпретатор языка правил* посредников обеспечивает:

- обработку программ пользователя;
- переписывание программ в запросы над зарегистрированными в посреднике ресурсами, отправку переписанных запросов на выполнение соответствующим адаптерам;
- получение результатов запросов от адаптеров;
- выполнение остаточного запроса и отправку результата клиенту.

Интерпретатор языка правил включает следующие компоненты: *Супервизор, Программу переписывания правил, Планировщик, Интерпретатор остаточных запросов.*

Супервизор является связующим компонентом посредника и определяет его интерфейс. *Супервизор* обрабатывает программы пользователя и для их выполнения взаимодействует с другими компонентами посредника. В общем случае алгоритм супервизора следующий:

- для каждой программы на правилах создается сессия;
- исходная программа переписывается программой переписывания правил с использованием взглядов;
- планировщиком переписанная программа преобразуется в подзапросы ко всем информационным ресурсом;
- в соответствии с деревом выполнения (построенным планировщиком) супервизор посредством адаптеров выполняет все подзапросы на ресурсах, получая от них данные;
- все классы, полученные от ресурсов, посредством интерпретатора остаточных запросов загружаются в объектно-реляционную СУБД, после чего над ними интерпретатором выполняется остаточный запрос;
- результат выполнения остаточного запроса возвращается пользователю;
- все данные созданные в СУБД во время сессии уничтожаются, после чего сессия закрывается.

Программа переписывания правил реализует функцию преобразования программы пользователя на языке правил над схемой посредника в программу над зарегистрированными ресурсами, представленную в канонической модели данных посредника. В дальнейшем переписанная программа обрабатывается планировщиком.

Планировщик осуществляет функцию планирования реализации программы над ресурсами. В результате формируется оптимизированный план

выполнения, включающий совокупность подзапросов к удаленным ресурсам. План имеет вид "дерева выполнения".

В среде посредников нужен компонент, способный организовать реализацию завершающей стадии выполнения программы на правилах. В качестве подобного компонента выступает интерпретатор остаточных запросов. Данный компонент по своему устройству и назначению представляет собой расширенную версию адаптера. Для функционирования интерпретатора необходима объектно-реляционная СУБД вместе с расширениями, позволяющими интерпретировать типы и классы канонической модели. Для данных целей может выступать любая объектно-реляционная СУБД, поддерживающая стандарт SQL не ниже SQL99. Этот компонент позволяет загружать и выгружать данные, которые в канонической модели представляются как набор классов. Кроме того компонент позволяет специфицировать методы и функции посредника на языке программирования используемого в СУБД (например PL/SQL, T-SQL, и др.), и выполнять их внутри СУБД над классами посредника. Наконец компонент реализует поддержку выполнения остаточных запросов, транслируя их в язык SQL. Остаточный запрос содержит в себе те операции, которые не могли быть выполнены удаленно на информационных ресурсах. Все данные в объектно-реляционной СУБД существуют в рамках сессии. После того как остаточный запрос выполнен, а результат получен пользователем, все временные данные, созданные в рамках сессии, удаляются.

Адаптеры реализуют унифицированный интерфейс доступа посредника к разнородным информационным ресурсам. Адаптер получает запрос от посредника на алгебраическом языке запросов среды посредников *Asyfs* (приложение Б). Адаптер должен вернуть результат посреднику в формате передачи данных между компонентами посредника. Адаптер может полностью выполнить запрос на ресурсе, либо реализовать его частично. К примеру, если ресурс не умеет выполнять какие-то операции (*join*, *select*, *project* и т.д.) то эти

операции могут быть выполнены в адаптере, прежде чем результат будет передан посреднику. Кроме того, если ресурс поддерживает выполнение каких-то встроенных процедур и функций, то они могут быть оформлены при регистрации как методы объектов в ресурсе, и вызываться адаптером.

Адаптеры сервисов позволяет использовать уже существующие сервисы обработки данных в виде функций посредников. Недостатком этого вида адаптеров является увеличение накладных расходов на передачу данных между инициатором вызова функции (адаптером) и сервисом. Другим видом подобных адаптеров является программируемый адаптер, в котором функция реализуется не существующим удаленным сервисом (веб-сервисом, грид-сервисом), а функцией на ЯП. Фактически это означает, что реализация функции обработки будет находиться на адаптере, который обслуживает некоторый информационный ресурс. Поэтому такие адаптеры и называются программируемыми. Тем самым осуществляется приближение реализации функции к фактическим данным, что значительно повышает производительность.

1.5. Пример решения задачи в среде предметных посредников

1.5.1. Описание задачи определения вторичных стандартов

Приведенное в настоящем разделе определение задачи в среде посредников является примером реальной задачи в области астрономии, решаемой в рамках виртуальных обсерваторий [39 - 41]. Задача заключается в определении вторичных стандартов для фотометрической калибровки оптических компонентов космических гамма-всплесков [42]. Задача поставлена ИКИ РАН и сформулирована в виде текстовой спецификации (ТЗ). Этот пример будет использован в дальнейшем во всех последующих главах диссертации.

Задача определения стандартов рассматривается как одна из задач, решаемых с помощью предметных посредников, позволяющих задавать

определение прикладных областей для формулирования и решения классов научных задач в терминах понятий этих областей, структур информационных объектов, декларативно объявляемых сервисов и процессов. Посредники располагаются между исследователями, формулирующими задачи в терминах посредников, и разнообразными распределенными информационными ресурсами (данными, сервисами, процессами), необходимыми для решения задачи.

Решение задачи определения стандартов выполняется в рамках архитектуры предметных посредников (Рисунок 1.1.). Процесс решения задачи состоит из следующих основных этапов:

- построение глоссария предметной области;
- построение онтологии предметной области и онтологических контекстов ресурсов [43];
- создание схемы посредника [44];
- поиск и регистрация (построение взглядов) ресурсов релевантных задаче;
- программирование посредников на основании схемы и взглядов для зарегистрированных ресурсов;
- задание программ на языке программирования и языке правил посредника;
- рассредоточение алгоритма решения задачи;
- выполнение алгоритма решения задачи.

Задача определения стандартов заключается в том, что по координатам гамма-всплеска, необходимо отобрать ряд стандартных звезд (звезд с хорошо изученными параметрами). Изначально заданы координаты ($queryRA$, $queryDE$) гамма-всплеска. Необходимо в некоторой площадке вокруг всплеска (задается радиусом $radius$) найти астрономические объекты (звезды), которые удовлетворяют ряду параметров.

В соответствии с анализом ТЗ были выявлены основные понятия и их связи необходимы для решения задачи. Список понятий включает:

- координаты экваториальные (CoordEQJ);
- фотометрическую систему (PhotometricSystem);
- фотометрическую полосу (Passband);
- магнитуду в некоторой фотометрической системе (Magnitude);
- абстрактный Астрономический объект (AstronomicalObject);
- звезду (Star);
- стандарт (Standard);
- изображение (Image).

Также были выявлены необходимые методы и функции:

- метод кросс-идентификации (matchObjects);
- метод вычисления цветового индекса (colorIndex);
- метод проверки типа объекта по некоторому эталонному каталогу (каталогам) (checkType);
- метод проверки, является ли звезда переменной на основе данных из многих других ресурсов (isVariable).

Схема посредника для астрономической задачи определения вторичных стандартов для фотометрической калибровки оптических компонентов космических гамма-всплесков представлена на рисунке 1.3. После описания схемы посредника были определены астрономические ресурсы, релевантные решаемой задаче. Каталоги SDSS, USNOB-1, 2MASS, GSC, UCAC – основные ресурсы, используемые для извлечения стандартов. Именно среди этих каталогов отбираются все звезды, удовлетворяющие параметрам, описанным в ТЗ. Каталоги VSX, ASAS, GCVS, NSVS – используются для проверки факта переменности выбранных стандартов. Эти ресурсы были зарегистрированы в посреднике, и получены соответствующие взгляды.

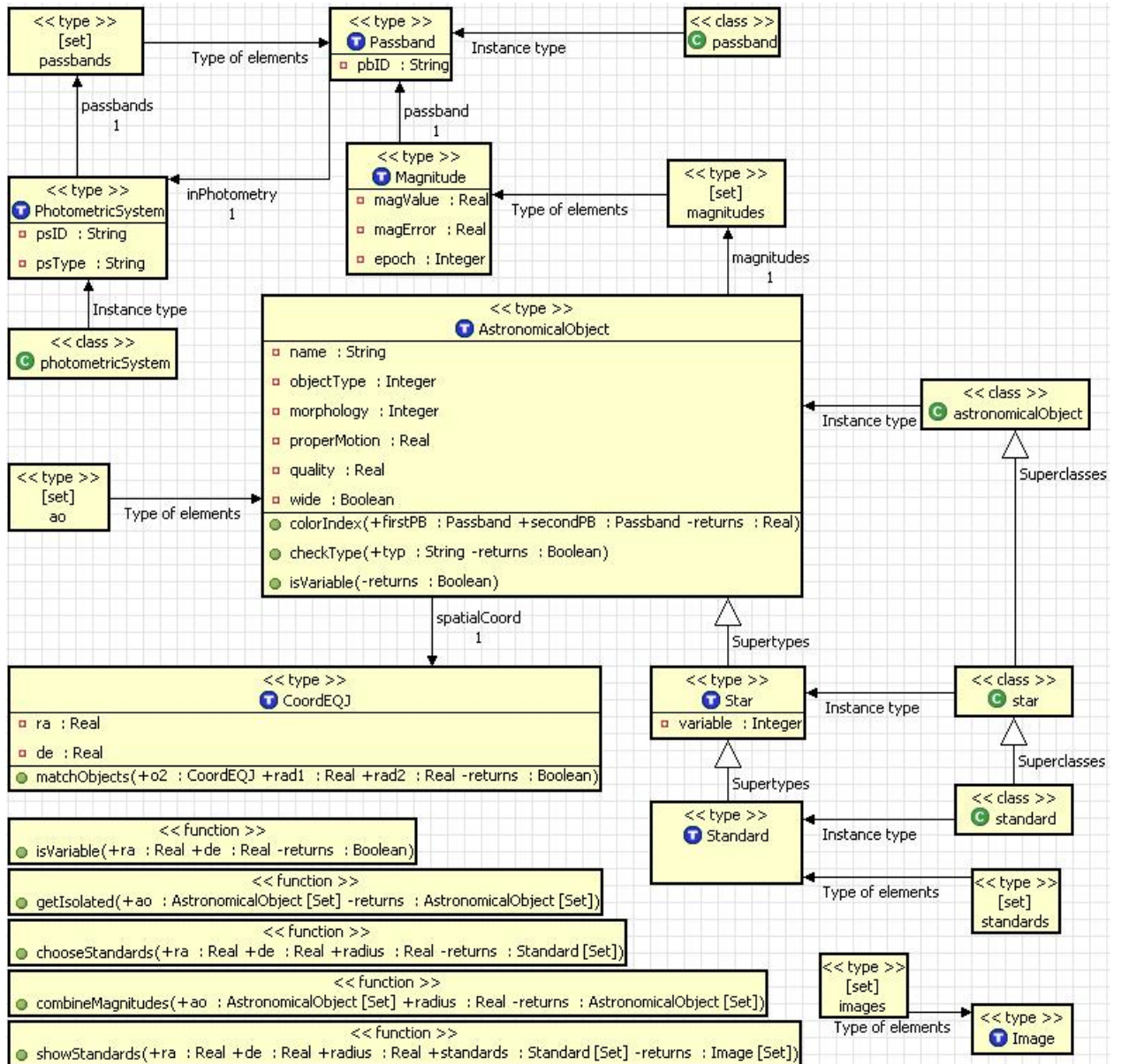


Рисунок 1.3. Схема посредника для задачи определения вторичных стандартов

Ниже представлен пример взглядов для каталога USNOB1:

```

v_USNOB1_Data(x/[ra, de, name, magnitudes, quality, objectType, properMotion])
:- CATALOG_USNOB1.catalogUSNOB1(x/[ra:RAJ2000, de:DEJ2000, name:@'USNO-B1.0', pmRA, pmDE, B1mag,
R1mag, B2mag, R2mag, Imag, B1sg, R1sg, B2sg, R2sg, Isg])
& I_FUNC.formMag(B1mag, 0, 'B', B1)
& I_FUNC.formMag(R1mag, 0, 'R', R1)
& I_FUNC.formMag(B2mag, 0, 'B', B2)
& I_FUNC.formMag(R2mag, 0, 'R', R2)
& I_FUNC.formMag(Imag, 0, 'I', I)
& I_FUNC.form0Mags(mags0)
& I_FUNC.addMag2Mags(B1, mags0, mags1)
& I_FUNC.addMag2Mags(R1, mags1, mags2)
& I_FUNC.addMag2Mags(B2, mags2, mags3)
& I_FUNC.addMag2Mags(R2, mags3, mags4)
& I_FUNC.addMag2Mags(I, mags4, magnitudes)
& I_FUNC.usnob1GetObjectTypes(B1sg, R1sg, B2sg, R2sg, Isg, objectType)
  
```

```

& I_FUNC.usnob1CheckColorIndex(B1mag, R1mag, B2mag, R2mag, acceptable)
& I_FUNC.usnob1getProperMotion(pmRA, pmDE, properMotion)
& acceptable = true
& id(0, quality)
& R1mag > 12 & R1mag < 18
& R2mag > 12 & R2mag < 18

Views.v_USNOB1_Data(x/[ra, de, name, magnitudes, quality, objectType, properMotion])
:- astronomicalObject(x/[ra: spatialCoord.ra, de: spatialCoord.de, name, magnitudes, quality,
objectType, properMotion])

```

Задача определения стандартов была сформулирована в виде программы над схемой, рассмотренной выше. Параметром программы является площадка на небесной сфере, в которой произошел гамма-всплеск. Площадка характеризуется центром с координатами *queryRA*, *queryDE* и радиусом *radius*. Программа посредника состоит из девяти последовательных правил.

Правило 1 – В первом правиле среди всех астрономических объектов выбираются те, что попадают в указанную площадку. При этом нас интересуют только координаты (*ra*, *de*), звездные величины в различных полосах (*magnitudes*), тип объекта (*objectType*), собственное движение (*properMotion*) и качество данных (*quality*). Это правило на языке правил посредников (язык СИНТЕЗ [8]) выглядит следующим образом:

```

r(x/[ra, de, name, magnitudes, objectType, properMotion, quality])
:- astronomicalObject(x1/[ra: spatialCoord.ra, de: spatialCoord.de, name, objectType,
properMotion, quality, magnitudes])
& ra < queryRA + radius & ra > queryRA - radius
& de < queryDE + radius & de > queryDE - radius

```

Правило продуцирует коллекцию *r*, состоящую из астрономических объектов (*astronomicalObject*), содержащих необходимые атрибуты и удовлетворяющих ограничениям на координаты, указанные в теле правила.

Правило 2 – Во втором правиле конструируются объекты, содержащие информацию о звездных величинах из всех возможных ресурсов. Для этого производится кросс-идентификация, заключающаяся в сопоставлении объектов из разных ресурсов между собой. Алгоритмически операцию можно свести к соединению по условию (совпадение координат *ra*, *dec*) объектов из разных ресурсов. Особенность этой операции в том, что координаты сравниваются с учетом погрешности, иными словами если значения отличаются меньше чем на погрешность, то эти значения считаются идентичными, и происходит

соединение. После соединения все звездные величины, представленные разными атрибутами, объединяются в одно множество. В результате получается новый атрибут содержащий множество звездных величин из всех. Таким образом, конструируемые объекты содержат как исходные данные (координаты, название объекта, тип объекта, качество данных, собственное движение), так и новый атрибут (множество магнитуд). Данная часть программы представляет собой вызов соответствующей функции:

```
combineMagnitudes (r/AstronomicalObject, r1);
```

Правило 3 – В третьем правиле отсеиваются неизолированные объекты:

```
getIsolated(r1, r2);
```

На вход функции *getIsolated* поступает коллекция *r1*, полученная на предыдущем шаге, в результирующую коллекцию *r2* попадают только изолированные объекты (в некоторой окрестности которых на небесной сфере не наблюдается других объектов).

Правило 4 – В четвертом правиле среди ранее выбранных объектов отсеиваются галактики, и выбираются звезды с очень малым собственным движением и качественными фотометрическими данными:

```
r3(x/[ra, de, name, magnitudes])
:- r2(x1/[ra, de, name, objectType, properMotion, quality, magnitudes])
& checkType(ra, de, 'Galaxy', nType) & nType = false
& objectType = Star
& properMotion < 0.01
& quality < 0.01
```

Все подходящие объекты попадают в коллекцию *r3*, определенную в голове правила. Выбираются объекты из коллекции *r2*, полученную на предыдущем шаге. При помощи функции *checkType* выбираются те объекты, тип которых не 'Galaxy' (галактика). Также проверяются условия: соответствия типу объекта (*objectType = Star*), малого собственного движения (*properMotion < 0.01*) и качества данных (*quality < 0.01*). В результате выполнения правила, получается множество кандидатов в стандартные звезды. Каждый кандидат необходимо проверить на переменность. Следующие шаги отвечают за извлечение переменных звезд.

Правило 5

В пятом правиле используются объекты, полученные в первом правиле. Среди объектов этого класса выбираются только те, для которых верно, что они переменные. Переменность определяется с помощью функции *isVariableByMagnitude*.

```
r4(x/[ra, de, name])
:- r1(x1/[ra, de, name, magnitudes])
& isVariableByMagnitudes(ra, de, isVar) & isVar = true
```

Правило 6

В шестом правиле выбираются переменные звезды из каталогов переменных звезд: GCVS, VSX, NSVS, ASAS.

```
r4(x/[ra, de, name])
:- variableStar(x1/[ra: spatialCoord.ra, de: spatialCoord.de, name])
```

Стоит отметить, что в этом правиле в голове используется тоже имя класса, что и в пятом правиле. Это означает что данные из пятого и шестого правила, объединяются операцией Union в один класс r4.

Правило 7

В седьмом правиле, производится кросс-идентификация объектов из класса кандидатов в стандарты (результат правила 4), и класса переменных звезд, посредством вызова функции *xmatch*. Алгоритм функции следующий: для каждого объекта из класса кандидатов в стандарты, выбирается ближайший к нему из класса переменных. Это расстояние добавляется отдельным атрибутом близости (*distance*). В результирующем классе мы получаем все кандидаты в стандарты, для каждого из которых, найден ближайший к нему переменный объект.

```
xmatch(r3, r4, r5);
```

Правило 8

В восьмом правиле из класса кандидатов в стандарты, полученного после кросс-идентификации, выбираются только те объекты, для которых не нашлось близко расположенного переменного объекта (*distance > 0.01*). На практике, это означает что кандидат в стандарты – не переменный объект.

```
r6(x/[ra, de, name magnitudes])
:- r5(x1/[ra, de, name, magnitudes, distance])
& distance > 0.01
```

Правило 9 – В предыдущем правиле построена коллекция *r6*, содержащая стандартные звезды. В заключительном правиле стандарты маркируются на изображение площадки гамма-всплеска, и предоставляются пользователю для утверждения.

```
r7(im/Image)
:- r6(x/ra, de, name, magnitudes])
& showStandards(ra, de, radius, magnitudes, im)
```

В результирующую коллекцию изображений *r7* попадают изображения площадки с заданными координатами *queryRA*, *queryDE* и радиусом *radius*, на которых промаркированы кандидаты из коллекции *r6*, полученную на предыдущем шаге.

1.5.2. Варианты оптимизации алгоритма решения задачи

Вариант реализации задачи далеко не единственный. В данном варианте вообще не использовались возможности СП. В то время как на шаге 2 функция *combineMagnitudes* реализована с помощью программируемого адаптера, который был приближен не к ресурсам, а к самому посреднику, т.к. она выполняется уже после того как данные из ресурсов собраны. С другой стороны эту же обработку данных можно было просто реализовать возможностями СП, что могло бы дать прирост производительности.

Функции во взглядах можно было реализовать иначе. Например, во взгляде, приведенном ниже, используется ряд функций разрешения конфликтов (*I_FUNC.formMag*, *I_FUNC.form0Mags*, *I_FUNC.addMag2Mags*, *I_FUNC.usnob1GetObject*, *I_FUNC.usnob1CheckColorIndex*, *I_FUNC.usnob1getProperMotion*), которые реализуются в посреднике (в СУБД посредника).

```
v_USNOB1_Data(x/[ra, de, name, magnitudes, quality, objectType, properMotion])
:- CATALOG_USNOB1.catalogUSNOB1(x/[ra:RAJ2000, de:DEJ2000, name:@'USNO-B1.0', pmRA, pmDE, B1mag,
R1mag, B2mag, R2mag, Imag, B1sg, R1sg, B2sg, R2sg, Isg])
& I_FUNC.formMag(B1mag, 0, 'B', B1)
& I_FUNC.formMag(R1mag, 0, 'R', R1)
& I_FUNC.formMag(B2mag, 0, 'B', B2)
& I_FUNC.formMag(R2mag, 0, 'R', R2)
& I_FUNC.formMag(Imag, 0, 'I', I)
& I_FUNC.form0Mags(mags0)
& I_FUNC.addMag2Mags(B1, mags0, mags1)
& I_FUNC.addMag2Mags(R1, mags1, mags2)
```

```

& I_FUNC.addMag2Mags(B2, mags2, mags3)
& I_FUNC.addMag2Mags(R2, mags3, mags4)
& I_FUNC.addMag2Mags(I, mags4, magnitudes)
& I_FUNC.usnob1GetObjectype(B1sg, R1sg, B2sg, R2sg, Isg, objectType)
& I_FUNC.usnob1CheckColorIndex(B1mag, R1mag, B2mag, R2mag, acceptable)
& I_FUNC.usnob1getProperMotion(pmRA, pmDE, properMotion)
& acceptable = true
& id(0, quality)
& R1mag > 12 & R1mag < 18
& R2mag > 12 & R2mag < 18

```

Соответственно выполняются эти функции уже после получения данных посредником от ресурсов, и некоторые операции выборки определенные в 4ом правиле (*objectType = Star, properMotion < 0.01, quality < 0.01*) и в самом это взгляде (*acceptable = true*) не могут быть выполнены на ресурсах. С другой стороны, реализация их с помощью программируемых адаптеров, позволяет выполнять операции выборки в ресурсах, что снижает объем передаваемых данных, а следовательно, повышает производительность.

Третий шаг тоже может быть оптимизирован. Функция проверки изолированности *GetIsolated* может быть разделена на две функции. Первая реализуется с помощью программируемых адаптеров и опускается к ресурсам, что сокращает объем передаваемых данных. Вторая функция реализуется, как и раньше, за тем исключением, что реализация может быть задана не в виде спецификации на ЯП (как это было исходно), а реализацией на PL-SQL процедурой в Oracle.

В задаче использовались ресурсы, с ограниченными возможностями выполнения запросов, и часть операций (а именно операция проекции *Project*) не выполнялась в адаптере. Настроив иначе адаптер, можно повлиять на планировщик, чтобы операция выполнялась в адаптере. Операция *Project* всегда уменьшает объем данных и сокращает время передачи данных.

Немалый прирост в производительности можно получить в девятом правиле. В своей реализации функция *showStandards* использует удаленный сервис для получения изображения (объем которого велик). Для повышения производительности функционал правила стоит реализовать с помощью программы на СП.

1.6. Выводы по главе

В главе представлена концепция предметных посредников, а также принципы построения сред предметных посредников и их обобщенная архитектура. Архитектура среды предметных посредников включает: уровень информационных ресурсов, уровень адаптеров, уровень предметных посредников, уровень задач. В главе также представлена архитектура исполнительных механизмов предметных посредников. Наконец, в главе приведен пример решаемой задачи, а также представлен алгоритм решения данной задачи.

Из приведенного примера видно насколько важны различные детали реализации, и как они могут повлиять на производительность в целом. При формулировании задачи с помощью посредников на основе подхода движимого приложениями, во главу угла ставится создание онтологии предметной области и схемы посредника. После чего ищутся релевантные ресурсы и также сервисы, реализующие функции. При таком подходе учесть все нюансы, связанные с производительностью, невозможно. Именно поэтому важно создание подхода, устойчивого к любым начальным условиям, и позволяющего строить эффективный и обоснованный алгоритм решения задачи (рассредоточение [45, 46]).

ГЛАВА 2. Рассредоточение реализации алгоритма решения задачи в распределенной среде предметных посредников

2.1. Постановка задачи рассредоточения в среде предметных посредников

Обобщенная архитектура среды предметных посредников для решения научных задач представлена в предыдущей главе на рисунке 1.1. в разделе 1.3. Не нарушая общности рассуждений относительно рассредоточения реализации, архитектуру среды предметных посредников можно свести к тем компонентам среды, на которых возможно осуществить рассредоточение для реализации алгоритма решения задачи. Реализация задается в виде программ и спецификаций следующих компонентов среды:

- систем программирования;
- предметных посредников;
- средств поддержки отображений классов ресурсов в классы посредников;
- адаптеров информационных ресурсов;
- конкретных информационных ресурсов.

Определение 2.1. Реализацией алгоритма решения задачи или реализацией задачи в среде предметных посредников будем называть интероперабельную композицию следующих компонентов, порождаемых спецификациями: предметных посредников, программ, порожденных системами программирования, взглядов, адаптеров, ресурсов.

Как было показано в первой главе, реализация алгоритма решения задачи может быть задана не единственным образом. Возможности каждого из компонентов ограничены их функциональными возможностями.

Определение 2.2. Функциональной операцией будем называть конструкцию программы, которой можно манипулировать в процессе построения рассредоточения как атомарной единицей.

Например, процедуры или функции в ЯП могут рассматриваться как атомарные функциональные операции с набором входных и выходных параметров.

Определение 2.3. Если функциональная операция op_1 среди входных параметров имеет выходные параметры функциональной операции op_2 , то говорят, что операция op_1 зависит от операции op_2 .

Стоит отметить, что у каждой операции может быть любое число зависимых от нее операций, и она может зависеть от любого числа операций.

Определение 2.4. Граф зависимостей функциональных операций или граф зависимостей – это ориентированный, вообще говоря, несвязный граф, без циклов, в вершинах которого расположены функциональные операции. Вершины в графе именуются также как и операции. Дуги в графе выражают зависимости операций. Если операция op_1 зависит от операции op_2 , то в графе зависимостей существует дуга, направленная от вершины op_1 к вершине op_2 .

Отсутствие циклов в графе объясняется тем, что рассматриваются не рекурсивные программы.

Реализация задачи, заданная как совокупность спецификаций на ЯП, на правилах и в виде взглядов взаимно однозначно отображается во внутреннее представление в виде графа зависимостей функциональных операций.

Определение 2.5. Назначением функциональной операции называется компонент, на котором операция специфицирована.

Для операций, заданных во взглядах, назначением являются средства поддержки отображений классов ресурсов в классы посредников. Для операций, заданных в программе на ЯП, назначением являются ЯП. Для операций, заданных в программе на правилах, назначением является язык правил предметных посредников. Если две различные операции специфицированы на одном компоненте, то говорят, что у этих операций совпадает назначение. Назначения операций, полученные при отображении

алгоритма решения задачи (в виде совокупности спецификаций) в граф зависимостей, называются начальными назначениями.

Определение 2.6. Моделью рассредоточения будем называть граф зависимостей функциональных операций, для каждой операции которого определены возможные назначения. Также в модели рассредоточения определены начальные назначения для каждой из операций.

Меняя назначения операций в модели рассредоточения можно перебирать различные варианты реализации задачи. Таким образом, модель рассредоточения определяет множество возможных вариантов реализации задачи.

Состояние модели определяется множеством назначений, состоящим из назначений для каждой операции модели рассредоточения. При этом состояние непротиворечиво, если у зависимых операций совпадает назначение.

Определение 2.7. Рассредоточенной реализацией или рассредоточением будем называть некоторое непротиворечивое состояние модели рассредоточения.

При этом рассредоточение, получаемое непосредственно из текстовых спецификаций (множество начальных назначений), называется начальным рассредоточением.

Определение 2.8. Перестановкой операции op в модели рассредоточения будем называть такое изменение состояния модели, в котором у операции op изменяется назначение по сравнению с текущим.

Переставляя операции, можно перебирать различные варианты рассредоточений. Несложно оценить и максимальное число вариантов рассредоточений. Рассмотрим случай, когда для каждой из операций модели рассредоточения доступны все назначения. Пусть дано n – количество функциональных операций в модели рассредоточения. Рассредоточение возможно между тремя компонентами – взглядами, программой посредника, ЯП. Тогда общее число вариантов рассредоточения для функциональных

операций – 3^n . В общем случае, компонентов рассредоточения может быть K , тогда вариантов реализаций – K^n .

Как было описано в разделе 1.4. первой главы, для выполнения алгоритма решения задачи, программу на правилах необходимо переписать в термины информационных ресурсов и построить план выполнения.

Эффективность рассредоточения оценивается временем, затрачиваемым на выполнение задачи. Это время включает в себя:

- время построения плана выполнения программы, включающее в себя и время переписывания – T_{plan} ;
- время выполнения плана – T_E .

Время выполнения плана T_E может быть выражено следующей формулой:

$$T_E = \sum_{n=1}^k (T_n^R + T_n^T) + T^M + T^{PL}, \text{ где}$$

k – число информационных ресурсов, T_n^R – время выполнения запроса на n -ом ресурсе, T_n^T – время передачи данных от n -ого ресурса в посредник или другой ресурс, T^M – время выполнения запроса в посреднике, T^{PL} – время выполнения программы в языке программирования. Важно отметить, что в случае если происходит многократное выполнение одного и того же рассредоточения, то план строится единожды. Так же стоит отметить, что формула для T_E приведена для случая последовательного выполнения подзапросов на ресурсах. Подобное выполнение имеет место, когда данные последовательно передаются из одного ресурса в другой. Иными словами, это худший случай. Если же все подзапросы независимы, тогда формула выглядит следующим образом.

$$T_E = \max_{n=1..k} (T_n^R + T_n^T) + T^M + T^{PL}$$

Определение 2.9. Оценкой эффективности рассредоточения будем называть функцию ET (Execution Time), существенно зависящую от рассредоточения. $ET = T_{\text{plan}} + T_E$.

Определение 2.10. Минимальным рассредоточением называется такое рассредоточение, при котором оценка эффективности рассредоточения ET будет минимальной среди всех возможных рассредоточений.

Определение 2.11. Эффективным рассредоточением называется такое рассредоточение, при котором оценка эффективности рассредоточения ET будет улучшена по сравнению с начальным рассредоточением.

Таким образом, задача данной работы заключается в поиске минимального рассредоточения для многоязычной спецификации алгоритма решения задачи в среде предметных посредников.

2.2. Описание языка правил предметных посредников и языка спецификации взглядов

В этом разделе приведено описание языка правил посредников как объектного Datalog'a, обобщающего класс подобных языков.

Правило состоит из головы правила и тела правила. Голова правила выглядит следующим образом:

$$q(x/R)$$

Где q – идентификатор коллекции, x – идентификатор переменной, а R – редукт абстрактного типа данных [8]. Редукт имеет вид: $T[r_1, \dots, r_n]$, где T – это абстрактный тип данных, а r_i – элементы редукта. Элементы редукта имеют следующий вид:

$$a: t$$

Здесь a – это идентификатор, определяющий атрибут элемента редукта, а t – это путь для атрибута элемента редукта. Путь элемента редукта выглядит следующим образом:

$$a_1.a_2. \dots a_k$$

Здесь a_1 атрибут исходного абстрактного типа данных, a_2 атрибут, содержащийся в типе атрибута a_1 , и т.д.

Тело правила – это формула, содержащая предикаты коллекций, встроенные предикаты, функциональные предикаты и предикаты условий, соединенные конъюнкциями и дизъюнкциями. Перед предикатами коллекций могут стоять отрицания. Отрицания также могут стоять и в условиях. Условие – это формула, содержащая предикаты соединенные конъюнкциями и дизъюнкциями. В условиях могут быть использованы следующие арифметические предикаты: $<$, $<=$, $=$, $>$, $>=$, $<>$. Аналогичные предикаты используются и для множеств.

Логическая программа определяется с помощью типизированной логики предикатов первого порядка. Каждая переменная должна быть типизирована встроенным типом, либо абстрактным типом данных. Коллекции, определенные предикатами коллекций, представляют собой множества объектов некоторого абстрактного типа данных.

Тип коллекции в голове правила и результирующий тип коллекции в теле правила должны быть согласованы. Формально результирующий тип экземпляра коллекции тела правила, должен быть подтипом [8] типа экземпляра коллекции головы правила. Правила определения типов формул (в частности головы и тела правила) представлены в таблице 2.1. Семантическая функция $t[^\circ]$ ставит в соответствие формуле тип. T_{aval} обозначает корень решетки типов [8], символы $|$ и $\&$ обозначают операции композиции типов $join$ и $meet$ соответственно [8].

Таблица 2.1. Правила определения типа формул

Формула	Тип	Описание
$C(T)$	T	<i>Предикат коллекции.</i>
$f(a_1, \dots, a_n, b_1, \dots, b_m)$	$\{ \text{in: type;}$ $a_1: A_1; \dots; a_n: A_n;$ $b_1: B_1; \dots; b_m: B_m;$ $\}$ Определение абстрактного типа данных, содержащего атрибуты b_j и a_i .	<i>Функциональный предикат.</i> a_i обозначают входные параметры, b_j обозначают выходные параметры, с соответствующими им типами B_1, \dots, B_m соответственно.

G	Taval	<i>Операция</i> усечения множества по условию.
F & G	t[F] t[G]	<i>Конъюнкция предикатов.</i>
F G	t[F] & t[G]	<i>Дизъюнкция предикатов.</i>

Язык правил предметных посредников – это объектный Datalog со стратифицированной семантикой и отрицанием (семантика отрицаний – NAF – negation as failure) [8].

В таблице 2.2. определены конструкции языка правил предметных посредников и их эквиваленты в логике предикатов первого порядка. Функция $s[^\circ]$ – это семантическая функция, которая ставит в соответствие конструкции языка правил терм или формулу логики. Таким образом, можно воспринимать язык правил предметных посредников как программу Datalog’а с дизъюнкциями и функциями, что позволяет определить семантику в стандартном Datalog стиле [47].

Таблица 2.2. Семантика операций языка правил

Конструкции языка правил	Семантика логики первого порядка	Описание
	AVAL	Множество значений всех типов данных.
T	$V_T \subseteq AVAL$	<i>Абстрактный тип данных (АТД).</i> АТД T определяется множеством допустимых значений V_T .
B	V_B	<i>Предопределенные типы.</i> Предопределенный тип B определяется множеством допустимых значений V_B . Например, целочисленный тип <i>integer</i> определяется множеством целых чисел $V_{integer} \subset \mathbb{Z}$.

<p>a</p> <p>b</p> <p>{ T; in: type;</p> <p>a: S;</p> <p>b: {set;</p> <p>type_of_element: S;}</p> <p>}</p>	$a \in V_T \rightarrow V_S$ $b \in V_T \rightarrow \mathbb{P}(V_S)$	<p><i>Атрибуты АТД.</i></p> <p>a, b – атрибуты АТД Т</p> <p>Атрибуты представлены функциями (get, set).</p> <p>$\mathbb{P}(V_S)$ – множество всех подмножеств V_S, т.к. атрибут b – это множество объектов типа S.</p>
C	$C \subseteq V_T$	<p><i>Коллекция.</i> Коллекция C определяется множеством объектов и является подмножеством всех значений АТД V_T.</p>
$C(v/T[a, b: c.d])$	$v \in C$ $\wedge a(v) = ?a$ $\wedge c(v) = ?c$ $\wedge d(?c) = ?b$	<p><i>Предикат коллекции с путем в элементе редукта.</i></p> <p>В логике атрибуты (a, b, c, d) типа T представлены функциями с таким же именем. Для каждого атрибута определяется новая переменная (?a, ?b, ?c, ?d).</p>
$C(v/T[a]) :- F$	$\forall v_1, \dots, v_n,$ $?a_1, \dots, ?a_m($ $\quad \exists v$ $\quad (v \in C \wedge a(v) =$ $\quad ?a) \leftarrow s[F]$ $)$	<p><i>Правило.</i> a_1, \dots, a_m атрибуты типа t[F] (тип тела правила, формулы F). $?a_1, \dots, ?a_m$ обозначают свободные переменные формулы логики s[F]. v_1, \dots, v_n обозначаются оставшиеся свободные переменные исходной формулы языка правил F.</p>
$\wedge F$	$\neg s[F]$	<i>Отрицание.</i>
$F \& G$	$s[F] \wedge s[G]$	<i>Конъюнкция.</i>
$F G$	$s[F] \vee s[G]$	<i>Дизъюнкция.</i>
$f(t_1, \dots, t_n)$	$f(s[t_1], \dots, s[t_n])$	<i>Указатель функции.</i>
v	v	<i>Переменная.</i>
c	c	<i>Константа.</i>
$t_1 \text{ op } t_2$	$s[t_1] \text{ op } s[t_2]$	<p><i>Арифметическая операция.</i></p> <p>op ::= + - * /</p>
$t_1 \text{ sigma } t_2$	$s[t_1] \text{ sigma } s[t_2]$	<p><i>Арифметический предикат.</i></p> <p>sigma ::= = < <= >= > <></p>
$t_1 < t_2, t_1 \leq t_2,$ $t_1 \geq t_2, t_1 > t_2$	$t_1 \subset t_2, t_1 \subseteq t_2,$ $t_2 \subseteq t_1, t_2 \subset t_1$	<i>Предикат множеств</i>

2.3. Функции компонентов среды предметных посредников

Целью исследований является обеспечение гибкости рассредоточений в обобщенной архитектуре среды предметных посредников. Для этого требуется введение классов (видов) функциональных операций и определение возможных корректных назначений для функциональных операций.

При работе с многоязычными программами [48] важно определить общие (для различных языков) семантические конструкторы, в которые эти программы будут отображены. В модели рассредоточения алгоритм решения задачи представлен в виде структур, называемых в работе функциональными операциями. Введение классов функциональных операций (общих конструкторов) позволяет анализировать состояние модели рассредоточения посредством применения специальных шаблонов [49]. Определение классов функциональных операций позволяет также ограничить пространство поиска при полном переборе, т.к. не все операции выразимы в каждом из языков.

В данной работе исследуется подход к построению эффективного рассредоточения для реализации алгоритма решения задачи, заданного в виде многоязычной программы на языке программирования, языке правил предметных посредников, а также на языке спецификации взглядов. В работе рассматриваются перестановки операций из взглядов в ЯП, из языка правил в ЯП, а также из языка правил во взгляды и обратно. Перестановка операций из ЯП рассматривается лишь ограниченно. Поэтому при выделении классов функциональных операций рассматриваются: язык спецификации правил предметных посредников и язык спецификации взглядов.

Для описания функциональных возможностей компонентов среды предметных посредников определим классы функциональных операций $FC = \{fc_i\}$, описывающих те функции (операции), которые могут быть описаны в среде предметных посредников. Язык правил предметных посредников рассмотрен в предыдущем разделе. Важно отметить, что выделение конкретных

классов функциональных операций, не влияет на общность рассуждений. Программа состоит из правил, поэтому для выделения классов функциональных операций, достаточно рассмотреть одно правило. Любое правило можно свести к дизъюнкции конъюнкций.

$$Q(v/T) :- \bigvee_i (C_{i1}(v_{i1}/T_{i1}) \& C_{i2}(v_{i2}/T_{i2}) \& \dots \& F_{i1}(t_{i1}, y_{i1}) \& \dots \& B_i),$$

где C_i является предикатом класса или множеством объектов заданного типа T_i , F_i – функциональным предикатом, B – ограничение, являющееся конъюнкцией предикатов над переменными v_i , типизированными типами T_i , либо над результатами функций переменными y_i . Наборы таких правил могут составлять блоки программ. Опишем классы функциональных операций (Таблица 2.3.) в том же стиле что и описан сам язык.

Таблица 2.3. Определение классов функциональных операций

Конструкции языка правил	Класс функциональных операций	Тип результата
$C(v/T[\dots])$	Первым классом функциональных операций (fc_1) являются предикаты коллекций с элементом редукта. Подобные предикаты коллекций могут быть в правиле, как в левой, так и в правой части.	$T[\dots]$
$\wedge C(v/T[\dots])$	Вторым классом функциональных операций (fc_2) является операция отрицания предиката коллекции. (C – предикат коллекций)	$T[\dots]$
$\wedge B$	Третьим классом функциональных операций (fc_3) является операция отрицания предикатов условий. B – Условие (Арифметический предикат, либо множественный предикат)	$Taval$
$F \& G$	Четвертым классом функциональных операций (fc_4) является операция конъюнкции формул (join типов экземпляров формул).	$t[F] \mid t[G]$ (join)
$F \mid G$	Пятым классом функциональных операций (fc_5) является операция дизъюнкции формул (union типов экземпляров формул).	$t[F] \& t[G]$ (meet)

$f(a_1, \dots, a_n, b_1, \dots, b_m)$	Шестым классом функциональных операций (fc_6) является функциональный предикат.	$T[b_1, \dots, b_m]$ тип с выходными атрибутами
$t_1 \text{ sigma } t_2$	Седьмым классом функциональных операций (fc_7) является операция усечения множество по условию (арифметический предикат, или предикат множеств). $\text{sigma} ::= = \mid \langle \rangle$ (для арифметически предикатов) $\text{sigma} ::= < \mid \leq \mid \geq \mid >$ (для арифметических и множественных предикатов)	Taval

Таким образом, для среды посредников были выделены 7 классов функциональных операций FC, соответствующих функциональным возможностям языков (таблица 2.4.).

Таблица 2.4. Классы функциональных операций

fc_1	класс операций предикатов коллекций
fc_2	класс операций отрицания предиката коллекций
fc_3	класс операций отрицания предиката условий
fc_4	класс операций конъюнкции формул
fc_5	класс операций дизъюнкции формул
fc_6	класс операций функциональных предикатов
fc_7	класс операций усечения множества по условию

Неоднородность выделенных классов возникает из-за неоднородности компонентов, между которыми осуществляется рассредоточение. Каждый компонент позволяет задавать лишь часть из выше описанных классов операций. Если в языке присутствуют операции, которые невозможно отнести ни к одному из классов, то такие операции обозначаются функциональным предикатом класса fc_6 . У такого функционального предиката не может быть изменено назначение по сравнению с исходным.

2.4. Представление графа зависимостей функциональных операций и модели рассредоточения

В рамках работы будет встречаться графическое представление модели рассредоточения, для понимания которого требуются пояснения. Рассмотрим упрощенный пример для задачи, описанной в первой главе (раздел 1.5.). Пусть дан один класс посредника, и в него зарегистрирован один класс ресурса. Также дана программа решения задачи. Спецификации представлены ниже:

```
//View
  v_USNOB1_Data(x/[ra, de, name, magnitudes, quality, objectType])
:- CATALOG_USNOB1.catalogUSNOB1(x/[ra:RAJ2000, de:DEJ2000, name:@'USNO-B1.0', B1mag, R1mag,
B2mag, R2mag, B1sg: '@'B1s/g', R1sg: '@'R1s/g', B2sg: '@'B2s/g', R2sg: '@'R2s/g', Isg: '@'Is/g'])
  & ProgramableA.formMag(B1mag, 0, 'B1', B1)
  & ProgramableA.formMag(R1mag, 0, 'R1', R1)
  & ProgramableA.form0Mags(mags0)
  & ProgramableA.addMag2Mags(B1, mags0, mags1)
  & ProgramableA.addMag2Mags(R1, mags1, magnitudes)
  & ProgramableA.usnob1GetObjectype(B1sg, R1sg, B2sg, R2sg, Isg, objectType)
  & ProgramableA.usnob1CheckColorIndex(B1mag, R1mag, B2mag, R2mag, acceptable)
  & id(0, quality)
  & acceptable = true
  & R1mag > 12
//Rules
1   r(x/[ra, de, name, magnitudes, objectType, quality])
   :- astronomicalObject(x1/[ra: spatialCoord.ra, de: spatialCoord.de, name, objectType,
quality, magnitudes])
   & ra < queryRA + radius & ra > queryRA - radius
   & de < queryDE + radius & de > queryDE - radius;

2   r3(x/[ra, de, name, magnitudes])
   :- r(x1/[ra, de, name, objectType, quality, magnitudes])
   & checkType(ra, de, 'G', nType) & nType = false
   & objectType = Star
   & quality < 0.01;
```

По этим спецификациям для задачи построена модель рассредоточения (рисунок 2.1.). Цветом на рисунке обозначены начальные назначения. Зеленым обозначены информационные ресурсы, серым – взгляды, желтым – классы посредника, голубым – программа посредника. На рисунке представлен ориентированный граф. Вершины изображены прямоугольниками. Вершины соединяются направленными дугами (стрелка). Каждая вершина это операция из множества функциональных операций. Стрелка обозначает зависимость между операциями. Внутри прямоугольника текстом обозначена, функциональная составляющая операции.

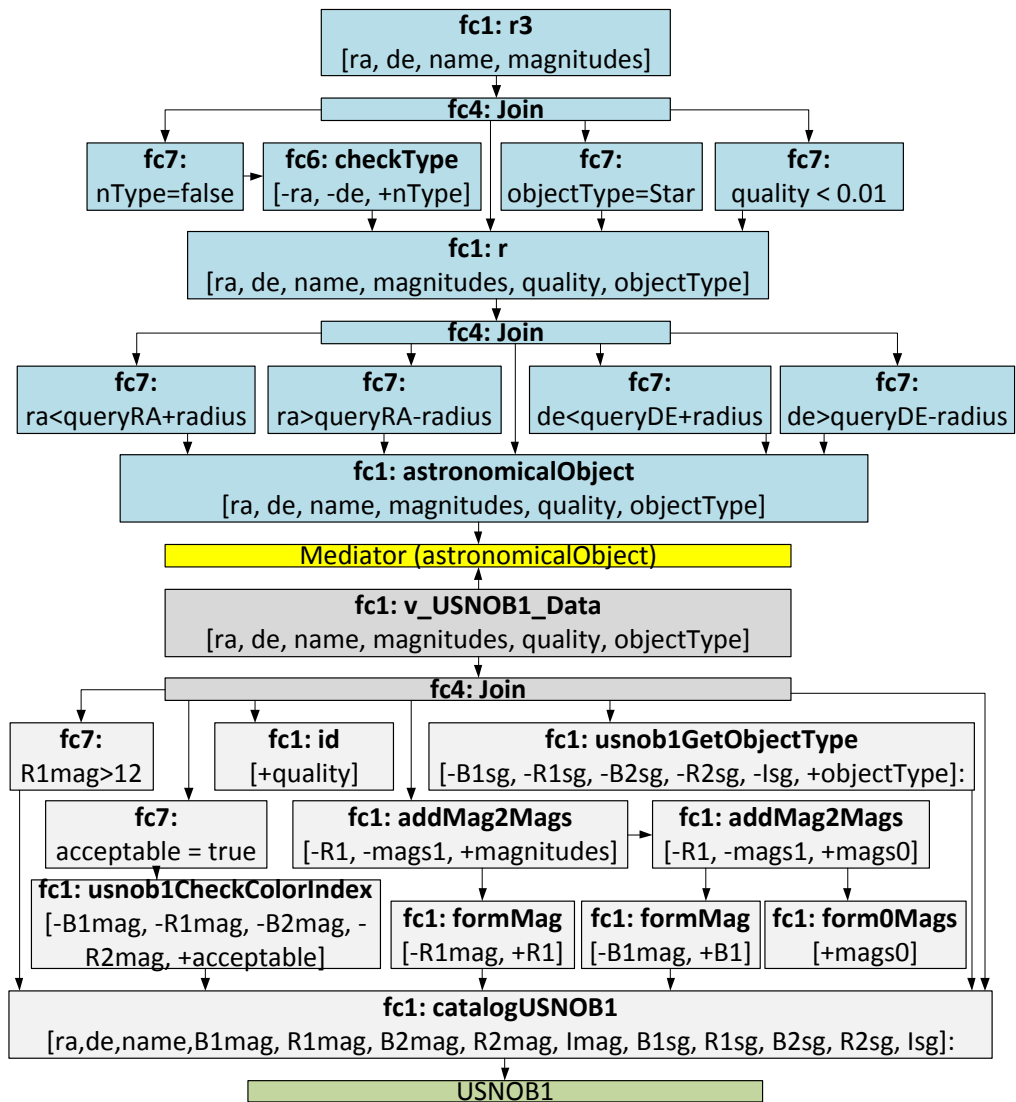


Рисунок 2.1. Модель рассредоточения упрощенной задачи

Представленная выше модель рассредоточения полностью соответствует спецификации. На рисунке 2.2. показано, что правило разбивается на элементарные операции. Каждая элементарная операции, добавляется в модель рассредоточения (справа на рисунке).

```

2  r3(x/[ra, name, de, magnitudes])
   :- r(x1/[ra, de, name, magnitudes, quality,
objectType])
   & checkType(ra, de, 'G', nType) & nType = false
   & objectType = Star
   & quality < 0.01;
    
```

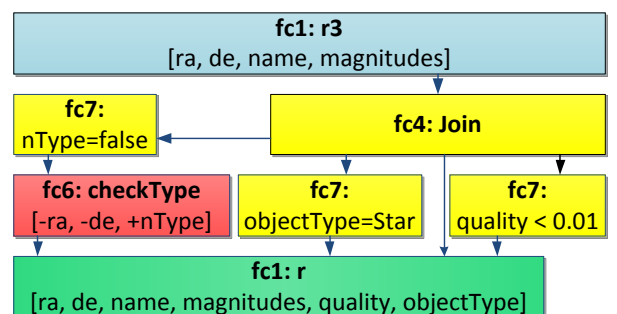


Рисунок 2.2. Пример построения модели рассредоточения для правила

В соответствии с зависимостями по атрибутам, операции соединяются дугами. В модели рассредоточения добавляется операция «Join», соответствующая операции «&» языка правил. Семантика операции аналогична семантике конъюнкции в языке правил. На примере операций усечения результирующего множества (рисунок 2.2.) видно, что условия могут быть выполнены одновременно, но тогда мы получим три множества объектов, каждое из которых удовлетворяет одному из условий. Чтобы получить итоговый результат необходимо произвести пересечение результирующих множеств. Другой вариант выполнения данных операций – это последовательное применение операций усечения множества, порядок значения не имеет. В правиле присутствует функция. Функция `checkType` выполняется для каждого объекта, и ее результат в виде атрибута `nType` присоединяется типу объектов. В данном примере операция «Join» не только производит пересечение результирующих множеств результата, но и строит *join* типов объектов. В случае если модель рассредоточения строится для нескольких правил (рисунок 2.3.), то одинаковая операция в модели рассредоточения не дублируется. В данном примере в нижнем правиле операция «*fc1: r*» является общей, с аналогичной операцией в верхнем правиле.

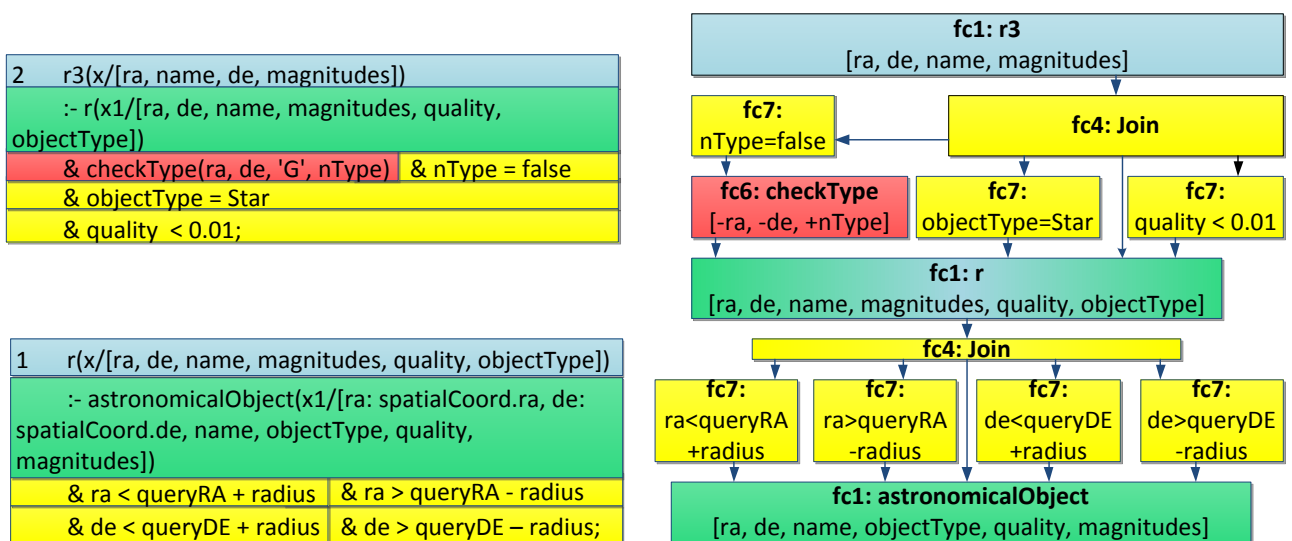


Рисунок 2.3. Модель рассредоточения для двух правил

2.5. Семантика графа зависимостей функциональных операций и модели рассредоточения

Как было описано выше, граф зависимостей включает операции. Каждая операция относится к одному из классов функциональных операций. Зависимости между операциями обозначаются дугами. Каждой операции в графе зависимостей ставится в соответствие назначение (тот компонент, где операция будет реализована). Кроме того, для функциональных предикатов определяется язык спецификации функции (ЯП, язык программирования СУБД, сервис, ресурс).

Семантика графа зависимостей соответствует потокам работ. В начальный момент могут быть выполнены только те операции, которые не зависят от других операций. На каждом последующем шаге могут быть выполнены те операции, которые зависят только от уже выполненных операций. Таким образом, возможна организация одновременного выполнения независимых операций.

Модель рассредоточения – это граф зависимостей, в котором каждой операции поставлено в соответствие множество ее возможных назначений и начальное назначение. У каждой операции может быть изменено назначение, такая операция называется перестановкой. В начальный момент программы на языке спецификации взглядов, языке правил и языке программирования, отображаются в модель рассредоточения. Рассмотрим, как это происходит.

Для каждого класса функциональных операций описаны шаблоны, определяющие то, как эти операции выглядят в каждом из языков. Именно в соответствии с данными шаблонами операции идентифицируются в исходных программах. В спецификации взглядов выделяются операции, которым найдено соответствие среди шаблонов для классов функциональных операций. Каждая такая операция добавляется в граф зависимостей с начальным назначением – взгляды. Для фрагментов спецификации взглядов, для которых не найдено

соответствующего шаблона (класса функциональных операций), в граф зависимостей добавляется операция класса fc_6 , назначенная для реализации во взглядах, и с не известным языком реализации. При этом помечается, что назначение у данной операции не может быть изменено. После этого, в соответствии с входными и выходными параметрами всех операций, вершины в графе соединяются дугами. Стоит отметить, что все операции во взглядах выражены в терминах ресурсов, за исключением операции fc_1 , которая определяет класс посредника и выходными ее параметрами являются атрибуты в терминах посредника. Именно от этих атрибутов зависят операции из правил.

После того как все взгляды рассмотрены, аналогично взглядам, в граф зависимостей добавляются все операции из правил, и из программы на языке программирования. Вершины соединяются дугами в соответствии с зависимостями операций. Для операций добавленных таким образом назначения – программа посредника и ЯП соответственно. Итоговый граф, со всеми зависимостями и назначениями для каждой операции и представляет собой модель рассредоточения. Пример построения модели рассредоточения для правил и взглядов дан в предыдущем разделе.

После этого происходит построение эффективного рассредоточения путем перестановки операций. Построение может быть автоматическим, либо полуавтоматическим, с участием эксперта.

Для выполнения рассредоточения необходимо преобразовать операции модели рассредоточения в спецификацию в виде взглядов, правил, и программы на ЯП.

Преобразование происходит последовательно, аналогично формированию модели рассредоточения. Из модели рассредоточения выделяются связные компоненты (подграфы), для каждой операции которых назначение – взгляды. Каждый выделенный таким образом подграф транслируется во взгляд, в соответствии с шаблонами, определенными для каждого класса

функциональных операций. После этого выделенные подграфы удаляются из модели рассредоточения.

Затем в модели рассредоточения выделяются связные компоненты (подграфы), для каждой из операций которых назначение – программа посредника. Каждый выделенный таким образом подграф транслируется в правило в соответствии с шаблонами, определенными для каждого класса функциональных операций. В модели рассредоточения, каждый выделенный таким образом подграф заменяется одной операцией класса fc_6 (функциональным предикатом), реализованным на ЯП, и представляющим собой вызов выполнения правила, соответствующего подграфу. Таким образом, после того как рассмотрены все подграфы с операциями, у которых назначение – программа посредника, в модели рассредоточения остаются лишь операции с назначением ЯП.

Генерация кода на ЯП происходит, начиная от независимых операций и далее по графу, в соответствии с зависимостями между операциями. Генерируемый код учитывает возможность одновременного выполнения независимых операций с помощью многопоточности и синхронизации потоков. Подробнее это рассмотрено на примере в пятой главе в разделе 5.2.

2.6. Обобщение задачи рассредоточения

Говоря о назначениях и рассредоточении, можно обобщить выше сказанное. Задача рассредоточения решается в многоязычной среде. Пусть дано K языков программирования (языков задания спецификаций). Для построения модели рассредоточения в языках должны быть выделены семантически общие конструкции, иными словами множество классов функциональных операций FC . Для каждого класса функциональных операций $fc_n \in FC$ определено то, как операция этого класса представлена в i -ом языке программирования. Фактически определен образ $Im_i(fc_n)$ выраженный конструкциями i -того языка программирования. Для пары языков i -того и j -того конструкции $Im_i(fc_n)$ в

языке i и конструкции $Im_j(fc_n)$ в языке j считаются семантически общими. Сама по себе задача выделения семантически общих конструкторов алгоритмически не разрешима. В работе считается, что это задача эксперта.

Важно отметить, что не каждая операция может быть представлена в каждом из языков. Поэтому для каждого языка программирования определяется множество выразимых в нем функциональных операций $FC_{i=1..K}$. Соответственно, в качестве назначения i -ому языку программирования могут выступать операции, класс которых содержится во множестве FC_i .

Для построения модели рассредоточения исходные спецификации на K языках программирования отображаются в граф зависимостей функциональных операций. Каждая операция графа зависимостей характеризуется своим классом из множества FC . Каждой операции ставится в соответствие ее назначение (один из K языков спецификаций). При этом проверяется ограничение: если дана функциональная операции fo класса fc_j , и назначение для нее – i -ый язык программирования, то $fc_j \in FC_i$.

Наконец, когда модель рассредоточения построена, перебираются возможные рассредоточения, и выбирается из них то, для которого время выполнения ET минимально.

2.7. Методы построения эффективного рассредоточения

Максимальная оценка общего числа рассредоточений определяется как K^n , где n – число функциональных операций в графе, а K – число языков программирования, на которых задается спецификация. Алгоритм, использованный при планировании [50], невозможно применить для функциональных операций и выбора эффективного рассредоточения вследствие естественной разнородности языков спецификации реализации задачи. Например, в среде посредников с компонентами (посредник, адаптеры) планировщик взаимодействует через унифицированный интерфейс, отправляя оценочные запросы и получая статистику. В случае же функциональных

компонентов, в среде предметных посредников подобная унификация невозможна, в силу того что рассредоточение осуществляется на уровне исходных языков.

Для построения эффективного рассредоточения были разработаны следующие методы и алгоритмы:

- метод построения модели рассредоточения;
- алгоритмы перестановки операций в модели рассредоточения;
- метод прогонки.

Метод построения модели рассредоточения включает в себя:

- построение графа зависимостей функциональных операций;
- определение начальных назначений для всех операций на основании начальной реализации;
- определение классов всех операций модели рассредоточения;
- определение возможных назначений для всех операций.

Метод прогонки – это выполнение всей реализации задачи на ограниченной выборке данных. Таким образом, можно существенно сокращать время выполнения конкретного рассредоточения.

Сама по себе задача перестановки операций не тривиальна в силу естественной разнородности компонентов назначений и языков, на которых задаются операции (язык программирования, язык правил, язык спецификации взглядов). Для перестановки операций используются три алгоритма:

- алгоритм полного перебора,
- алгоритм направленного перебора,
- алгоритм, основанный на экспертных правилах.

Построение минимального рассредоточения возможно путем полного перебора возможных перестановок операций, и оценки производительности методом прогонки. Для осуществления полного перебора необходимо

определить всевозможные варианты перебора. Для этого используется следующий алгоритм.

Вначале строятся всевозможные варианты назначений. Максимальное число таких вариантов – K^n . Затем среди всех вариантов отсеиваются те, которые не удовлетворяют корректности возможных назначений, задаваемых моделью рассредоточения. Наконец, среди оставшихся вариантов отсеиваются те, которые не удовлетворяют зависимостям операций в графе зависимостей. Когда все корректные варианты построены, для каждого выполняется прогон, после чего выбирается минимальное рассредоточение.

Алгоритм направленного перебора заключается в анализе производительности отдельных операций, а не модели рассредоточения в целом. При этом фиксируются операции усечения множества по условию (класс fc_7) и их назначения. Такие операции оцениваются совместно с операциями, продуцирующими те данные, на которые накладывается условие. Все остальные операции рассматриваются независимо друг от друга. Для каждой операции перебираются все возможные назначения, и выбирается оптимальное.

Процесс построения эффективного рассредоточения методом направленного перебора считается завершенным, когда для каждой операции, независимо друг от друга, выбрано оптимальное назначение. Таким образом, если в модели рассредоточения n операций (не включая операции условий) и для каждой операции возможно K назначений, тогда для реализации алгоритма направленного перебора, нам нужно оценить методом прогонки $K \cdot n$ вариантов, вместо K^n для полного перебора. Это существенно снижает время, затрачиваемое на построение эффективного рассредоточения, но не гарантирует построения минимального рассредоточения. Опишем данный алгоритм, на примере упрощенной задачи, представленной в разделе 2.4.

Программа для начальной оценки получается следующим образом: исключаются все операции, кроме операций условий (fc_7). Затем

выбрасываются те атрибуты, которые продуцируются выброшенными операциями (например, функциями). Затем выбрасываются условия, накладываемые на атрибуты, которые не содержатся в ресурсе. Таким образом, остается минимальная модель рассредоточения (рисунок 2.4.).

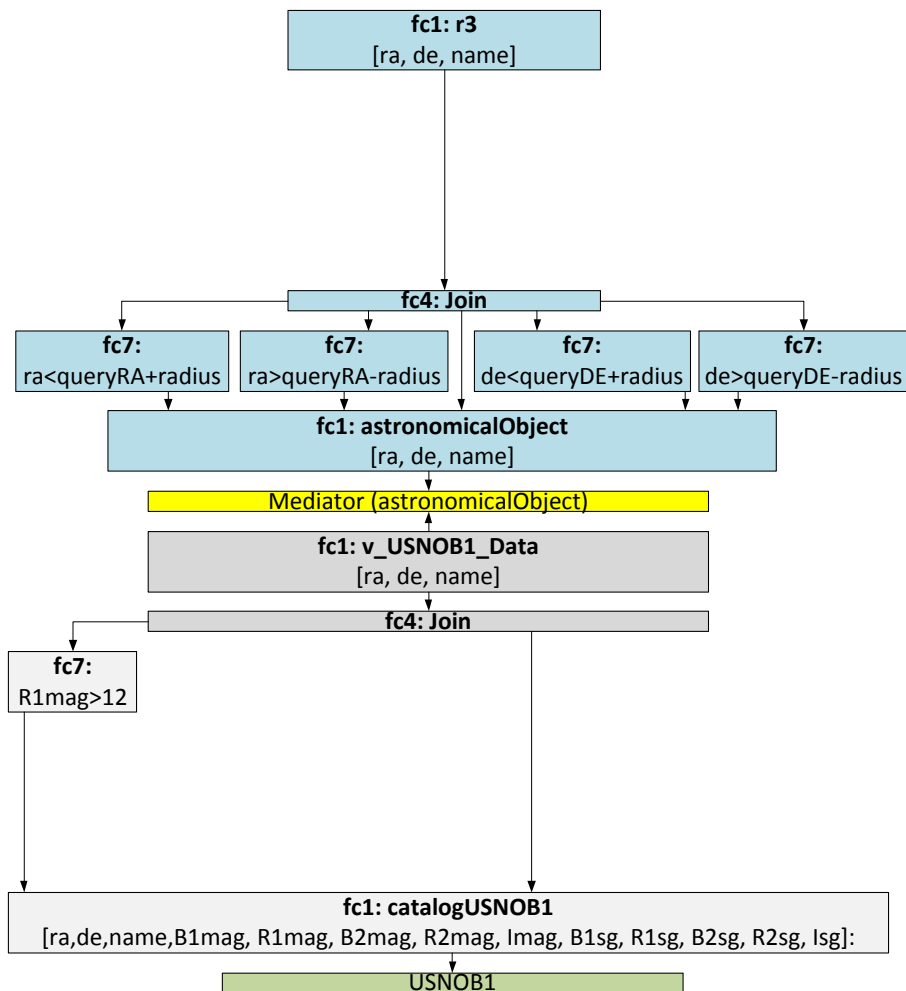


Рисунок 2.4. Минимальная модель рассредоточения

Методом прогонки оценивается время выполнения T . Также оценивается время передачи данных из ресурса в посредник N . Общее время выполнения ET – это сумма времен выполнения на каждом из ресурсов и времени затраченному на передачу данных.

Затем последовательно в минимальную модель рассредоточения добавляются другие операции (рисунок 2.5). Условия добавляются вместе с операциями, от которых они зависят.

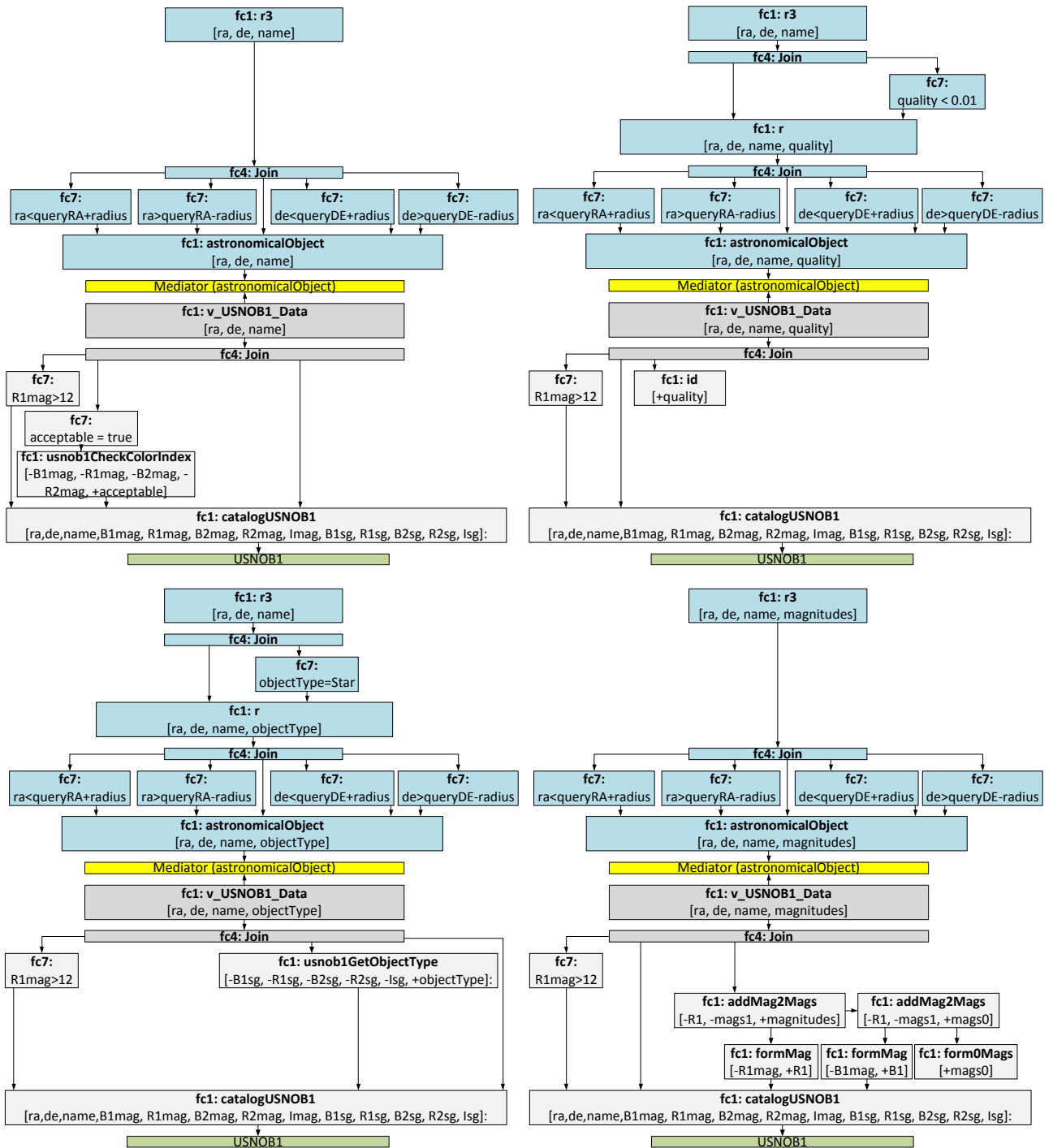


Рисунок 2.5. – Добавление операций в модель рассредоточения независимо друг от друга

Как видно на рисунке 2.5., операции добавляются в модель рассредоточения независимо друг от друга. Для каждой добавляемой операции методом прогонки оценивается время выполнения и продуцируемый объем данных. Для каждой операции оценивается каждое возможное назначение, а также язык реализации в случае функциональных предикатов. Оптимальным

для каждой операции считается то назначение, которое минимизирует общее время выполнения. Когда выбрано оптимальное назначение для каждой из операций, алгоритм считается завершенным, а построенное таким образом расщедоточение – эффективным.

Алгоритм перестановки операций на основании экспертных правил заключается в последовательном применении экспертных правил для анализа возможности перестановки каждой из операций. Примеры экспертных правил представлены в разделе 2.9. В графе отсутствуют циклы (т.к. в работе рассматриваются не рекурсивные программы). Алгоритм полуавтоматический, рассчитан на взаимодействие с экспертом. Последовательно применяются все экспертные правила, пока они меняют модель расщедоточения, улучшая ее. После чего эксперту предлагается переставить те операции, для которых во время анализа определено, что их выполнение не эффективно. Важно отметить, что построенное таким образом расщедоточение может быть и не минимальным. Оптимальным на практике, оказывается сочетание полуавтоматического подхода построения расщедоточения на основе экспертных правил и алгоритма направленного перебора.

2.8. Перестановка операций при построении расщедоточения

В данном разделе приведены результаты анализа целесообразных перестановок операций.

Предикат коллекции, входящий в тело правила, является базовым элементом правила, его перестановка для реализации в ЯП нецелесообразна, аналогичное утверждение справедливо и для взглядов. Предикат коллекции может стоять и в голове правила. Результат, описываемый формулой в теле правила, может содержать ненужные атрибуты. Для этого в типе предиката коллекции, указываются только нужные атрибуты.

Если это правило из программы посредника, то такая операция может быть переставлена только в ЯП. Данная операция в случае ее описания в

правиле выполняться будет в интерпретаторе остаточных запросов. В случае перестановки операции в ЯП, операция будет выполняться в памяти, что не так эффективно. Стоит отметить, когда запрос включает в себя только единственную подобную операцию, выполнение в ЯП может быть более эффективным.

Операция отрицания не может встречаться самостоятельно. Отрицание всегда связано либо с предикатом коллекций, либо с предикатом условий. Переставлять операцию отрицания можно только совместно со связанной с ней операцией. Перестановка предикатов коллекций недопустима, следовательно, и предикаты коллекций с отрицаниями переставлены быть не могут.

Семантика конъюнкции различается в зависимости от операндов. Если операндами конъюнкции являются коллекции, то результат также является коллекцией, тип которой представляет собой *join* типов (описание дано в разделе 2.2). Результирующая коллекция представляет собой соединение коллекций. Семантика конъюнкции в случае, если один из операндов предикат коллекции, а второй – функциональный предикат, представляет собой присоединение атрибута. Результат функции присоединяется к коллекции в виде новых атрибутов. Атрибут присоединяется для каждого выходного параметра. Конъюнкция может быть переставлена только совместно с функциональным предикатом. Семантика конъюнкции в случае, если один из операндов предикат коллекции, а второй – операция усечения множества по условию, представляет собой ограничение множества объектов коллекции. В коллекции остаются лишь те объекты, которые удовлетворяют условию. Конъюнкция может быть переставлена только совместно с предикатом условий. Если оба операнда – предикаты коллекций, тогда семантика конъюнкции такова, что если в двух коллекциях есть атрибуты с общим именем, то выполняется соединение по общим атрибутам. Если же общих атрибутов нет, то выполняется декартово произведение. Операция может быть переставлена только в системы программирования (СП). Эффективность

перестановки может зависеть от различных обстоятельств. Например, наличие накладных расходов на передачу данных от посредника в системы программирования может играть важную роль. Объем коллекции после декартового произведения – $n*m$ вместо двух коллекций суммарного объема $n+m$, где n – кол-во объектов первой коллекции, а m – кол-во объектов второй коллекции. Поэтому может оказаться более эффективным передать две коллекции по отдельности в СП, и там уже выполнять операцию декартового произведения пусть и менее эффективным образом. Также важно отметить, что присутствие операций зависящих от обеих коллекций может повлиять на эффективность перестановки.

Дизъюнкция формул может встречаться только в правилах. Во взглядах эта операция недопустима. Соответственно перестановка данной операции возможна только в СП. Рассуждения здесь аналогичны конъюнкциям в формулах. Операция дизъюнкции формул продуцирует коллекцию объемом $n+m$. Где n – кол-во объектов первой коллекции, а m – кол-во объектов второй коллекции. Поэтому с точки зрения затрат на передачу данных по сети, нет разницы выполнять операцию в посреднике или в СП.

Перестановка методов и функций играет важную роль с точки зрения производительности. Рассмотрим различные способы реализации функций с точки зрения эффективности выполнения при решении задач.

Реализация функции в информационном ресурсе. Это один из лучших возможных вариантов реализации функций. Ресурсы поддерживают возможность загрузки данных из других коллекций и выполнения над ними встроенных функций и процедур. Такие функции выполняются достаточно эффективно. Перестановка подобной операции невозможна.

Реализация функции веб-сервисом. Если веб-сервис умеет принимать коллекцию данных и выполнять функцию над коллекцией, возвращая множество значений, то этот случай ничем принципиально не отличается от случая, когда функция реализована в информационном ресурсе. Если веб-

сервис не умеет принимать коллекции, то чтобы выполнить функцию над коллекцией данных необходимо многократно вызывать удаленный сервис. Это влечет за собой большие накладные расходы.

Реализация функции на языке программирования объектно-реляционной СУБД (PL/SQL для Oracle, T-SQL для MS SQL Server, и.т.д.).

Такие функции не могут быть переставлены в другие компоненты и всегда выполняются в интерпретаторе остаточных запросов. При этом в отдельных случаях может быть более эффективным заменить подобную реализацию реализацией на ЯП. С точки зрения эффективности, для подобных функций справедливо все, что описано для функций в информационном ресурсе. При этом стоит учитывать случай, когда остаточный запрос состоит только из данной функции. В этом случае реализация функции на ЯП позволяет избежать накладных расходов, связанных с загрузкой и выгрузкой данных из СУБД. Также функции разрешения конфликтов могут быть реализованы на ЯП в адаптере. Перестановка функций в адаптер повышает производительность, если в модели рассредоточения присутствуют операции усечения множества по условию. Проверка условия на ресурсах позволяет сократить объем данных.

Реализация функции на ЯП. Это последний способ реализации функций, и предоставляющий самые широкие возможности с точки зрения перестановки операций. Функции, реализованные на ЯП, могут выполняться в рамках СП, в интерпретаторе остаточных запросов, а также на любом из адаптеров. При этом нельзя заранее сказать о том где функции, реализованные на ЯП, будут выполняться эффективнее.

Операции усечения множества по условию среди всех объектов выбирают только те, которые удовлетворяют заданным условиям. Таким образом, эти операции могут существенно ограничивать количество передаваемых данных по сети, повышая тем самым итоговую производительность. Ограничения могут задаваться в программе на правилах, во взглядах, а также в программе на ЯП. Тестирование показало, что с точки

зрения производительности нет разницы, задавать ли ограничения в программе на правилах или во взглядах, т.к. итоговый план выполнения эквивалентен (в случае эквивалентных операций ограничений).

2.9. Экспертные правила

В этом разделе приведены примеры экспертных правил, которые определяют перестановки операций в модели рассредоточения. В качестве условий применимости экспертных правил выступают класс функциональных операций и состояние модели рассредоточения.

Для описания правил введем обозначения. Все условия будут описываться в следующем виде. Вначале характеризуется класс операции. Затем характеризуется операция уже внутри класса (для предикатов коллекции – в голове правила или в теле, для функциональных предикатов – то, как реализована функция). Затем перечисляются различные условия, влияющие на производительность в среде предметных посредников. Наконец даются ограничения на модель рассредоточения. При описании используются следующие обозначения.

- op – рассматриваемая операция.
- (op_1, op_2) – зависимость между операциями (дуга в графе), операция op_1 зависит от операции op_2 .
- $(op_1, op_2)_x$ – зависимость между операциями (дуга в графе), операция op_1 , зависит от операции op_2 по переменной x . Иными словами, x является входным параметром операции op_1 , и выходным параметром операции op_2 .
- fc_i – класс функциональных операций, где $i = 1..7$.
- fc_6^{PL} – класс функциональных предикатов реализованных на ЯП.
- $fc_6^{PL-Wrapper}$ – класс функциональных предикатов реализованных на ЯП в адаптерах.

- $fc_6^{PL-Mediator}$ – класс функциональных предикатов реализованных на ЯП в интерпретаторе остаточных запросов.
- fc_6^{PL-SP} – класс функциональных предикатов реализованных на ЯП в системах программирования.
- fc_6^{IR} – класс функциональных предикатов реализованных на ресурсе.
- fc_6^{DB} – класс функциональных предикатов реализованных в СУБД посредника.
- fc_6^{WS} – класс функциональных предикатов реализованных на веб-сервисе.
- fc_1^H – класс предикатов коллекций задаваемых в голове правила.
- fc_1^B – класс предикатов коллекций задаваемых в теле правил.
- DM – модель рассредоточения.
- Назначения: PL – системы программирования, R – программа посредника, V – семантические отображения, взгляды.
- $A(op)$ – назначение операции op .
- $FC(op)$ – функциональный класс операции.
- $TransferTime(R, PL)$ – время передачи данных из посредника (программы на правилах – Rules) в язык программирования (PL).
- $TransferTime(R, PL) = low$ – время передачи данных мало, и не вносит лишних накладных расходов, $= high$ – время передачи велико.
- $InputParameterCollection(f) = true$ – функция оптимизирована для обработки коллекций, $= false$ – функция умеет принимать только отдельные объекты.

Примеры условий:

- $FC(op) = fc_i$ – рассматриваемая операция op принадлежит классу функциональных операций fc_i .
- $FC'(op) = fc_i$ – после выполнения экспертного правила, рассматриваемая операция op принадлежит классу функциональных операций fc_i .

- $a \in DM \ \& \ b \in DM \ \& \ (a, b) \in DM$ – операции a и b принадлежат модели рассредоточения, также в модели рассредоточения присутствует зависимость операции a от b .
- $A(op) = R$ – назначение операции op – программа посредника.
- $A'(op) = R$ – после выполнения экспертного правила, назначение операции op – программа посредника.

Экспертное правило описывается условием применимости и действием по изменению назначения, или по иному преобразованию модели рассредоточения. Условие применимости – булево условие, если оно истинно, тогда необходимо применить действия соответствующего правила. После каждого правила приведено его текстовое описание.

Rule#01

$$\forall op \in DM ($$

$$FC(op) = fc_1^H \ \& \ A(op) = R$$

$$\ \& \ \neg \exists z \in DM (z \neq op \ \& \ A(z) \in \{R, V\} \ \& \ FC(z) \in \{fc_1, fc_4, fc_5, fc_6^{WS}, fc_6^{DB}\})$$

$$\rightarrow A'(op) = PL)$$

Пусть операция op является «предикатом коллекций в голове правила», назначенная для реализации в программе посредника. Если в модели рассредоточения не существует операции z , отличной от операции op , и назначенной для реализации во взглядах или в программе посредника, и функциональный класс которой один из пяти вариантов, тогда назначение операции нужно изменить на ЯП.

Rule#02

$$\forall op \in DM ($$

$$FC(op) = fc_4 \ \& \ A(op) = R \ \& \ TransferTime(R, PL) = low$$

$$\ \& \ \neg \exists z \in DM (z \neq op \ \& \ A(z) \in \{R, V\} \ \& \ FC(z) \in \{fc_1, fc_4, fc_5, fc_6^{WS}, fc_6^{DB}\})$$

$$\rightarrow A'(op) \in \{PL, R\})$$

Rule#03

$$\forall op \in DM ($$

$$FC(op) = fc_4 \ \& \ A(op) = R \ \& \ TransferTime(R, PL) = high$$

$$\ \& \ \neg \exists z \in DM (z \neq op \ \& \ A(z) \in \{R, V\} \ \& \ FC(z) \in \{fc_1, fc_4, fc_5, fc_6^{WS}, fc_6^{DB}\})$$

$$\rightarrow A'(op) = PL)$$

Rule#04
 $\forall op \in DM ($
 $FC(op) = fc_4 \ \& \ A(op) = R \ \& \ TransferTime(R, PL) = high$
 $\& \ \exists z \in DM (z \neq op \ \& \ A(z) \in \{R, V\} \ \& \ FC(z) \in \{fc_1, fc_4, fc_5, fc_6^{WS}, fc_6^{DB}\})$
 $\rightarrow A'(op) \in \{PL, R\}$

Пусть операция op , является «конъюнкцией коллекций», назначенная для реализации в программе посредника. В модели рассредоточения не существует операции z , отличной от рассматриваемой, и назначенной для реализации во взглядах или в программе посредника, и функциональный класс которой один из пяти вариантов. Если нет накладных расходов при передаче данных от посредника в СП, тогда нужно сравнить и выбрать лучшее из назначений (ЯП, программа посредника). Если же накладные расходы присутствуют, то изменить назначение на ЯП. Если же накладные расходы присутствуют, но в модели рассредоточения существует операция z , отличная от рассматриваемой, назначенная для реализации во взглядах или программе посредника, и функциональный класс которой один из пяти вариантов, тогда нужно сравнить и выбрать лучшее из назначений (ЯП, программа посредника).

Rule#05
 $\forall op \in DM ($
 $FC(op) = fc_6^{IR}$
 $\& \ \exists z \in DM (z \neq op \ \& \ FC(z) = fc_1 \ \& \ (op, z) \in DM)$
 $\rightarrow FC'(op) \in \{fc_6^{IR}, fc_6^{PL}, fc_6^{DB}\}$

Пусть операция op , является «функциональным предикатом, реализованным на ресурсе». Если в модели рассредоточения существует операция z – предикат коллекции, не совпадающая с рассматриваемой, и от которой зависит функциональный предикат op , тогда нужно выбрать лучшую из реализаций (в ресурсе, в ЯП, в СУБД посредника) для функции op .

Rule#06
 $\forall op \in DM ($
 $FC(op) = fc_6^{WS} \ \& \ InputParameterCollection(op) = true$
 $\rightarrow FC'(op) \in \{fc_6^{WS}, fc_6^{PL}, fc_6^{DB}\}$

Пусть операция op является «функциональным предикатом, реализованным веб-сервисом». Если веб-сервис умеет принимать коллекции,

тогда нужно выбрать лучшую из реализаций (веб-сервис, ЯП, СУБД посредника) для функции op .

Rule#07

$$\forall op \in DM ($$

$$FC(op) \in \{fc_6^{WS}, fc_6^{PL}\} \& A(op) = R$$

$$\& InputParameterCollection(op) = false$$

$$\rightarrow FC'(op) = fc_6^{PL} \& InputParameterCollection(op) = true)$$

Пусть операция op , является «функциональным предикатом, реализованным веб-сервисом или на ЯП», назначенная для реализации в программе посредника. Если функциональный предикат не умеет принимать коллекции, тогда нужно заменить реализацию на ЯП, умеющую принимать коллекции.

Rule#08

$$\forall op \in DM ($$

$$FC(op) \in \{fc_6^{PL}, fc_6^{DB}\}$$

$$\& \exists z \in DM (z \neq op \& FC(z) = fc_7 \& (z, op) \in DM)$$

$$\rightarrow FC'(op) = fc_6^{PL-Wrapper})$$

Пусть операция op , является «функциональным предикатом, реализованным в СУБД посредника или на ЯП». Если в модели рассредоточения присутствует операция z «усечения множества по условию», зависящая от рассматриваемого функционального предиката op , тогда нужно реализовать программируемую функцию на ЯП в адаптере.

Rule#09

$$\forall op \in DM ($$

$$FC(op) = fc_6^{PL} \& InputParameterCollection(op) = true$$

$$\rightarrow FC'(op) = fc_6^{PL-SP})$$

Пусть операция op , является «функциональным предикатом, реализованным на ЯП». Если параметром функции является коллекция, тогда нужно реализовать функцию на ЯП в системах программирования.

Rule#10

$$\forall op \in DM ($$

$$FC(op) = fc_6^{DB}$$

$$\& \neg \exists z \in DM (z \neq op \& FC(z) = \{fc_1, fc_4, fc_5, fc_6^{WS}, fc_6^{DB}\})$$

$$\rightarrow FC'(op) = fc_6^{PL-Mediator})$$

Пусть операция op , является «функциональным предикатом, реализованным в СУБД посредника». Если в модели рассредоточения не существует операции z , отличной от рассматриваемой, функциональный класс которой один из пяти вариантов, тогда нужно реализовать функцию в посреднике на ЯП.

Rule#11

$$\forall op \in DM ($$

$$FC(op) = fc_6^{DB}$$

$$\& \neg \exists z \in DM (z \neq op \& (z, op) \in DM)$$

$$\rightarrow FC'(op) \in \{fc_6^{PL-Wrapper}, fc_6^{PL-SP}\})$$

Пусть операция op , является «функциональным предикатом, реализованным в СУБД посредника». Если в модели рассредоточения нет операции z , зависящей от рассматриваемого функционального предиката op , тогда нужно выбрать лучшую из реализаций функции (на ЯП в адаптере, на ЯП в системах программирования).

Rule#12

$$\forall op_1, \dots, op_n \in DM ($$

$$FC(op_1) = fc_7 \& (op_1, op_2)_x, \dots, (op_{n-1}, op_n)_x \in DM$$

$$\rightarrow (op_1, op_n)_x \in DM \& \neg (op_1, op_2)_x \in DM)$$

Пусть в модели рассредоточения присутствует n операций, $n \geq 2$. Пусть операция op_1 , является «усечением множества по условию». И известно, что операции последовательно зависят друг от друга по переменной x : 1ая от 2ой, 2ая от 3ей, и.т.д. Тогда транзитивным замыканием нужно добавить в модель рассредоточения зависимость операции op_1 от op_n . И удалить зависимость операции op_1 от op_2 . Фактически это правило означает, что операцию усечения множества по условию, нужно выполнять сразу же после операции, которая продуцирует данные, на которые накладывается условие.

Rule#13

$$\forall op_1, \dots, op_n \in DM ($$

$$(op_1, op_2), \dots, (op_{n-1}, op_n) \in DM$$

$$\& \forall i, j (A(op_i) = A(op_j))$$

$$\& \neg \exists z \in DM (z \neq op_i \& ((z, op_2) \in DM \mid \dots \mid (z, op_n) \in DM))$$

$$\rightarrow op_1 \cup \dots \cup op_n = op)$$

Пусть в модели рассредоточения присутствует n операций, $n \geq 2$. И известно, что операции последовательно зависят друг от друга: 1ая от 2ой, 2ая от 3ей, и.т.д. Кроме того у всех операций совпадает назначение. Если в модели рассредоточения не существует другой операции z , которая зависит от любой из операций op_2, \dots, op_n , тогда рассматриваемые n операций, можно объединить в одну.

2.10. Обзор существующих подходов

В том виде, в котором задача поставлена в настоящей работе, она ранее не рассматривалась. Тем не менее, можно выделить ряд областей, которые в той или иной степени близки к поставленной задаче. В первую очередь задача близка к подходам построения эффективных планов выполнения запросов в среде предметных посредников. Во вторых, задача близка к работам, посвященным анализу многоязычных программ (программ, заданных на нескольких языке программирования). Наконец, задача близка к вопросам оптимизации выполнения потоков работ. Рассмотрим отдельно эти различные аспекты.

В работе [51] рассматривается подход к оптимизации планов выполнения запросов, который можно обобщить и на процесс построения эффективного рассредоточения. Так, для построения рассредоточения необходимо: определить пространство поиска (возможные варианты рассредоточений), и далее использовать либо оценочные модели, либо статистику.

В работе [52] обсуждаются вопросы планирования запросов в среде предметных посредников. В качестве подхода к планированию используется общая парадигма планирования, называемая планированием посредством переписывания (Planning by Rewriting – PbR) [53]. В работе [52] оцениваются параметры выполнения запроса для построения эффективного плана. Главная идея подхода заключается в том, что строится начальный план, а затем данный план улучшается посредством применения набора описанных декларативно

правил. Аналогичный подход используется в настоящей работе при поиске рассредоточения на основании экспертных правил.

В работе [54] исследуется проблема построения плана выполнения запроса, в частности, особая роль уделяется проблеме выполнения операции соединения в среде предметных посредников. В [54] используется простая оценочная модель, подсчитывающая общее число подзапросов, что дает весьма общую оценку. В подходе поиска минимального рассредоточения в качестве оценочной модели используется более объективная – временная характеристика. В работе [54] предлагается два алгоритма построения плана. Первый – «жадный» алгоритм, строит быстро достаточно эффективный, но возможно не самый оптимальный план. Второй алгоритм существенно сложнее, строит оптимальный план, но за существенно большее время. Аналогичный подход применен при построении минимального рассредоточения. Используются два алгоритма для построения рассредоточения. Первый – алгоритм направленного перебора строит эффективное рассредоточение, но не всегда минимальное. Второй – алгоритм полного перебора строит всегда минимальное рассредоточение.

В работе [55] исследуется проблема оптимизации запросов. В ней предполагается, что применение подхода на основе оценочной модели не всегда возможно, т.к. нет доступа к статистике ресурсов и сложно оценить производительность ресурсов, поэтому предлагается кэшировать статистическую информацию реальных запросов к информационным ресурсам. Подобный подход применен при построении минимального рассредоточения. На основании статистической информации были выявлены экспертные правила для построения рассредоточений. Кроме того статистика выполнения отдельных подзапросов накапливается на адаптерах, и учитывается планировщиком при выборе того или иного варианта реализации.

В работе [48] обсуждается проблема анализа программ в многоязычных системах. Проблема возникает в связи с тем, что программа может быть

специфицирована одновременно на разных языках, при этом возможности языков могут зачастую пересекаться. Наиболее распространенный подход для анализа программ на разных языках – это отображение программы во множество общих структур, в рамках которых уже осуществляется анализ. В различных проектах в качестве общих структур используются ERM-модели, либо много чаще используются концептуальные графы [56, 57]. Недостатком подхода является его ориентация на анализ конкретных программ. Иными словами – общие структуры языков в виде концептуального графа строятся для конкретных программ, а не для языков в целом. При поиске минимального рассредоточения для работы с многоязычной спецификацией алгоритма решения задачи используется граф (модель рассредоточения). При этом важным моментом является то, что классы функциональных операций не зависят от конкретной реализации алгоритма решения задачи, а зависят лишь от исходных языков.

Наконец, задача построения рассредоточения близка к работам по оптимизации потоков работ [49], представленных графами, т.к. минимальное рассредоточение строится путем перестановок операций в графе (модели рассредоточения). В работе выявлен важный принцип подобных перестановок. При перестановке операций в графе важно доказать эквивалентность преобразований. Сам подход, рассматриваемый в работе [49], представляет собой последовательное применение правил оптимизации графа, что близко к применению экспертных правил при построении рассредоточения, за тем исключением, что экспертные правила могут учитывать не только состояние модели рассредоточения, но и другие условия.

2.11. Выводы по главе

В главе определяются понятия, необходимые для постановки задачи рассредоточения, среди которых важную роль играют: модель рассредоточения,

рассредоточенная реализация или рассредоточение, а также минимальное рассредоточение.

В главе представлено описание языка правил, на основании которого для обобщенной среды предметных посредников определены классы функциональных операций. Представлено описание и семантика графа зависимостей операций. Для перестановки операций в модели рассредоточения, а также для построения минимальной реализации разработаны специальные методы и алгоритмы, рассматриваемые в этой главе.

Подход к построению модели рассредоточения позволяет строить саму модель, а также определять возможные назначения для операций. Метод прогонки позволяет оценивать эффективность рассредоточения, ограничивая процент выборки данных. Алгоритм перестановки операций на основе оценки эффективности, позволяет строить минимальное рассредоточение, используя полный перебор. Алгоритм направленного перебора позволяет строить эффективное рассредоточение, существенно снизив пространство поиска по сравнению с полным перебором. Алгоритм перестановки операций на основе экспертных правил позволяет переставлять часть операций автоматически. Также алгоритм позволяет определить проблемные операции, на которые приходится существенная часть накладных расходов. Для них предлагаются пути оптимизации для окончательного принятия решения экспертом.

В главе представлен анализ целесообразности перестановки различных операций в среде предметных посредников, и приведен пример набора экспертных правил, используемых для автоматизации построения рассредоточения.

Выполнение рассредоточенной программы может происходить на ресурсах управляемых адаптерами, на самих адаптерах, в компонентах посредника (представляющих собой особый вид адаптера), а также в системах программирования. Поэтому в связи с задачей рассредоточения возникают две вспомогательные задачи. Первая – задача адекватного проблеме

рассредоточения связывания языков программирования с языками правил посредника [58]. Системы программирования являются одним из компонентов для назначений, а значит должны быть средства, позволяющие выполнять операции в СП. Вторая задача связана с построением адаптеров [38]. Ресурсов множество, поэтому необходим общий подход к конструирования адаптеров. Кроме того, важным является вопрос выполнения функций приложения на адаптерах (т.н. программируемые адаптеры). Стоит отметить, что интерпретатор остаточного запроса также представляет собой расширенную версию адаптера для объектно-реляционной СУБД. Поэтому все, что справедливо для адаптеров, справедливо также и для интерпретатора остаточных запросов.

ГЛАВА 3. Сопряжение языков программирования с декларативным языком правил предметных посредников

3.1. Краткая характеристика задачи сопряжения

Задача сопряжения языков программирования (ЯП) с декларативным языком правил предметных посредников возникает в связи с тем, что выполнение рассредоточенной программы может происходить на ресурсах управляемых адаптерами, на самих адаптерах, в компонентах посредника (представляющих собой особый вид адаптера) а также в системах программирования. Необходим подход к обоснованному и эффективному сопряжению среды предметных посредников с языками программирования. Эта задача важна, поскольку некоторым операциям в качестве назначения может быть определена система программирования. Кроме того реализация функций может задаваться на ЯП. Наконец алгоритм решения задачи, может изначально включать в себя программу на ЯП.

Несмотря на многочисленные исследования в области интероперабельных архитектур, сопряжения ЯП с базами данных и существующие стандарты (стандарт ODMG 3.0 [59], сопряжения C++ с Oracle (OCCI) [60], сопряжение Java с БД (JDBC) [61], стандарт SQLJ для встраивания языка SQL в ЯП Java [62, 63], стандарт Sun JDO [64], Microsoft LINQ [65] и др.), в подходах к проектированию и созданию подобных сопряжений рассматривались лишь отдельные аспекты сопряжений. Это затрудняет оценку сравнения качества различных подходов к сопряжению, а также создание сопряжения для новых языков и систем.

Важной частью работы по сопряжению является предложение системы характеристик, которыми можно было бы характеризовать и оценивать разнообразные средства сопряжения ЯП с системами управления информационными ресурсами. Основным мотивом разработки такой

систематизации является необходимость обоснованного определения адекватной архитектуры сопряжения средств поддержки предметных посредников с языками программирования. При такой систематизации мы ограничимся классом объектно-ориентированных языков.

3.2. Описание проблемы несоответствия импеданса при сопряжении языков запросов к базам данных и языков программирования

Проблемы несоответствия импеданса подробно описаны в статье [66]. Наиболее важные проблемы описаны ниже.

- **Проблема Синтаксиса («Синтаксис»)**. Программист должен одновременно работать с двумя разными (синтаксически) языками. Некоторые одинаковые символы, могут означать разное в разных языках, например, в Java символ “=” означает присваивание, в то время как в SQL символ “=” означает сравнение.
- **Проблема Типов («Типы»)**. Типы в ЯП и в языке запросов могут сильно отличаться. Коммутативное отображение типов может оказаться невозможным или неэффективным. Эта проблема актуальна как для простых типов, так и для сложных, абстрактных типов данных.
- **Проблема механизмов связывания («Связывание»)**. Язык запросов основан на динамическом (времени выполнения) связывании имен в запросе, в то время как в языках программирования используется статическое (времени компиляции) связывание.
- **Проблема пространства имен («Имена»)**. Пространства имен в языке запросов и языке программирования различаются. Например, мы не можем использовать переменные из запроса в языке программирования и переменные ЯП в языках запросов. При этом важной остается задача параметризации запросов. Также важно наличие возможности

использования результата запроса как переменной языка программирования.

- **Проблема коллекций («Коллекции»)**. Коллекции в базах данных и языках программирования семантически отличаются. В языках программирования возможности коллекции сильно ограничены. Коллекции, возвращаемые в качестве результата запросов, не имеют явного отображения в языках программирования, и должны обрабатываться специальными конструкциями со своим синтаксисом и семантикой.
- **Проблема долговечности объектов («Долговечность»)**. Языки запросов оперируют долговечными данными, в то время как языки программирования обычно оперируют кратковременными данными (хранящимися в памяти). Объекты в языках программирования, получаемые из базы данных (БД), имеют образ в базе. Таким образом, изменения в объектах должны быть отражены и в БД.
- **Проблема запросов и выражений («Запросы»)**. Некоторые запросы и выражения синтаксически могут выглядеть идентично и при этом иметь разную семантику. Например, в языках запросов $2+2$ – это запрос, в то время как в языках программирования – это выражение. Запрос не может быть параметром функции, в то время как выражение может.
- **Проблема ссылок («Ссылки»)**. Для изменения данных в БД нужны ссылки на данные в базе. В языках запросов результат – это таблица, а не ссылка. Таким образом, в ЯМД требуется поддержка ссылок на данные в БД, или же должны предоставляться иные возможности изменения данных (например, средствами долговечных объектов).
- **Проблема рефакторинга («Рефакторинг»)**. При разработке сложных проектов важным моментом является возможность рефакторинга. Рефакторинг — процесс изменения внутренней структуры программы, не затрагивающий её внешнего поведения и имеющий целью облегчить

понимание её работы [67]. В случае, когда запросы трактуются как строки, рефакторинг не возможен.

3.3. Характеризация сопряжений ЯП с базами данных

Хотя предметом работы является сопряжение средств поддержки предметных посредников с языками программирования (ЯП), в этом разделе будет рассматриваться и анализироваться сопряжение ЯП с базами данных в силу подавляющего числа исследований, стандартов и реализаций в этой области. Общность рассуждений при этом не нарушается, т.к. предметные посредники могут рассматриваться как системы управления виртуальными ресурсами.

Модель данных в работе рассматривается как пара языков: языка определения данных (ЯОД) и языка манипулирования данными (ЯМД). Вместо ЯМД зачастую рассматривается его подмножество – язык запросов (ЯЗ). Отображение типов ЯОД в типы ЯП, включая функции и инварианты типов, должно быть коммутативным [68]. При этом требуется сохранение отношений между типами (такими, как тип – подтип). Под коммутативностью отображения здесь понимается отображение, которое сохраняет операции и информацию исходной модели данных (модели информационного ресурса) при ее отображении в целевую (модель данных ЯП). Коммутативность достигается при условии, что диаграмма отображения ЯОД (схем) является коммутативной. В частности, при доказательстве коммутативности отображения типов необходимо устанавливать факт уточнения [34] операций типов исходной модели операциями типов целевой модели [69].

Другим важным моментом сопряжения является отображение языка запросов в ЯП. Отображение должно обладать свойством включения (containment) [70]. Для поддержки статического контроля типов требуется расширение ЯП конструкциями языка запросов. В динамике же подобного расширения не требуется, и запрос можно воспринимать как строку.

Манипулирование может осуществляться как посредством изменения значений долговечных объектов, так и посредством специальных операций ЯМД. Для поддержки статического контроля типов ЯП должен быть расширен конструкциями ЯМД. В динамике этого не требуется, и операции ЯМД можно передавать СУБД в виде строки.

Важным моментом такого отображения является различие в ЯП долговечных и недолговечных типов (и их экземпляров). Поскольку в ЯП эти понятия не определены, ЯП требует расширения. Отображение классов ЯОД в коллекции ЯП требует изменения семантики коллекций (долговечность). Важным моментом такого отображения является поддержка универсальных (generic) коллекций [71]. Для поддержки статического контроля типов требуется, чтобы результирующая коллекция была универсальной (Set<Type>).

При таких предположениях множество возможных сопряжений ЯП и СУБД будем определять следующим набором ортогональных характеристик.

Рассмотрим представление конструкций ЯОД в ЯП («ЯОД-в-ЯП»).

Тип-в-Тип. Спецификация типа (класса) ЯОД представляется спецификацией типа (долговечной коллекцией) в ЯП. Пример отображения спецификации типа на SQL в спецификацию типа на Java, представлен ниже.

```
//Спецификация типа на ЯОД (SQL)
CREATE TYPE emp UNDER person AS (
  EMP_ID INTEGER,
  SALARY REAL) INSTANTIABLE NOT FINAL REF (EMP_ID)
INSTANCE METHOD GIVE_RAISE (
  AMOUNT REAL) RETURNS REAL;
CREATE TABLE emp1s OF emp;

//Спецификация типа на ЯП(Java)
Class Emp extends Person implements PersistentObject {
  private int emp_id;
  private float salary;
  //getters and setters for emp_id and salary
  public float give_raise(float amount);
}
DBCcollection<Emp> emp1s = new DBCollection<Emp>();
```

Из приведенного выше пример видно, что всем конструкциям ЯОД сопоставлены синтаксически и семантически релевантные конструкции ЯП.

Тип-в-Объект. Спецификация типа (класса) ЯОД представляется значением (объектом) в ЯП, изображающим соответствующую спецификацию типа в ЯП. Пример отображения спецификации типа на SQL в спецификацию на Java, предоставлен ниже.

```
//Спецификация на ЯОД (SQL)
CREATE TABLE customer (
  Name char(50),
  Birth_Date date)

//Спецификация объектов на ЯП(Java)
Class Attribute {String attName; String attType;}
Class DBTable {String tableName; List<Attribute> atts;}
//PL Object
table = {"customer", atts = {
  {"Name", "char(50)"}, {"Birth_Date", "date"}}}
```

Из приведенного выше примера видно, что конструкции ЯОД отображены в объекты ЯП. В ЯП определены классы атрибута, содержащего поля имени и типа, а также класс таблицы, содержащий поля имени таблицы и списка атрибутов. В ЯП создается объект table, являющийся отображением искомой спецификации на ЯОД.

ТипЯП-в-Тип. Отображение происходит от типов ЯП к типам схемы базы данных. Спецификация типа задается не традиционным способом на ЯОД, а на ЯП. По заданным спецификациям в СУБД создаются таблицы, соответствующие типам ЯП. Пример задания спецификации типа на языке C#, отображаемого в таблицы в СУБД представлен ниже.

```
//Спецификация типа на ЯП (C#)
[Table(Name="People")]
public class Person {
  [Column(DbType="nvarchar(32) not null", Id=true)]
  public string Name;
  [Column]
  public int Age;
}

//В СУБД создается следующая таблица
create table People (
  Name nvarchar(32) primary key not null,
  Age int not null,
)
```

Из приведенного выше примера видно, что тип задается на ЯП. В ЯП присутствуют специальные конструкции для описания того как будет выглядеть таблица в СУБД. По спецификации типа ЯП в СУБД создается соответствующая таблица.

Рассмотрим представление конструкций ЯЗ в ЯП («ЯЗ-в-ЯП»).

Запрос-строка. Запрос представляется строкой. В этом случае запрос в виде строки, посредством некоторого интерфейса передается на СУБД, и там уже выполняется. Проверка корректности подобных запросов осуществляется в момент исполнения в СУБД. Именно поэтому подобный вид представления запросов ведет к большому числу ошибок. Пример представления запроса строкой представлен ниже.

```
//ЯЗ - SQL, ЯП - Java
OQLQuery query = impl.newOQLQuery();
query.create(
"select t.assists.taughtBy from t in TA where t.salary > $1 and t in $2 ");
```

Запрос-объект. Запрос представляется значением (объектом) в ЯП. В этом случае в ЯП для запроса создается класс-контейнер. Класс-контейнер позволяет задавать параметризованные запросы, а также существенно снизить возможность ошибок, связанных с корректностью запроса. Пример представления запроса в виде объекта ЯП, представлен ниже.

```
//ЯЗ - декларативный JDOQL, ЯП - Java
Query q = pm.newQuery(org.jpox.Person.class, "lastName == \"Jones\" && age < age_limit");
q.declareParameters("double age_limit");
List results = (List)q.execute(20.0);
```

Запрос-в-ЯП. ЯП расширяется конструкциями языка запросов. В этом случае язык запросов встраивается в язык программирования. Проверка синтаксиса и контроль типов производятся статически, тем самым снижая ошибки во время исполнения. Пример представления запроса в виде конструкций ЯП, представлен ниже.

```
//ЯЗ - SQLJ, ЯП - Java
#sql ordIdIter = { SELECT OrderId FROM otn_deliverydetail };
while (ordIdIter.next()) {
id = ordIdIter.orderid();
gui.addToList(id);
}
```

Рассмотрим полноту отображения ЯЗ («полнота ЯЗ»).

Полная-Поддержка-ЯЗ (ППЯЗ). Язык запросов отображается полностью. Средствами ЯП предоставляется возможность выразить любой запрос, допустимый в информационном ресурсе.

Ограниченная-Поддержка-ЯЗ (ОПЯЗ). Язык запросов сильно ограничен. Зачастую, в системах сопряжения, обеспечивается только извлечение объектов одного класса по условию. При этом никакие операции с типами (соединение, декартово произведение, объединение, проекция) не поддерживаются.

Рассмотрим представление конструкций ЯМД в ЯП («ЯМД-в-ЯП»).

ЯМД-строка. Операции манипулирования объектами представляются строкой. Все, что было сказано для представления ЯЗ строкой справедливо и в данном случае.

ЯМД-в-ЯП. ЯП расширяется конструкциями языка манипулирования данными. Все, что было сказано для представления ЯЗ конструкциями в ЯП, справедливо и в данном случае.

Рассмотрим поддержку манипулирования долговечными данными в ЯП («объекты»).

Долговечные-объекты (Д-объекты). Долговечные данные поддерживаются, изменение объектов влечет за собой изменение данных в БД.

Транзиентные-объекты (Т-объекты). Долговечные данные не поддерживаются, изменение объектов не влечет за собой изменения данных в БД.

Рассмотрим поддержку универсальных (generic) коллекций («коллекции»).

Универсальные-Коллекции (У-коллекции). Универсальные коллекции вида `Set<Type>` поддерживаются.

Обычные-Коллекции (О-коллекции). Универсальные коллекции не поддерживаются.

Полезность введенной характеристики сопряжений может быть продемонстрирована, применяя ее к определению набора характеристик, достаточных для обеспечения статического и динамического контроля типов.

Для обеспечения статического контроля типов достаточно реализовать следующий набор решений (статический подход): *Тип-в-Тип*, *Запрос-в-ЯП*, *ЯМД-в-ЯП* либо *Долговечные-объекты*, *Универсальные-Коллекции*. Для динамического контроля достаточно реализовать следующий набор решений (динамический подход): *Тип-в-Объект*, *Запрос-строка*, *ЯМД-строка*, *Обычные-Коллекции*.

В таблице 3.1. представлена характеристика известных способов сопряжения по предложенным выше критериям. Подход, выбранный в настоящей работе, включен в таблицу как СИНТЕЗ.

Таблица 3.1. Характеристика известных способов сопряжения по предложенным критериям

	ЯОД-в-ЯП	ЯЗ-в-ЯП	ЯМД-в-ЯП, объекты	Полнота ЯЗ	Коллекции
ODMG 3.0	Тип-в-Тип	Запрос-строка	Т-объекты	ППЯЗ	О-Коллекции
JDO	ТипЯП-в-Тип	Запрос-строка Запрос-объект	Д-объекты	ОПЯЗ	У-Коллекции
JDBC	Тип-в-Объект	Запрос-строка	ЯМД-строка, Т-объекты	ППЯЗ	О-Коллекции
SQLJ	Тип-в-Объект	Запрос-в-ЯП	ЯМД-в-ЯП, Д-объекты	ППЯЗ	У-Коллекции
OCCI	Тип-в-Тип	Запрос-строка	Т-объекты	ОПЯЗ	О-Коллекции
LINQ	ТипЯП-в-Тип	Запрос-в-ЯП	ЯМД-в-ЯП, Д-объекты, Т-объекты	ППЯЗ	У-Коллекции
СИНТЕЗ	Тип-в-Тип Тип-в-Объект	Запрос-строка Запрос-в-ЯП	Д-объекты, Т-объекты	ППЯЗ	У-Коллекции

Ниже в таблице 3.2. описано, какие из характеристик, решают проблемы несоответствия импеданса.

Таблица 3.2. Зависимость решения проблемы несоответствия импеданса от характеристик

	Синтаксис	Типы	Связывание	Имена	Коллекции	Долговечность	Запросы	Ссылки	Рефакторинг
Тип-в-Тип		√	√ ₁ *		√ ₁ *				
Тип-в-Объект									
ТипЯП-в-Тип			√ ₁ *		√ ₁ *				
Запрос-в-ЯП	√		√ ₂ *	√			√		√
Запрос-строка									
Запрос-объект	√								√
ЯМД-строка								√	
ЯМД-в-ЯП								√	
Д-Объекты						√		√	
Т-Объекты									
У-Коллекции					√ ₂ *				
О-Коллекции									

√ - означает, что подход с данной характеристикой полностью решает проблему. √* – означает, что проблема решается только в сочетании с другой характеристикой. Из таблицы видно, что для решения всех проблем несоответствия импеданса подход к сопряжению должен обладать следующим набором характеристик: *Тип-в-Тип*, *Запрос-в-ЯП*, *Долговечные Объекты*, *Универсальные Коллекции*. Несложно заметить, что этот набор характеристик соответствует статическому подходу. Видно, что ни один из существующих подходов (Таблица 3.1.) полностью не реализует статический подход. Кроме того, из таблицы 3.2. следует, что предложенный набор характеристик полностью покрывает проблемы несоответствия импеданса, следовательно, набор характеристик достаточен.

3.4. Подход к сопряжению языков программирования с предметными посредниками

В предыдущих разделах был приведен набор характеристик для оценки подходов к сопряжению систем программирования с базами данных, а также описаны проблемы несоответствия импеданса. Было показано также, что выбранный набор характеристик (обозначенный в Таблице 3.1. как СИНТЕЗ) покрывает проблемы несоответствия импеданса. Из таблицы 3.2. следует что для того чтобы решить все проблемы несоответствия импеданса необходимо реализовать статический подход включающий в себя следующие характеристики: *Тип-в-Тип*, *Запрос-в-ЯП*, *Долговечные объекты*, *Универсальные коллекции*. В подходе, предлагаемом в данной работе, наравне со статическим подходом предлагается реализовать и динамический подход. Статический подход при всей своей привлекательности с точки зрения решения проблемы несоответствия импеданса, накладывает на разработчика сильные ограничения. Запросы в среде предметных посредников можно задавать и изменять динамически. Семантика языка правил предметных посредников такова, что тип результирующего класса определяется в соответствии с головой правила (аналогично в SQL, тип таблицы определяется атрибутами, перечисленными в Select выражении). Следовательно, тип результирующего множества заранее предсказать невозможно, а это уже противоречит статическому подходу. Именно поэтому реализуется одновременно как статический, так и динамический подход. Стоит отметить, что динамический подход используется только для классов результатов, все типы известные заранее (описанные в схеме посредника) связываются статически. Таким образом, в соответствии с таблицей 3.1. в подходе, разрабатываемом в данной работе (проект СИНТЕЗ), реализуются следующие характеристики: *Тип-в-Тип*, *Тип-в-Объект*, *Запрос-строка*, *Запрос-в-ЯП*, *Долговечные объекты*, *Транзиентные объекты*, *Полная поддержка ЯЗ*, *Универсальные Коллекции*.

3.4.1. Статический подход

Для обеспечения статического контроля типов достаточно реализовать следующий набор отображений (статический подход):

- отображение типов – спецификация типа (класса) канонической информационной модели представляется спецификацией типа (долговечной коллекцией) ЯП;
- отображение языка запросов – ЯП расширяется конструкциями языка правил предметных посредников языка СИНТЕЗ;
- отображение средств манипулирования объектами – манипулирование объектами осуществляется поддержкой долговечных объектов, изменение объектов в ЯП влечет за собой изменение объектов канонической модели;
- отображение результирующих множеств – поддерживается строгая типизация результирующих множеств с помощью generic-коллекций.

3.4.1.1. Отображение типов

Для реализации статического подхода был реализован транслятор, преобразующий спецификации на языке СИНТЕЗ в конструкции языка программирования Java. Важным моментом этого транслятора является то, что статическое отображение строится не только для типов данных, но и для спецификации классов (множества объектов), а также для инвариантов и функций, задаваемых своими пред- и постусловиями. Также транслятор позволяет строить статическое отображение и для типа результата программы к посреднику. Подробно правила отображения типов и классов языка СИНТЕЗ в язык Java описаны в Приложении В.

Для обеспечения необходимых свойств отображения информационной модели предметных посредников в модель ЯП особое внимание в подходе уделяется проведению верификации коммутативности отображения типов. Для этого семантика информационных моделей ЯП и предметного посредника

определяется формально в рамках логики первого порядка (нотация абстрактных машин [34]) и обеспечивается доказательство коммутативности отображений, демонстрируя уточнение типов посредника типами ЯП.

Технически отображение моделей реализуется в стиле MDA [72] при использовании языка ATL [73] и метамодели Ecore [74] для представления абстрактного синтаксиса отображаемых моделей. Средства поддержки языка ATL реализованы в виде встраиваемого приложения платформы Eclipse, позволяющего редактировать и исполнять трансформации моделей. В качестве средства доказательства уточнения спецификаций используется инструментальный Atelier В, поддерживающий язык спецификаций AMN (Abstract Machine Notation [34]). В соответствии с подходом создается набор образцов (система тестов), которые отображаются в AMN, где и доказывается факт уточнения. Подобный подход верификации на образцах позволяет, с некоторой долей абстрактности, утверждать, что все отображение построено корректно. Вторым важным моментом является то, что подход позволяет проводить верификацию для конкретных схем, отобразив их в AMN,

3.4.1.2. Встраивание языка правил посредника в язык программирования

Для реализации статического подхода был расширен язык программирования Java для включения возможности задания формул на языке СИНТЕЗ прямо из программы на Java. Идея заимствована из подхода SQLJ. Допускается следующая конструкция:

```
double queryRA = 216;
double queryDE = 24;
double radius = 0.02;
#SYNTHESIS
SynthClass r(x/[ra, de, magnitudes, objectType])
:- astronomicalObject(x1/[ra: spatialCoord.ra, de: spatialCoord.de, objectType])
& stars(x2/[ ra2: spatialCoord.ra, de2: spatialCoord.de, magnitudes]
& xmatch(ra, de, ra2, de2, result) & result = true
& ra < #queryRA + #radius & ra > #queryRA - #radius
& de < #queryDE + #radius & de > #queryDE - #radius
#SYNTHESIS
System.out.println(r.getLength());
```

Т.к. подобные конструкции компилятором Java не поддерживаются, то был реализован предтранслятор, который транслирует Java программу с

использованием формул на языке СИНТЕЗ, в программу на Java без каких либо вставок. При этом предтранслятор осуществляет статический (в момент компиляции) контроль имен и типов. Выше приведенный пример транслируется в следующий Java код:

```
double queryRA = 216;
double queryDE = 24;
double radius = 0.02;
SynthClass r = Mediator.submitQuery(
  "{{r(x/[ra, de, magnitudes, objectType])" +
  ":- astronomicalObject(x1/[ra: spatialCoord.ra, de: spatialCoord.de, objectType])"
  + "& stars(x2/[ ra2: spatialCoord.ra, de2: spatialCoord.de, magnitudes]"
  + "& xmatch(ra, de, ra2, de2, result) & result = true"
  + "& ra < " + queryRA + " + " + radius + " & ra > " + queryRA + " - " + radius
  + "& de < " + queryDE + " + " + radius + " & de > " + queryDE + " - " + radius
  + "}}");
System.out.println(r.getLength());
```

В данном примере важно отметить главную особенность встраивания. Переменные, описанные в языке программирования, доступны для использования в правилах языка СИНТЕЗ, для подобных обозначений используется символ #, который указывает предтранслятору, что здесь используется переменная языка программирования. Тип переменной предтранслятор берет из Java программы. Вместо переменной в правила посредника подставляются константы соответствующих типов. Корректность типа константы и правильность получаемого выражения проверяется предтранслятором, после чего формируется чистый Java код. Также стоит отметить, что в правилах языка СИНТЕЗ могут использоваться не только простые переменные (объектные типы, скалярные типы), описанные в языке программирования, но и переменные классы. Например, если в программе на Java определен класс `astronomicalObject`, то вполне допустимо выполнение такого правила:

```
double queryRA = 216;
double queryDE = 24;
double radius = 0.02;
#SYNTHESIS
SynthClass r(x/[ra, de, magnitudes, objectType])
:- #astronomicalObject(x1/[ra: spatialCoord.ra, de: spatialCoord.de, objectType])
& stars(x2/[ ra2: spatialCoord.ra, de2: spatialCoord.de, magnitudes]
& xmatch(ra, de, ra2, de2, result) & result = true
& ra < #queryRA + #radius & ra > #queryRA - #radius
& de < #queryDE + #radius & de > #queryDE - #radius
#SYNTHESIS
System.out.println(r.getLength());
```

Здесь используется не виртуальный класс посредника, а класс, определенный в языке программирования. Таким образом, вполне естественно получить класс как результат некоторого правила (программы) посредника, произвести какие-то манипуляции с ним, после чего задать новое правило к этому классу.

3.4.1.3. Использование типизированных коллекций

В предыдущем разделе описывался пример, в котором использовалась не типизированная коллекция. В соответствии с правилом:

```
#SYNTHESIS
SynthClass r(x/[ra, de, magnitudes, objectType]) :- ...
    создавался класс типа SynthClass
```

```
SynthClass r = Mediator.submitQuery(
```

`SynthClass` представляет собой коллекцию объектов (класс), тип экземпляров которой, определяется динамически (подробнее описано далее в динамическом подходе). Если же нужно использовать типизированную коллекцию, то правило следовало писать так:

```
#SYNTHESIS
Class r(x/[ra, de, magnitudes, objectType]) :- ...
```

В этом случае предтранслятор по правилу создает Java класс `Type_r`, содержащий атрибуты *ra*, *de*, *magnitudes*, *objectType*. Типы атрибутов берутся из схемы посредника, к которой задано правило.

```
public class Type_r
{
    private double ra;
    private double de;
    private Set<Magnitude> magnitudes;
    private int objectType;
}
```

Кроме того создается класс для результата *r* (подробнее см. Приложение В, раздел классы).

```
public class Cls_r
{
    private class Type_rSet extends HashSet<Type_r> implements Set<Type_r>
    {...}
    private static Type_rSet r;
    public static Set<Type_r> getSet()
    { return this.r; }
    ...
}
```

Соответствующий Java код после пред-трансляции выглядит следующим образом:

```
double queryRA = 216;
double queryDE = 24;
double radius = 0.02;
Cls_r.submitQuery(
  "{{r(x/[ra, de, magnitudes, objectType])" +
  ":- astronomicalObject(x1/[ra: spatialCoord.ra, de: spatialCoord.de, objectType])"
  + "& stars(x2/[ ra2: spatialCoord.ra, de2: spatialCoord.de, magnitudes]"
  + "& xmatch(ra, de, ra2, de2, result) & result = true"
  + "& ra < " + queryRA + " + " + radius + " & ra > " + queryRA + " - " + radius
  + "& de < " + queryDE + " + " + radius + " & de > " + queryDE + " - " + radius
  + "}}");
System.out.println(Cls_r.getLength());
```

Использование подобных статических конструкций хоть и возможно, но не всегда удобно. Более простой и естественный способ использования типизированных коллекций достигается, если в правиле к посреднику используется не два класса `astronomicalObject` и `stars`, как в ранее рассмотренном примере, а только один. Рассмотрим пример, в котором не используется один класс.

```
double queryRA = 216;
double queryDE = 24;
double radius = 0.02;
#SYNTHESIS
Class r(x/[spatialCoord, objectType])
:- astronomicalObject(x1/[ra: spatialCoord.ra, de: spatialCoord.de, spatialCoord, objectType])
& ra < #queryRA + #radius & ra > #queryRA - #radius
& de < #queryDE + #radius & de > #queryDE - #radius
#SYNTHESIS
System.out.println(r.getLength());
```

В этом случае для класса `r` не будут создаваться соответствующие типы, а будут использованы классы Java, сгенерированные для `AstronomicalObject` (подробнее в Приложении В, раздел классы). Код на Java будет выглядеть идентично, за тем исключением, что класс `Cls_r` изменится:

```
public class Cls_r
{
  private class Type_rSet extends
    Cls_astronomicalObject.AstronomicalObjectSet implements
    Set<AstronomicalObject>
  {...}
  private static Type_rSet r;
  public static Set<AstronomicalObject> getSet()
  { return this.r; }
  ...
}
```

Таким образом, тип объектов в классе `r` уже будет типа `AstronomicalObject`.

3.4.2. *Динамический подход*

Для динамического контроля типов достаточно реализовать следующий набор сопряжений (динамический подход):

- отображение типов – спецификация типа (класса) ЯОД представляется значением (объектом) в ЯП, изображающим соответствующую спецификацию типа в ЯП;
- отображение языка запросов – запрос представляется строкой;
- отображение результирующих множеств – Generic коллекции не поддерживаются.

Плюсом динамического подхода является то, что все конструкции могут строиться динамически (во время исполнения) и не зависят от исходных данных. Например, тип результирующего множества может быть произвольным, в отличие от статического подхода, где требуется явное задание типа. С другой стороны, достоинства являются также и недостатками, поскольку во многих системах безопасность играет важную роль, необходима максимальная поддержка статического контроля типов.

Для реализации динамического подхода, язык программирования был расширен конструкциями, позволяющими хранить произвольные спецификации схем, модулей, типов, классов и функций языка СИНТЕЗ в виде объектов ЯП. Также был разработан АПИ для выполнения текстовой программы на языке правил языка СИНТЕЗ в посредниках. Наконец, язык программирования был расширен для представления объектов канонической модели и произвольных классов (коллекций объектов). Также для выполнения инвариантов и проверки пред- и постусловий функций был реализован интерпретатор формул языка СИНТЕЗ. Подробно представление произвольных спецификаций языка СИНТЕЗ, представление классов объектов, инвариантов и функций описано в Приложении Г.

Идеологически динамический подход близок к стандарту JDBC, за тем исключением, что оперирует объектными данными и способен проверять инварианты, пред- и постусловия функций. В случае применения динамического подхода, спецификации языка СИНТЕЗ загружаются динамически, после чего текстовая программа на языке правил посредника может быть выполнена, а результат получен в виде класса объектов SynthClass. Таким образом, одно и то же приложение (программный продукт) может работать с любыми спецификациями посредника и/или адаптеров, а также выполнять любые запросы, без предварительной трансляции.

3.4.3. Долговечные и транзистные объекты

Посредники обеспечивают виртуальный способ интеграции неоднородных информационных ресурсов. Данные нигде не материализуются (за исключением результата). Пользователь оперирует виртуальными классами посредника, задавая программу к посреднику, после чего получает результат, образованный в результате взаимодействия с множеством ресурсов. В такой архитектуре отразить изменения, производимые в объектах, обратно в ресурсы зачастую невозможно. Поэтому все объекты получаемые пользователем транзистные, и их изменение никак не затрагивает данные в ресурсах. Тем не менее, долговечные объекты поддерживаются для уже материализованных классов. Например, если пользователь задал программу на правилах и получил результат в виде класса посредника, то все объекты этого класса являются долговечными. Таким образом, изменения в самом классе или в отдельных объектах отразятся в базе посредника, где материализуются классы.

3.5. Реализация сопряжения предметных посредников с языками программирования

Сопряжение предметных посредников с языками программирования на примере языка Java включает в себя реализацию динамического и статического связывания (рисунок 3.1.).



Рисунок 3.1. Структура компонентов связывания языка правил и языка программирования

Реализация динамического связывания включает в себя следующее:

- структуры языка программирования для представления спецификаций предметных посредников на языке СИНТЕЗ (схем, модулей, классов, типов, функций, инвариантов);
- структуры языка программирования для представления классов – коллекций объектов;
- интерпретатор формул языка СИНТЕЗ, используемый для проверки инвариантов, пред- и постусловий;
- АПИ для работы с предметными посредниками.

Структуры языка программирования Java для представления конструкций языка СИНТЕЗ и классов объектов подробно описаны в приложении Г. Стоит отметить, что для всех конструкций разработаны функции их сериализации и десериализации в текстовое представление. Для конструкций языка СИНТЕЗ текстовым представлением служит спецификация на языке СИНТЕЗ. Для классов объектов текстовое представление представляет собой разработанный XML формат, за основу которого взято представление таблиц в XML. Представление таблиц было расширено для возможности представления объектных структур. Также XML формат представления классов содержит в себе сериализованный тип класса. Анализатор языка СИНТЕЗ разработан с использованием технологии Antlr, анализатор классов в основе своей использует стандартный анализатор для XML документов.

Интерпретатор формул языка СИНТЕЗ был реализован в виде статических методов класса: *checkCondition*, *checkConditionComposition*, *checkConditionConjunction*, *checkPrePostCondition*. Все эти методы описаны в приложении Г.

Интерфейс для работы с предметными посредниками представлен ниже:

```
public interface Supervisor {
    Long openSession();
    Long executeQuery(Long sessionId, String syfsQuery);
    void uploadClasses(Long sessionId, SynthClass synthClass);
    SynthClass getResult(Long sessionId);
    QueryStatus getStatus(Long sessionId, Long queryId);
    boolean cancelQuery(Long sessionId, Long queryId);
    void closeSession(Long sessionId);
}
```

Здесь описаны методы для работы с сессиями (*openSession*, *closeSession*), метод для загрузки классов в посредник (*uploadClasses*), методы для задания программ (*executeQuery*, *getResult*, *getStatus*, *cancelQuery*).

Статическое связывание включает в себя:

- транслятор схем языка СИНТЕЗ в конструкции языка программирования на основании таблицы соответствия в Приложение В;

- предтранслятор языка JSyfs, отображающего программу на языке Java со встроенными конструкциями программы на правилах предметных посредников в чистый код Java, компилируемый Java-компилятором;
- реализацию поддержки долговечных объектов, основанную на технологии Hibernate [75].

Правила преобразования схем языка СИНТЕЗ в конструкции языка Java описаны в приложении В. На основании этих правил был разработан транслятор. В основе своей транслятор использует конструкции динамического связывания. Спецификация на языке СИНТЕЗ отображается в конструкции языка Java, после чего транслятор генерирует соответствующий Java код.

Предтранслятор JSyfs реализован с помощью библиотеки Antlr. Для Java взята готовая antlr грамматика, и расширена для представления правил языка СИНТЕЗ. Часть полученной грамматики языка JSyfs для представления языка правил предметных посредников (Syfs) описана в Приложении А.

Как было описано ранее, предметные посредники являются виртуальными, поэтому поддержка долговечности объектов, в традиционном его понимании, невозможна. Т.к. невозможно отразить изменения в объектах, обратно в ресурсы. Долговечность поддерживается для тех объектов, которые уже загружены в интерпретатор остаточных запросов посредника. При задании программы пользователь может выбрать, какие объекты получать - долговечные или транзистные. Кроме того у каждого объекта можно проверить статус, является ли он долговечным. А также любой долговечный объект вызовом простого метода может быть сделан транзистным, в этом случае связь с базой аннулируется. Реализация долговечности объектов основывается на технологии Hibernate, являющейся известным ORM инструментарием.

Важным моментом при реализации связывания является вопрос представления результата операции соединения. Операция соединения

используется, когда результирующий класс мы получаем более чем из одного класса. Рассмотрим пример правила:

```
r(x/[ra, de, name, ra1, de1])
:-radioCatalogData(y/[name, ra: spatialCoord.ra, de: spatialCoord.de])
& opticalCatalogData(x/[name, ra1: spatialCoord.ra, de1: spatialCoord.de, colorIndexURG,
deltaColorIndexURG])
```

В правой стороне выше приведенного правила стоит два предиката классов `radioCatalogData` и `opticalCatalogData`, соединенных между собой символом «&», означающим соединение. Для эффективной реализации данной операции в СП реализованы два вида представления классов.

Класс – это множество объектов некоторого типа. Тип объектов определяет тип экземпляра класса. Каждый объект характеризуется множеством значений (для каждого атрибута), в том числе и объектных значений. Логичным представлением для такой структуры является множество множеств (список списков или массив массивов). В реализации, каждый объект представлен не множеством значений, а таблицей `HashMap<ключ, значение>`, это позволяет крайне эффективно выполнять операции редукта типа. Здесь в качестве ключа используется имя атрибута, а в качестве значения произвольный объект в соответствии с типом атрибута. Атрибут удаляется только из типа класса, а физически данные не трогаются. Это влечет за собой увеличение расхода памяти, при этом существенно повышается скорость работы таких операций. Для эффективного выполнения операций соединения в классе возможны два способа хранения данных: материализованное и ссылочное. Материализованное представляет собой явное хранение множества таблиц (`HashMap`) для каждого объекта. Ссылочное представление для каждого объекта хранит не сами данные, а ссылки на них. Фактически в отдельном множестве данных хранятся все объекты с численными идентификаторами, а в другом хранится множество ссылочных объектов. Каждый ссылочный объект представлен также таблицей `HashMap<ключ, значение>`, где ключ представляет собой имя атрибута, а значение представляет собой пару <идентификатор объекта во множестве данных, имя атрибута в этом объекте>. Таким образом,

при выполнении соединения нам не нужно перестраивать множество данных, а лишь перестраивается множество ссылочных объектов. Для наглядности рассмотрим пример:

Класс А, атрибуты a,b,c.

Класс В, атрибуты d,e,f.

Пусть в классе А содержатся объекты: {1,1,1}, {2,2,2}, {3,3,3}.

Пусть в классе В содержатся объекты: {4,4,4}, {5,5,5}.

Тогда после декартового произведения (т.к. нет общих атрибутов) класс будет содержать следующие данные:

- В случае материализованного представления мы будем хранить объекты: {1,1,1,4,4,4}, {1,1,1,5,5,5}, {2,2,2,4,4,4}, {2,2,2,5,5,5}, {3,3,3,4,4,4}, {3,3,3,5,5,5}. Т.е. кол-во объектов получилось $n*m$ ($2*3$).
- В случае ссылочного представления, храниться будут объекты: {1,1,1}, {2,2, 2}, {3,3,3}, {4,4,4}, {5,5,5}. Т.е. объем данных $n+m$ ($2+3$), также еще для этих данных будут созданы ссылочные объекты. Их объем $n*m*C$ где C – коэффициент во сколько раз ссылка на объект по объему меньше самого объекта.

Например, в случае если атрибуты – целые числа как в примере, то ссылочное представление займет больше памяти, если же хранятся строки по 255 байт каждая, либо сложные объектные структуры, то ссылочное представление займет много меньше памяти. Кроме того построение класса – результата естественного соединения со ссылочным представлением в случае когда объекты занимают много памяти, значительно быстрее нежели построение обычного материализованного представления.

3.6. Обзор существующих подходов

В стандарте ODMG 3.0 [59] описывается связывание объектных баз данных с языками программирования C++, SmallTalk, Java. С точки зрения ЯОД подход относится к подходам со статическим связыванием. Конструкции

ЯОД отображаются в конструкции ЯП. С точки зрения языка запросов, подход ODMG 3.0 относится к подходам, где при связывании с ЯП запрос задается просто текстовой строкой. Результирующие объекты не типизируются, результат возвращается в виде обычной (не типизированной) коллекции. Манипуляция объектами предполагается на языке программирования.

Стандарт JDO (Java Data Objects) [64] представляет собой программный интерфейс на языке Java для поддержки долговечных объектов. Подход можно отнести к подходам с чисто динамическим связыванием. Явного отображения для ЯОД не происходит. Объекты задаются явно на ЯП, и определяют объектную модель для JDO. Манипуляция с объектами осуществляется средствами ЯП (Java). Запрос задается на языке JDOQL, поддерживающий как задание запросов строкой, так и представление запроса объектом. В JDO в качестве результата может выступать только коллекция долговечных объектов некоторого фиксированного типа. При этом в стандарте не используются универсальные коллекции, как и в ODMG.

Наиболее широкое распространение получил подход JDBC [61], при этом доступ посредством JDBC используется не только прикладными программистами, но и при реализации более сложных подходов связывания, например, при реализации JDO или SQLJ. Подход представляет собой пример чисто динамического связывания, и по сути, реализует программный интерфейс для доступа к реляционным БД из ЯП Java. Запрос передается в виде строки, строка запроса может быть параметризованной. Результат представляет собой коллекцию объектов (кортежей), каждый такой объект представляет собой массив объектов. Отображение ЯОД в подходе не рассматривается. Тем не менее, из-за своей простоты подход получил широкое распространение.

Одним из немногих подходов по сопряжению ЯП с БД, в котором реализовано встраивание языка запросов в язык программирования, является подход SQLJ [62, 63]. SQLJ представляет собой стандарт для совместного использования Java и SQL. Стандарт был разработан консорциумом компаний,

среди которых: IBM, Informix, Microsoft, Sun, Sybase и Oracle. В спецификации определены: статическое встраивание языка SQL в язык программирования Java; возможность использования Java методов, как SQL хранимых процедур на сервере базы данных; связь между типами языков, позволяющая использоваться Java классы, в качестве определяемых пользователем типов SQL. Компилятор Java не может распознать встроенный SQL. Поэтому в SQLJ используется особый предтранслятор, который заменяет все конструкции языка SQLJ на Java, после чего уже используется стандартный Java компилятор. В данном подходе все ошибки проверяются статически. Существенным ограничением подхода, является то, что поддержка методов как встраиваемых процедур и связь между типами не были реализованы.

Подход Oracle C++ Call Interface (ОСЦИ) [60] представляет собой высокопроизводительный программный интерфейс (API) для доступа к базе данных Oracle посредством языка программирования C++. ОСЦИ обеспечивает использование посредством языка C++ всех возможностей языка SQL, включая операции ЯОД, операции ЯМД, SELECT запросы. С точки зрения ЯОД подход реализует статическое связывание, т.к. для всех объектных типов, используемых в Oracle, генерируются соответствующие C++ классы. Запрос при этом передается строкой, а объекты возвращаются не типизированными, поэтому нужно явно приводить их к объектному типу.

Не так давно компанией Microsoft был предложен подход LINQ to SQL [65]. В подходе реализуется встраивание языка запросов в ЯП. В LINQ запрос встроен в базовые конструкции языка C# и Visual Basic. В LINQ to SQL модель данных реляционной базы данных сопоставляется объектной модели, выраженной в языке программирования. В подходе можно использовать запросы как для доступа к СУБД, так и к коллекциям объектов в памяти. Важным моментом подхода LINQ to SQL является то, что нет явного связывания для языков определения данных (SQL и C#). Объектная модель определяется в языке программирования, а соответствующие объектам таблицы

создаются в СУБД автоматически. С этой точки зрения подход близок к множеству подходов по объектно-реляционному отображению (ORM), среди которых JDO или Hibernate [75]. Таким образом, подход не может быть отнесен к подходам со статическим связыванием.

Подводя итог рассмотренным подходам, можно сделать вывод что ни один из подходов не решает все проблемы несоответствия импеданса. В подходе ODMG 3.0 решается только проблема *Типов*. В подходе OCCi решаются проблемы *Типов* и *Ссылок*. В подходе JDBC решается только проблема *Ссылок*. В подходе SQLJ решаются *все проблемы кроме Типов*. В подходе JDO решаются проблемы *Синтаксиса, Долговечности, Ссылок и Рефакторинга*. В подходе LINQ, как и в подходе SQLJ решаются *все проблемы кроме Типов*.

Часть рассмотренных статей были посвящены сравнению различных подходов по связыванию. Например, в статье [76] предлагается критерий оценки качества подходов. Характеристики, рассматриваемые в статье похожи на те, что рассмотрены в разделе 3.3. Более того, характеристики, рассматриваемые в статье и связанные с проблемой несоответствия импеданса, являются подмножеством характеристик рассмотренных в разделе 3.3. В некоторых подходах также как и в настоящей работе утверждается, что проблема несоответствия решена.

Основная идея подхода Sather [77] заключается в использовании структур ЯП для представления сущностей реляционной модели данных. Например, такие сущности как *База Данных* или *Отношение*. В работе предполагается, что поскольку одна и та же система типов используется и для ЯП и для представления сущностей СУБД, то проблема несоответствия импеданса может считаться решенной. Если же рассматривать проблему несоответствия импеданса как список проблем представленных в разделе 3.2., тогда в данном подходе решаются только проблемы *Синтаксиса, Долговечности, Ссылок и Рефакторинга*.

В проекте ARARAT [78] проблема несоответствия импеданса, по словам авторов, решается посредством библиотеки шаблонов. Библиотека шаблонов нацелена на безопасную генерацию SQL. Шаблоны генерируются статически для схемы базы данных. В библиотеке шаблонов содержатся классы только для представления запросов, а не для представления типов базы данных. Долговечные объекты и универсальные коллекции не поддерживаются. Подход реализует только характеристику *Запрос-объект*, и решает проблемы *Синтаксиса и Рефакторинга*.

В подходе, основанном на стеках (SBA) [79] все проблемы несоответствия импеданса решаются с помощью введения нового самодостаточного языка программирования запросов. В объектно-ориентированном языке программирования нет различия между выражениями языка программирования и запросами. Язык, разработанный в подходе, называется SBQL. В подходе отвергаются любые механизмы проверки типов, следующие из теории типов. Таким образом, проблема статического контроля типов по-прежнему актуальна.

В диссертационной работе [80] исследуются проблемы интеграции языков программирования и языков запросов для распределенных баз данных. Автор использует SBA подход для разработки языка программирования запросов для баз данных iDBPQL (Data Base Programmin Query Language). В работе стандартный подход SBA расширяется посредством введения: долговечных объектов, транзакций, поддержки распределенных СУБД, поддержки работы с большими объемами данных, поддержки системы типов.

Подход SBA и его реализации (SBQL, iDBPQL) представляют собой многообещающий подход, предлагая новую парадигму запроса-ориентированных языков программирования, вместо применения обычных императивных языков программирования. В подходе решается большинство (SBQL) или все (iDBPQL) проблемы несоответствия импеданса. В отличие от подхода, предложенного в данной главе, SBA направлен на изобретение нового

языка программирования запросов, вместо того чтобы решать проблему несоответствия импеданса для стандартных ЯП и ЯЗ (например, Java и SQL).

3.7. Выводы по главе

В главе предложена система характеристик (features), которыми может характеризоваться и оцениваться способ сопряжения ЯП с системами управления информационными ресурсами. В терминах предложенной системы дана характеристика известных способов сопряжения ЯП с СУБД. Показано, как следует выбирать характеристики сопряжения для решения тех или иных проблем несоответствия импеданса. Осуществлен обоснованный выбор набора характеристик для развитого способа сопряжения процедурных ЯП с декларативным языком предметных посредников. Предлагается одновременно реализовать как статический подход, так и динамический подход. Статический подход позволяет решить проблемы несоответствия импеданса, а также обеспечивает приемлемую для задачи рассредоточения эффективность выполнения программ. Тем не менее, статический подход не позволяет задание программ к посреднику в динамике, поэтому в работе предлагается реализовать и динамический подход, обеспечивающий подобные возможности.

Анализ вариантов использования показал, что наиболее удобным оказывается совместное использование статического и динамического подходов. Схемы посредника и адаптеров редко изменяются и можно их считать фиксированными. В соответствии со статическим подходом, для них можно сгенерировать соответствующие Java классы. При этом программа на языке правил может часто изменяться, поэтому для получения результата удобно использовать динамический подход и конструкции SynthClass для представления результата. В контексте диссертационной работы, важными являются также вопросы производительности. Анализ показал, что при прочих равных условиях лучшие результаты получаются при использовании полностью статического подхода.

ГЛАВА 4. Конструирование адаптеров информационных ресурсов

4.1. Краткая характеристика адаптеров

Адаптеры реализуют унифицированный интерфейс доступа посредника к разнородным информационным ресурсам. Различные информационные модели ресурсов должны поддерживаться разными адаптерами.

Как было отмечено во второй главе, выполнение рассредоточенной программы может происходить на ресурсах управляемых адаптерами, на самих адаптерах, в интерпретаторе остаточных запросов (представляющего собой особый вид адаптера), а также в системах программирования. Поэтому в связи с задачей рассредоточения возникает задача конструирования адаптеров. Задача заключается в разработке архитектуры адаптера, поддерживающей адаптивное планирование и рассредоточение, и позволяющей, осуществлять эффективное выполнение операций на ресурсах, адаптерах и в посреднике. Адаптеры реализуют возможность выполнения запросов над ограниченной выборкой, что лежит в основе метода прогонки, и позволяет существенно снизить время оценки качества того или иного рассредоточения. Все операции, назначенные для реализации во взглядах или в программе посредника, выполняются в итоге на адаптерах либо в посреднике. На адаптерах выполнять операции выгоднее, т.к. это зачастую снижает сложность конечной обработки данных, а также может значительно уменьшить объем передаваемых данных. Именно поэтому для построения эффективного рассредоточения важно обеспечить возможность выполнения всех операций на адаптерах, даже если они не реализованы на информационном ресурсе. Особенно остро эта проблем стоит с функциональными предикатами, т.к. в отличие от других операций их алгоритм заранее не известен. Адаптеры должны обеспечивать возможность программирования функций для их выполнения на ресурсах или адаптерах.

Такие адаптеры называются программируемыми. Подобная возможность является важной с точки зрения производительности, т.к. позволяет приблизить реализацию функций обработки данных к самим данным.

Важным моментом также является разработка подхода к конструированию адаптеров, т.к. разработка нового адаптера для каждой новой информационной модели ресурса весьма накладна.

При более детальном рассмотрении архитектуры адаптеров (на примере адаптера молекулярно-генетических баз данных [81, 82], и адаптера XML-Schema [83 - 85]) становится видно, что все адаптеры имеют схожую структуру. Отличие наблюдается только в нескольких компонентах, зависящих от модели информационного ресурса [86 – 88].

Информационная модель определяется языком определения данных и языком манипулирования данными. Последний ограничивается в главе языком запросов.

4.2. Архитектура простого адаптера

Как было отмечено, адаптер реализуют унифицированный интерфейс доступа посредника к разнородным информационным ресурсам. Таким образом, адаптеры должны обеспечивать:

- преобразование запросов, поступающих от посредника (на языке запросов канонической модели посредника), в запросы к информационным ресурсам (на языке запросов конкретного ресурса);
- инициирование выполнения запроса на ресурсе и получение результата от ресурса;
- преобразование объектов результирующего множества, полученных от ресурса, в объекты канонической модели данных посредника;
- прием данных от других адаптеров.

На основании ранее разработанных в ИПИ РАН адаптеров (XML [83 - 85], реляционный, адаптер молекулярно-генетических баз данных[81, 82]), была разработана обобщенная архитектура адаптера, представленная на Рисунке 4.1.

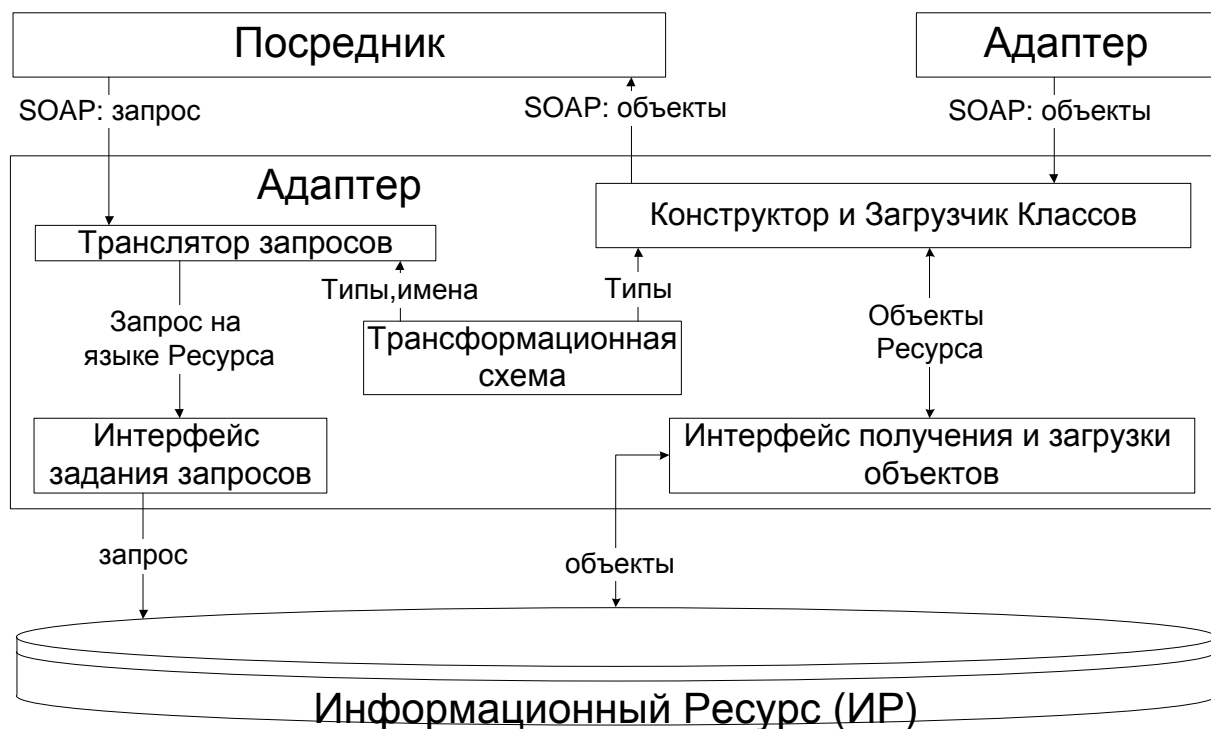


Рисунок 4.1. Архитектура Адаптера

Как видно на рисунке 4.1., основными компонентами адаптера являются:

- Транслятор;
- Трансформационная схема;
- Интерфейсы задания запрос, получения объектов, загрузки объектов;
- Конструктор и Загрузчик Классов.

Транслятор запросов получает запрос на каноническом языке запросов предметных посредников и осуществляет преобразование в запрос на языке задания запросов информационного ресурса. Для преобразования типов и их значений из канонической модели данных в типы и значения информационного ресурса используются методы *Трансформационной схемы*. Также *Трансформационная схема* используется для преобразования имен. После преобразования запрос возвращается на языке модели данных

информационного ресурса и выполняется посредством интерфейсных компонентов.

Трансформационная схема предназначена для преобразования имен и типов канонической модели посредника к именам и типам модели данных информационного ресурса. Компонент необходим, т.к. значения для разных моделей данных могут отличаться. Например, константы типа «*Дата*», могут быть представлены строкой, целым числом, или задаваться специальным форматом. Для преобразования типов используется транслятор схем, получаемый с помощью унификатора моделей при регистрации информационных ресурсов. Также *трансформационная схема* используется и для обратного преобразования из значений результата в каноническую форму для дальнейшей передачи результату посреднику.

Информация о преобразовании имен берется из таблицы соответствия. Данное описание выглядит как список пар вида:

<имя1> = <имя2>

Подобные соответствия имен устанавливаются в момент регистрации информационного ресурса, посредством инструмента регистрации.

Таким образом, видно, что *трансформационная схема* с одной стороны является компонентом, зависящим от модели данных информационного ресурса, на примере преобразования значений типов. С другой стороны компонент зависит от схемы конкретного ресурса, на примере преобразования имен.

Интерфейсные компоненты отвечают за взаимодействие с конкретным информационным ресурсом. Компоненты отвечают за передачу запроса, информационному ресурсу, за получение результата выполнения запроса, а также за загрузку данных в информационный ресурс. Данные, загруженные таким образом, могут быть использованы в запросах.

Конструктор классов преобразует данные, полученные от информационного ресурса, в формат обмена данными между компонентами

посредника. Данное преобразование происходит с использованием методов Трансформационной схемы, которые соответствующим образом преобразуют значения типов информационного ресурса, к значениям типов посредника. Загрузчик классов выполняет обратные действия. Данные полученные от других адаптеров, используя методы *трансформационной схемы*, преобразуются в значения модели данных информационного ресурса, и загружаются в информационный ресурс.

Адаптер обеспечивает взаимодействие компонентов архитектуры, определяя логику функционирования адаптера и его интерфейс. Простой адаптер позволяет задавать запрос к ресурсам, и получать от ресурсов результат. При получении запроса от посредника, адаптер транслирует запрос в язык запросов информационного ресурса (в случае реляционного адаптера в SQL92), который исполняется на информационном ресурсе. Результат, получаемый от информационного ресурса, преобразуется к классам посредника и возвращается посреднику. В общем случае алгоритм функционирования адаптера значительно сложнее. Важно отметить, что *адаптер*, и реализуемый им алгоритм функционирования адаптера, не зависят от конкретных информационных ресурсов.

4.3. Подход к конструированию адаптеров

Адаптеры к информационным ресурсам, представленные в разных моделях данных похожи. Поэтому возникает возможность, автоматизации процесса создания адаптеров [38].

В основе метода конструирования адаптеров, лежит идея выделения в архитектуре адаптера общей части для всех адаптеров (*шаблона адаптера*), и оформление оставшихся архитектурных компонентов (различающихся для разных адаптеров), как встраиваемых модулей. Все компоненты адаптера можно разделить на три группы:

- компоненты, зависимые от модели данных информационного ресурса (*Транслятор запросов, Трансформационная схема*);
- компоненты, общие для всех адаптеров (*Адаптер, Конструктор и Загрузчик классов*);
- компоненты, зависимые от конкретного информационного ресурса (*Интерфейсные компоненты*).

Компоненты, общие для всех адаптеров, можно создавать автоматически. Компоненты, зависимые от модели данных информационного ресурса, можно создавать полуавтоматически. Интерфейсные компоненты необходимо разрабатывать для каждого конкретного информационного ресурса. Рассмотрим компоненты зависимые от модели информационного ресурса.

Компонент *трансформационная схема* общий для всех адаптеров, но чтобы производить какие-то преобразования ему необходимо знать, что во что отображать. Для функционирования компонента используется транслятор схем, определяющий правила преобразования значений типов атрибутов. Также в трансформационной схеме используется таблица соответствия имен, определяющая правила преобразования имен атрибутов, которые зависят от конкретного информационного ресурса.

Существенной частью подхода, автоматизирующего создание зависимых от модели компонентов, является использование унификатора моделей [2], отображающего неоднородные модели информационных ресурсов в каноническую модель предметных посредников, и позволяющего создавать трансляторы как для схем, так и для запросов. В основе унификатора моделей лежит метод построения трансформаций на основе установленных соответствий между элементами моделей. В унификаторе используются трансформации моделей, сочетающие как декларативные, так и императивные средства описания отображения моделей.

Такие средства развиваются в контексте движимой моделями архитектуры (Model-Driven Architecture, MDA [72]) – подхода,

поддерживаемого стандартом OMG MOF (Meta-Object Facility) [89]. Базовыми составляющими MDA являются модели. Модель определяется в соответствии с семантикой некоторой метамодели, при этом говорят, что модель конформна (conforms to) метамодели. Стандартом MOF определена четырехуровневая архитектура моделей: модели уровня M0 описывают объекты реального мира, модели уровня M1 называются обычно схемами, модели уровня M2 представляют собой собственно информационные модели (например, в архитектурах систем управления данными представляют собой совокупность языка определения данных и языка манипулирования данными), модели уровня M3 – метамодели, предназначенные для описания моделей уровня M2. В качестве языка трансформации моделей в унификаторе рассматривается язык ATL (ATLAS Transformation Language) [73]. Система типов и операций над типами языка ATL очень близка (но не тождественна) системе типов языка OCL [90]. Для языка ATL на базе платформы Eclipse реализована интегрированная среда разработки, в качестве модели уровня M3 рассматривается модель Ecore [91].

Метод построения трансформаций с помощью унификатора моделей заключается в следующем:

- задание модели данных информационного ресурса в метамодели Ecore;
- определение соответствий между элементами модели информационного ресурса и элементами канонической модели данных предметных посредников;
- генерация ATL-модуля трансформации и последующее его редактирование экспертом;
- генерация транслятора.

Таким образом, конструирование нового адаптера сводится к следующим этапам: создание трансляторов с помощью унификатора моделей; реализация интерфейсных компонентов, определяющих пути и протоколы доступа к конкретному информационному ресурсу; автоматическое встраивание

интерфейсных компонентов и трансляторов как встраиваемых модулей в шаблон адаптера. Шаблон адаптера реализует основное поведение, определяемое архитектурой, и требованиями к адаптеру, рассматриваемыми ниже.

Важно отметить, что адаптер создается не для конкретного ресурса, а для некоторого класса ресурсов соответствующих одной модели данных. В рамках работы созданы следующие адаптеры: реляционный; объектно-реляционный; адаптер веб-сервисов; XML-адаптер на примере адаптера реестров Астрогрид [92]; адаптер информационных ресурсов системы Астрогрид (DSA) [93]; адаптер для информационного грида VizieR [94]; адаптер для специализированного ресурса SDSS [95]; адаптер потоковых данных [96].

4.4. Требования к адаптеру для поддержки эффективного выполнения рассредоточенной программы

Выше была описана архитектура простого адаптера, обеспечивающего задание запроса к ресурсу и получение результата. В распределенной среде зачастую этого не достаточно. Для выполнения рассредоточенной программы необходимо построить эффективный план выполнения.

Планировщику при построении эффективного плана необходимо опираться на оценочные и статистические данные, предоставляемые адаптером. Адаптер должен обеспечивать возможность выполнения операций, если они не поддерживаются ресурсом. Адаптер должен позволять задание операционных возможностей ресурса, чтобы запрещать выполнение тех операций, которые не поддерживаются, а также эффективно планировать нагрузку на вычислительные ресурсы. Кроме того важным моментом является стабильная работа адаптера в распределенной среде при многопоточном доступе. Соответствующие требования к адаптерам перечислены ниже:

- Одновременное использование всех методов классов в нескольких потоках (thread safe). При реализации адаптеров особое внимание должно

уделяться проблеме возникновения ситуаций, вызывающих взаимоблокировки при синхронизации потоков.

- Поддержка возможности выполнения запроса на информационном ресурсе и получения результата этого запроса. Это основная функция адаптера.
- Поддержка сессий. Поскольку адаптеры функционируют в распределенной среде, наличие сессий, изолированных друг от друга, с изолированными ресурсами, является необходимым.
- Поддержка возможности повторного использования результата запроса. В случае, когда одни и те же данные требуются в нескольких частях плана, запрос, продуцирующий эти данные, придется выполнять несколько раз. Подобный подход не приемлем для эффективного планирования, и следовательно необходима возможность повторного использования результатов.
- Поддержка возможности загрузки данных в адаптер планировщиком, а также возможности материализации результата запроса в адаптерах.
- Учет операционных возможностей адаптеров. Ресурсы бывают различные, с различными возможностями. Например, с возможностью или невозможностью хранить временные данные. Для эффективного планирования необходимо учитывать все такие особенности.
- Поддержка оценочных запросов. Для эффективного планирования необходимы оценки объема результата, количества объектов и других оценочных данных.
- Поддержка возможности задания запроса над ограниченной выборкой. Зачастую требуется выполнить запрос не над всеми данными, а над ограниченным объемом. Например, для оценок выполнения операции соединения, необходим пример реального результата. Кроме того, подобная возможность лежит в основе метода прогонки, используемого для построения эффективного рассредоточения.

- Поддержка статистики. Для эффективного планирования требуется статистика, чтобы можно было оценивать такие переменные параметры как скорость передачи данных по сети и вычислительную мощность информационных ресурсов.

Все выше обозначенные требования были учтены при реализации адаптеров. Важно отметить, что усложнение адаптера не влияет на общность рассуждений по конструированию адаптеров, представленных в разделе 4.3. Таким образом, сложность разработки новых адаптеров не увеличивается.

4.5. Программируемый адаптер

Для обеспечения возможности построения эффективного рассредоточения адаптер должен обладать возможностью выполнения методов и функций на ресурсах. Программируемый адаптер представляет собой расширение стандартной архитектуры адаптеров дополнительными возможностями. Важно отметить, что данное расширение не связано с конкретными адаптерами и может быть использовано в любом из существующих адаптеров.

Традиционно адаптер предоставляет лишь те возможности, которыми обладает информационный ресурс. Таким образом, адаптер предоставляет доступ к данным в ресурсе, а также возможность вызова хранимых процедур, если таковые имеются. На практике хранимые процедуры и функции в информационных ресурсах встречаются крайне редко, а все методы обработки чаще всего доступны в виде веб-сервисов, или реализациях на некотором ЯП.

Идея программируемого адаптера заключается в том, чтобы и функции и данные были описаны в одном адаптере (даже если функция не реализуется информационным ресурсом). Таким образом, если нужно выполнить некоторую функцию над коллекцией, то данные извлекаются из ресурса, после чего происходит выполнение функции, и далее данные уже передаются куда-либо. Таким образом, можно программировать новые методы в адаптере,

поэтому такой адаптер и называется программируемым. Рассмотрим теперь варианты реализации функций в программируемом адаптере.

Реализация функции веб-сервисом. Чаще всего функция представлена готовым веб-сервисом на некотором внешнем ресурсе, доступным по стандартному SOAP протоколу. Таким образом, для выполнения функции над коллекцией из n объектов, необходимо n раз вызвать веб-сервис (если веб-сервис не умеет принимать коллекции). В программируемом адаптере подобное выполнение можно распараллелить. К сервису можно одновременно открывать много соединений, что за счет параллельного выполнения повышает производительность. Оптимальное количество одновременных соединений с веб-сервисом, определяется автоматически.

Реализация функции на языке программирования. Нередко бывает так, что алгоритм функции (реализованной веб-сервисом) известен, либо может быть легко найден, и не сложен с программистской точки зрения. В случае же множественной семантики, когда необходимо выполнять функцию над коллекцией объектов, использование веб-сервисов неэффективно. Реализация функции на языке программирования – альтернатива веб-сервисам, существенно повышающая производительность. Прирост производительности получается за счет исчезновения почти всех накладных расходов, т.к. вместо n вызовов по сети веб-сервиса (с передачей данных туда и обратно) функция вызывается для данных в памяти адаптера. Кроме того подобные функции могут быть описаны не только в адаптерах, но и в интерпретаторе остаточных запросов, т.к. этот компонент представляет собой расширенную версию адаптера.

Реализация функции на языке программирования объектно-реляционной СУБД. Выполнение функций реализованных на языке программирования, происходит в памяти адаптера. Хотя подобная реализация и дает существенный выигрыш в производительности по сравнению с использованием веб-сервисов, она может быть недостаточно эффективной. В

случае если обрабатываются большие объемы данных, выполнение функции может быть накладным в силу ограниченности памяти. В этом случае эффективной реализацией может служить реализация на языке программирования СУБД, выполняемая на СУБД информационного ресурса. Подобная реализация может существенно повышать производительность.

4.6. Основные особенности реализации адаптеров для задачи рассредоточения

Техническая сторона реализации адаптеров описывается в приложении Д, здесь же рассматриваются основные моменты реализации адаптеров, связанные с задачей рассредоточения.

Все адаптеры являются программируемыми. Адаптер функционирует как отдельный веб-сервис с доступом по SOAP или посредством Java АПИ. Те данные и функции, что доступны в адаптере, описываются его схемой. Чтобы добавить функцию в адаптер, достаточно загрузить в адаптер реализацию функции в виде кода на Java и описать функцию в схеме Адаптера. Таким образом, адаптер можно программировать, не перестраивая его.

Адаптеры реализуют метод прогонки. Метод прогонки, используемый при построении эффективного рассредоточения, заключается в выполнении задачи над ограниченной выборкой данных. Подобное возможно благодаря адаптерам, которые позволяют ограничить количество объектов в результирующей выборке, объем в байтах или объем в процентном отношении. Объекты выбираются псевдослучайным образом.

Адаптеры могут быть оформлены *как веб-сервисы* и функционировать «рядом» с информационным ресурсом, а могут быть оформлены *как модули исполнительного слоя предметных посредников* и функционировать «рядом» с посредником. Функционирование адаптеров «рядом» с ресурсом позволяет избежать издержек связанных с передачей данных от ресурса в адаптер, а также позволяет эффективно выполнять функции над данными ресурса. Подобная

возможность доступна далеко не всегда. Зачастую доступ к информационному ресурсу доступен только по сети, и нет возможности запустить адаптер на сервере информационного ресурса. В этом случае, может быть выгодно функционирование адаптера как модуля исполнительного слоя предметных посредников, что позволяет избежать издержек связанных с передачей данных от адаптера в посредник по сети.

Адаптеры могут выполнять все операции, вместо ресурса. Адаптеры могут использоваться даже с ресурсами, поддерживающими только запросы вида «select * from», т.е. такими ресурсами, что не умеют выполнять ни операции усечения множества по условию, ни проекции, ни соединения, ни объединения, ни функции. Операции, не поддерживаемые ресурсом, может выполнять адаптер. Но далеко не всегда выполнение операций в адаптере эффективно. Например, выполнение операции соединения в памяти – весьма трудоемкая операция. Для этого в адаптере можно задать операционные возможности ресурса, указав, что та или иная операция не поддерживается. В этом случае подобные операции не будут поступать для выполнения на адаптер. Подробная спецификация операционных возможностей адаптеров представлена в приложении Д.

Адаптеры позволяют загружать данные в ресурс (если это поддерживается ресурсом). Зачастую требуется выполнять операции соединения над данными из разных ресурсов. В этом случае, данные можно передать в посредник, и там выполнить соединение. Это не всегда эффективно и в отдельных случаях просто нереализуемо. Например, астрономический ресурс SDSS содержит терабайты данных. Передача такого объема данных по сети достаточно трудоемкое занятие. Если же в подобный ресурс (SDSS) загрузить данные из другого ресурса, выполнить операцию соединения, и далее из результата выбрать данные по некоторому условию, то объем данных может существенно уменьшиться. Если ресурс позволяют загружать данные, то

адаптеры предоставляет возможности материализации результата выполнения запросов, с тем чтобы их (результаты) можно было повторно использовать.

Адаптеры позволяют *выполнять оценочные запросы*, а также *собирать статистику*. Данная функциональность необходима планировщику для построения эффективного плана. Зачастую недостаточно возможности выполнения запроса над ограниченной выборкой данных. Оценочные запросы позволяют оценить объем результата, кол-во объектов в результате, средний объем объекта. Статистика собирается как по скорости выполнения запросов, так и по времени передачи данных по сети. Средняя скорость выполнения используется при принятии решения, в каком ресурсе выполнять трудоемкие операции, например операцию соединения. Статистика по передаче данных используется для принятия решения выполнять ли вообще операцию в адаптере. Например, если скорость передачи данных от адаптера низкая, то выгоднее передать две коллекции объектов следующему ресурсу, а там уже выполнить операцию соединения.

Адаптеры, разработанные в работе, реализуют все выше перечисленные возможности. Естественно, базовая функциональность (выполнение запросов, получение результата) также реализована. Важным моментом является то, что все вышеперечисленные возможности являются общими для всех адаптеров. Создание нового адаптера требует лишь создания трансляторов и реализации интерфейсных компонентов, что существенно быстрее, нежели конструировать новый адаптер с нуля.

4.7. Описание реализации реляционного адаптера

В соответствии с предложенной архитектурой простого адаптера, а также требованиями к адаптерам, связанными с задачей рассредоточения, был разработан ряд адаптеров, перечисленных ранее. Рассмотрим разработанные адаптеры на примере реляционного адаптера. Отличие между двумя различными адаптерами, поддерживающими разные информационные модели,

заключается только в двух компонентах, реализующих следующие интерфейсы: *QueryTranslator* и *SourceConnector*.

```
public interface QueryTranslator
{
    public NativeQuery translate(Plan plan);
}
public interface SourceConnector
{
    public void openSession() throws ConnectionException;
    public void closeSession() throws ConnectionException;

    public long estimateRows();
    public long estimateDataSize();
    public long averageRowSize();

    public void getResultData(long rowsToSkip, long numberOfRows, SynthClassWriter writer)
throws IllegalArgumentException, ConnectionException;

    public long loadData(String tableName, SynthClassReader synthClassReader, Module
sessionModule, String className);
    public void removeData(String tableName);

    public int registerAdapterSession();
    public void keepAlive(int sessionID);
    public void unRegisterSession(int sessionID);
}
```

В интерфейсе *QueryTranslator* определен один метод, который принимает объектное представление запроса на языке *Asyfs* (приложение Б), и возвращает объект *NativeQuery*, который содержит в себе текстовое или иное представление запроса, а также некоторые метаданные, необходимые для выполнения запроса. За генерацию компонента *QueryTranslator* для конкретного адаптера отвечает унификатор моделей.

В интерфейсе *SourceConnector* определены методы для работы с сессиями (*openSession*, *closeSession*, *registerAdapterSession*, *keepAlive*, *unRegisterSession*), методы для оценки запросов (*estimateRows*, *estimateDataSize*, *averageRowSize*), метод получения данных результата запроса (*getResultData*), методы загрузки данных во временную таблицу (*loadData*) и удаления временных таблиц (*removeData*). Данный интерфейс необходимо разрабатывать вручную. Правда, это совсем не трудоемкое занятие. К примеру, реализация интерфейса для разработанных адаптеров содержит всего 300-500 строк кода. Максимум содержал реляционный адаптер, т.к. он обладает более широкими

возможностями по сравнению с другими адаптерами, не умеющими принимать коллекции и загружать их в информационный ресурс.

Сам адаптер реализует интерфейс RemoteAdapter (подробно описанный в приложении Д). Посредник, получив запрос от пользователя в терминах схемы посредника, переписывает данный запрос в термины локальных схем информационных ресурсов. Затем этот запрос планируется и разбивается на подзапросы планировщиком. Затем посредник выполняет подзапросы на адаптерах и получает данные. Каждый подзапрос отправляется отдельному адаптеру. Компонент посредника *супервизор* отправляет запрос адаптеру посредством вызова метода, и получает в качестве результата идентификатор, по которому, можно получить результат. Адаптер представляет собой пассивный компонент, все методы которого вызываются либо планировщиком, либо супервизором. На рисунке 4.2. представлена диаграмма классов адаптера.

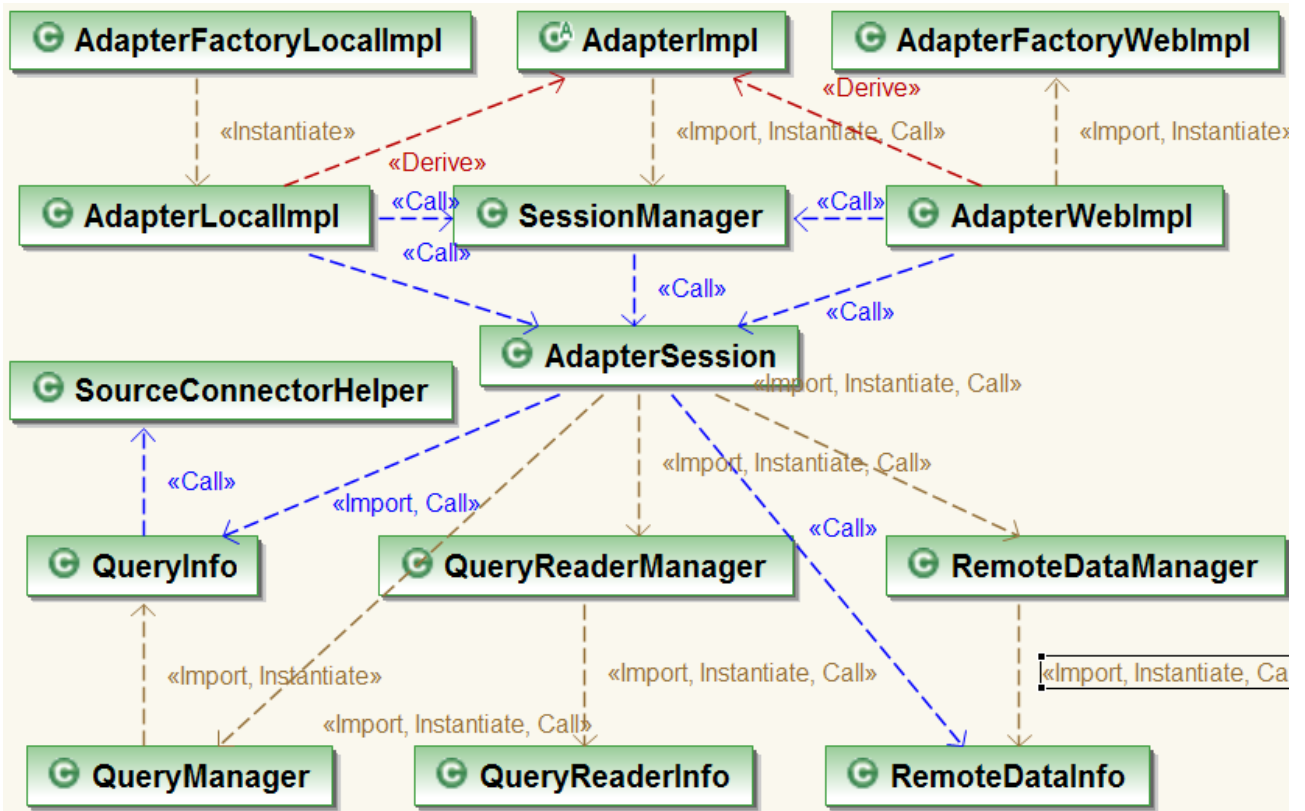


Рисунок 4.2. Диаграмма классов Адаптера информационных ресурсов

Помимо транслятора и интерфейсных компонентов (рисунок 4.2.), адаптер включает:

- менеджер запросов (*QueryManager*), организующий поступающие запросы и определяющий очередность их выполнения;
- менеджер загрузки и передачи данных (*RemoteDataManager*), отвечающий как за передачу данных от ресурса к адаптеру, так и от адаптера к посреднику;
- фабрику адаптеров (*AdapterFactoryLocalImpl*, *AdapterFactoryWebImpl*), позволяющую создавать новые сущности адаптеров (иницируемые только схемой ресурса) в виде веб-сервисов либо модулей посредника;
- интерфейсы адаптера в виде модуля посредника (*AdapterLocalImpl*) или веб-сервиса (*AdapterWebImpl*), обеспечивающие безопасное выполнение (thread safe) всех методов при многопоточном доступе (т.к. сущностей одного адаптера может быть много);
- менеджер сессий (*SessionManager*), отвечающий за распределение вычислительных ресурсов (объем доступной памяти, количество доступных соединений с ресурсом, и.т.д.) между сессиями и запросами;
- менеджер больших объемов данных (*QueryReaderManager*), задачей которого является разбиение получаемых или отправляемых данных на порции в случае, если объем данных превышает выделенные для сессии ресурсы;
- менеджер сбора статистики, отвечающий за аккумуляцию статистики выполненных запросов (*QueryReaderInfo*) и переданных данных (*RemoteDataInfo*).

4.8. Обзор существующих подходов

Существует ряд работ посвященных проблеме создания адаптеров. В последнее время, проблема конструирования адаптеров зачастую используется в контексте доступа к данным из сети интернет. Предлагаются различные подходы автоматического [97] или полуавтоматического [98] конструирования адаптеров для веб данных. В подобных подходах задача адаптера заключается обычно в структуризации представления данных из веб-страниц. Зачастую этого не достаточно, поэтому в некоторых работах [99] обсуждаются также и вопросы задания запросов к подобным данным. Проблема создания адаптеров достаточно актуальна, и получила немалое распространение в промышленности. Об этом говорит внушительный список различных инструментов для конструирования адаптеров [100]. Главный недостаток всех этих инструментов в том, что они не касаются вопроса создания трансляторов. В проекте TSIMMIS [25] вопросам конструирования компонентов уделено особое внимание. В этом проекте генерируются не только посредники, но и адаптеры посредством языка спецификации адаптеров. Интерфейс доступа к информационным ресурсам, и программа выполнения запросов пишется программистом вручную. В работе [101] отмечается, что с точки зрения адаптера, наибольшую важность представляет транслятор запросов, и именно автоматизацией создания трансляторов и стоит заниматься. В работе используется подход построения трансформации для образцов, что существенно сужает мощность языка запросов. В подходе, обсуждаемом в данной главе, необходимо поддерживать максимальные возможности языка запросов, именно поэтому используются трансляторы, получаемы с помощью унификатора моделей. Наконец во всех рассмотренных проектах адаптеры выполняют достаточно простые действия (задание запроса, получения результата). Сложные адаптеры, описываемые в разделе 4.4 и 4.5, не рассматриваются.

4.9. Выводы по главе

В главе разработана обобщенная архитектура программируемого адаптера. Для эффективного выполнения рассредоточенной программы, а также для поддержки адаптивного планирования определены требования, которым должен соответствовать адаптер. В соответствии с данными требованиями был реализован ряд адаптеров: реляционный; объектно-реляционный; адаптер веб-сервисов; XML-адаптер на примере адаптера реестров Астрогрид; адаптер информационных ресурсов системы Астрогрид (DSA); адаптер для информационного грида VizieR; адаптер для специализированного ресурса SDSS; адаптер потоковых данных. Важным с точки зрения задачи рассредоточения является также предложенный подход использования программируемых адаптеров для исполнения функций и методов. В главе предложен подход конструирования адаптеров для сред предметных посредников. Важным моментом является то, что адаптер создается не для каждого отдельного ресурса, а для класса ресурсов, оперирующих одной информационной моделью.

ГЛАВА 5. Практическое применение и тестирование системы построения рассредоточений

5.1. Описание программной реализации системы построения рассредоточений

Методы и алгоритмы, описываемые во 2ой главе, были реализованы в виде программного инструментария системы построения рассредоточений. Архитектура компонентов системы построения рассредоточений представлена ниже (рисунок 5.1.).



Рисунок 5.1. - Компоненты системы построения рассредоточений

В общем случае, система построения рассредоточения включает в себя:

- загрузчик спецификаций языка СИНТЕЗ;
- трансляторы спецификаций алгоритма решения задачи в модель рассредоточения, и обратные трансляторы;
- систему проведения симуляций (методом прогонки);

- редактор рассредоточения;
- редактор модели рассредоточения (для модели рассредоточения используется текстовое представление графов основанное на языке XML);
- редактор программы на ЯП;
- редактор GLAV взглядов;
- редактор возможностей адаптеров;
- редактор реализации функций;
- редактор программы на языке правил посредника;
- компонент, осуществляющий автоматическое построение эффективного рассредоточения одним из трех алгоритмов (на основе экспертных правил, на основе направленного перебора, на основе полного перебора).

Система построения рассредоточений разрабатывалась как модуль для платформы Eclipse. Подобное решение позволяет использовать систему в качестве самостоятельного приложения, так и легко встраивать ее в другие приложения в рамках платформы Eclipse. Также другие модули платформы Eclipse могут быть использованы в системе. Редакторы Java программ, программ на языке правил, а также семантических правил (взглядов) были взяты готовыми и встроены в систему.

В начальный момент времени системой построения рассредоточений по заданным спецификациям алгоритма решения задачи строится модель рассредоточения, а также производится симуляция выполнения начального рассредоточения. После чего специалист может осуществлять следующие действия:

- редактирование исходных спецификаций алгоритма решения задачи;
- редактирование модели рассредоточения вручную;
- редактирование операционных возможностей адаптеров;

- задание реализации для функций, причем для одной функции может быть задано несколько реализаций (например, на ЯП и на PL/SQL), для эффективного рассредоточения будет выбрана оптимальная реализация;
- построение эффективного рассредоточения применением экспертных правил;
- построение эффективного рассредоточения алгоритмом направленного перебора;
- построение минимального рассредоточения алгоритмом полного перебора;
- осуществление симуляции методом прогонки для определения производительности текущего рассредоточения.

Важно отметить, что специалист может зафиксировать назначение для любой из операций модели рассредоточения, в этом случае любой из методов построения эффективного рассредоточения не меняет назначение зафиксированной операции. Это удобно в случае, когда необходимо найти минимальное рассредоточение лишь для некоторых операций, для которых не удалось выбрать оптимальное назначение с помощью экспертных правил и знаний эксперта. В следующем разделе рассматривается применение разработанной системы построения рассредоточений для реальной научной задачи.

5.2. Пример применения системы построения эффективного рассредоточения для научной задачи

Рассмотрим пример задачи, описанной в разделе 1.5. Текстовые спецификации выглядят следующим образом.

GAV взгляды:

```
// GAV for SDSS
v_SDSS_Data(x/[ra, de, name, magnitudes, quality, objectType, properMotion])
:- CATALOG_SDSS.catalogSDSS(x/[ra:RAJ2000, de:DEJ2000, name:SDSS, umag, gmag, rmag, imag, zmag,
Q, c1])
& ProgramableA.ugriz2JohnsonB(umag, gmag, rmag, JohnsonB)
```

```

& ProgramableA.ugriz2JohnsonV(umag, gmag, rmag, JohnsonV)
& ProgramableA.ugriz2JohnsonR(gmag, rmag, imag, JohnsonR)
& ProgramableA.ugriz2JohnsonI(rmag, imag, zmag, JohnsonI)
& ProgramableA.formMag(umag, 0, 'UGRIZ_U', U)
& ProgramableA.formMag(gmag, 0, 'UGRIZ_G', G)
& ProgramableA.formMag(rmag, 0, 'UGRIZ_R', R)
& ProgramableA.formMag(imag, 0, 'UGRIZ_I', I)
& ProgramableA.formMag(zmag, 0, 'UGRIZ_Z', Z)
& ProgramableA.form0Mags(mags0)
& ProgramableA.addMag2Mags(U, mags0, mags1)
& ProgramableA.addMag2Mags(G, mags1, mags2)
& ProgramableA.addMag2Mags(R, mags2, mags3)
& ProgramableA.addMag2Mags(I, mags3, mags4)
& ProgramableA.addMag2Mags(Z, mags4, magnitudes)
& ProgramableA.sdssGetQuality(Q, quality)
& ProgramableA.sdssGetObjecttype(cl, objectType)
& ProgramableA.sdssCheckColorIndex(JohnsonB, JohnsonV, JohnsonR, JohnsonI, acceptable)
& acceptable = true
& id(0, properMotion)
& rmag > 15 & rmag < 20
// GAV for 2MASS
v_2MASS_Data(x/[ra, de, name, magnitudes, quality, objectType, properMotion])
:- CATALOG_2MASS.twomass_psc(x/[ra:RAJ2000, de:DEJ2000, name:@'2MASS', Kmag, Jmag, Hmag, Qflg,
Rflg, Bflg, Cflg, Xflg, Aflg])
& ProgramableA.formMag(Kmag, 0, 'K', K)
& ProgramableA.formMag(Jmag, 0, 'J', J)
& ProgramableA.formMag(Hmag, 0, 'H', H)
& ProgramableA.form0Mags(mags0)
& ProgramableA.addMag2Mags(K, mags0, mags1)
& ProgramableA.addMag2Mags(J, mags1, mags2)
& ProgramableA.addMag2Mags(H, mags2, magnitudes)
& ProgramableA.c2MassGetQuality(Qflg, Rflg, quality)
& ProgramableA.getUnknownObjectType(objectType)
& Bflg = '111'
& Cflg = '000'
& Xflg = 0
& Aflg = 0
& id(0, properMotion)
& Jmag > 12 & Jmag < 18
// GAV for USNOB1
v_USNOB1_Data(x/[ra, de, name, magnitudes, quality, objectType, properMotion])
:- CATALOG_USNOB1.catalogUSNOB1(x/[ra:RAJ2000, de:DEJ2000, name:@'USNO-B1.0', pmRA, pmDE, B1mag,
R1mag, B2mag, R2mag, Imag, B1sg: '@'B1s/g', R1sg: '@'R1s/g', B2sg: '@'B2s/g', R2sg: '@'R2s/g', Isg:
@'Is/g'])
& ProgramableA.formMag(B1mag, 0, 'B1', B1)
& ProgramableA.formMag(R1mag, 0, 'R1', R1)
& ProgramableA.formMag(B2mag, 0, 'B2', B2)
& ProgramableA.formMag(R2mag, 0, 'R2', R2)
& ProgramableA.formMag(Imag, 0, 'I', I)
& ProgramableA.form0Mags(mags0)
& ProgramableA.addMag2Mags(B1, mags0, mags1)
& ProgramableA.addMag2Mags(R1, mags1, mags2)
& ProgramableA.addMag2Mags(B2, mags2, mags3)
& ProgramableA.addMag2Mags(R2, mags3, mags4)
& ProgramableA.addMag2Mags(I, mags4, magnitudes)
& ProgramableA.usnob1GetObjecttype(B1sg, R1sg, B2sg, R2sg, Isg, objectType)
& ProgramableA.usnob1CheckColorIndex(B1mag, R1mag, B2mag, R2mag, acceptable)
& ProgramableA.usnob1getProperMotion(pmRA, pmDE, properMotion)
& acceptable = true
& id(0, quality)
& R1mag > 12 & R1mag < 18
& R2mag > 12 & R2mag < 18
// GAV for VSX
v_VSX_Data(x/[ra, de, name])
:- CATALOG_VSX.catalogVSX(x/[ra:RAJ2000, de:DEJ2000, name:Name])
// GAV for ASAS

```

```

v_ASAS_Data(x/[ra, de, name])
:- CATALOG_ASAS.catalogASAS(x/[ra:_RA, de:_DE, name:ASAS])
// GAV for GSC
v_GSC_Data(x/[ra, de, name, magnitudes, objectType])
:- CATALOG_GSC.catalogGSC(x/[ra:RAJ2000, de:DEJ2000, name:@'GSC2.3', Vmag])
& ProgramableA.formMag(Vmag, 0, 'V', V)
& ProgramableA.form0Mags(mags0)
& ProgramableA.addMag2Mags(V, mags0, magnitudes)
& ProgramableA.getUnknownObjectType(objectType)
// GAV for UCAC
v_UCAC_Data(x/[ra, de, name, magnitudes, objectType])
:- CATALOG_UCAC.catalogUCAC(x/[ra:RAJ2000, de:DEJ2000, name:@'3UC', fmag: @'f.mag'])
& ProgramableA.formMag(fmag, 0, 'f', f)
& ProgramableA.form0Mags(mags0)
& ProgramableA.addMag2Mags(f, mags0, magnitudes)
& ProgramableA.getUnknownObjectType(objectType)

```

LA V ВЗГЛЯДЫ:

```

Views.v_SDSS_Data(x/[ra, de, name, magnitudes, quality, objectType, properMotion])
:- astronomicalObject(x/[ra: spatialCoord.ra, de: spatialCoord.de, name, magnitudes, quality,
objectType, properMotion])

Views.v_2MASS_Data(x/[ra, de, name, magnitudes, quality, objectType, properMotion])
:- astronomicalObject(x/[ra: spatialCoord.ra, de: spatialCoord.de, name, magnitudes, quality,
objectType, properMotion])

Views.v_USNOB1_Data(x/[ra, de, name, magnitudes, quality, objectType, properMotion])
:- astronomicalObject(x/[ra: spatialCoord.ra, de: spatialCoord.de, name, magnitudes, quality,
objectType, properMotion])

Views.v_VSX_Data(x/[ra, de, name])
:- variableStar(x/[ra: spatialCoord.ra, de: spatialCoord.de, name])

Views.v_ASAS_Data(x/[ra, de, name])
:- variableStar(x/[ra: spatialCoord.ra, de: spatialCoord.de, name])

Views.v_GSC_Data(x/[ra, de, name, magnitudes, objectType])
:- astronomicalObject(x/[ra: spatialCoord.ra, de: spatialCoord.de, name, magnitudes,
objectType])

Views.v_UCAC_Data(x/[ra, de, name, magnitudes, objectType])
:- astronomicalObject(x/[ra: spatialCoord.ra, de: spatialCoord.de, name, magnitudes,
objectType])

```

Программа на языке правил состоит из девяти правил:

```

1 r(x/[ra, de, magnitudes, objectType, properMotion, quality])
:- astronomicalObject(x1/[ra: spatialCoord.ra, de: spatialCoord.de, objectType,
properMotion, quality, magnitudes])
& ra < queryRA + radius & ra > queryRA - radius
& de < queryDE + radius & de > queryDE - radius;

2 combineMagnitudes (r/AstronomicalObject, r1);

3 getIsolated(r1, r2);

4 r3(x/[ra, de, name, magnitudes])
:- r2(x1/[ra, de, name, objectType, properMotion, quality, magnitudes])
& checkType(ra, de, 'G', nType) & nType = false
& objectType = Star
& properMotion < 0.01
& quality < 0.01;

5 r4(x/[ra, de, name])
:- r1(x1/[ra, de, name, magnitudes])
& isVariablebyMagnitudes(ra, de, isVar) & isVar = true;

```

```

6   r4(x/[ra, de, name])
   :- variableStar(x1/[ra: spatialCoord.ra, de: spatialCoord.de, name]);
   & ra < queryRA + radius & ra > queryRA - radius
   & de < queryDE + radius & de > queryDE - radius;

7   xmatch(r3, r4, r5);

8   r6(x/[ra, de, name, magnitudes])
   :- r5(x1/[ra, de, name, magnitudes, distance])
   & distance > 0.01;

9   r7(im/Image)
   :- r6(x/ra, de, name, magnitudes)
   & showStandards(ra, de, radius, magnitudes, im);
    
```

На рисунке 5.2. представлена модель рассредоточения, соответствующая выше описанным спецификациям. В данном примере, алгоритм решения задачи описан только в виде GLAV взглядов (серый цвет) и программы на языке правил посредника (синий цвет).

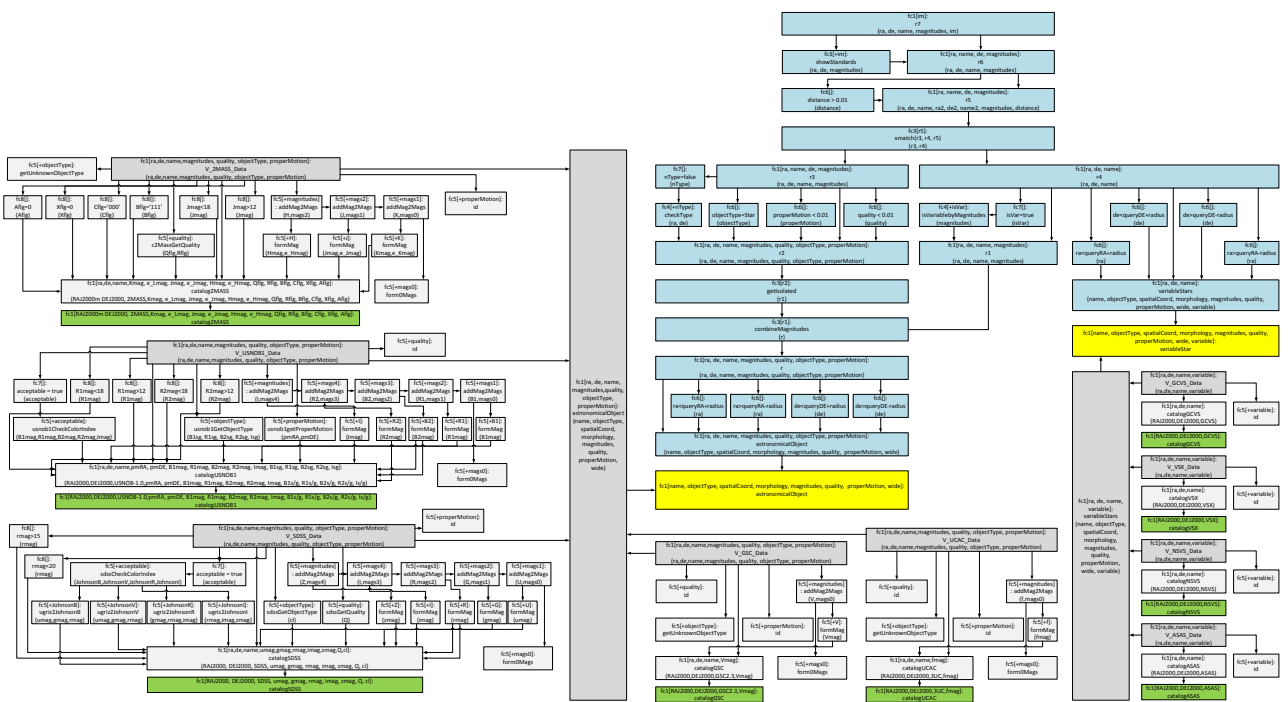


Рисунок 5.2. Модель рассредоточения для задачи поиска стандартов
 Разберем подробнее, что в данной реализации алгоритма решения задачи плохо с точки зрения производительности:

- Все функции, реализуемые на отдельном ресурсе, влечут за собой накладные расходы, связанные с необходимостью передавать данные в удаленный ресурс и обратно.

- Во взглядах используется вызов множества функций, которые не реализованы на самом ресурсе, что влечет за собой необходимость передачи всех данных из ресурса и отсутствие возможности выполнения условий, ограничивающих выборку.
- В правилах №2,3,7 используются функции, оперирующие всей коллекцией в целом, что влечет за собой необходимость собрать данные из всех параллельных ветвей выполнения перед тем как начать выполнять данные правила.
- Условия, ограничивающие выборку в правиле №4, не могут быть выполнены на ресурсах.
- В правиле №3 функция checkType представляет собой обращение к удаленному сервису, проверяющему тип для конкретного объекта. Функция вызывается n раз (по разу на каждый объект), что влечет за собой накладные расходы.
- В правиле №5 переменность определяется функцией, реализованной на удаленном ресурсе.
- Правило №6 и первые пять правил могут выполняться параллельно. Правило №3,4 и правило №5 также могут выполняться одновременно. Это невозможно в случае реализации алгоритма решения задачи только как программы на правилах, т.к. правила выполняются последовательно.
- В правиле №8 отсеиваются элементы, не удовлетворяющие условию, это условие можно проверять и в правиле №7.
- В Правиле №9 большие временные затраты связаны с передачей изображения по сети, чего можно избежать если заранее получить изображение одновременно с правилами 1-8. Кроме того функция использует стороннее средство Aladin, поэтому прирост производительности можно получить, если реализовать функцию сразу в ЯП, а не вызывать в правилах реализованную на стороннем ресурсе функцию.

Рассмотрим поэтапно процесс построения эффективного рассредоточения с помощью разработанного инструментария, включающий в себя: построение модели рассредоточения (автоматически), перестановку операций (автоматически или с участием эксперта), генерацию итоговой программы (автоматически).

Как было описано в разделе 2.5., процесс построения модели рассредоточения начинается с анализа взглядов. Рассмотрим взгляд для ресурса USNOB1, и как по данному взгляду операции добавляются в модель рассредоточения. Ниже представлены шаблоны операций, выделенные во взгляде, отмечено то, каким классам функциональных операций они соответствуют, а также выделены зависимости.

```
// GAV for USNOB1
• 00# CATALOG_USNOB1.catalogUSNOB1 – класс ресурса, добавляется в модель рассредоточения
автоматически, ни от какой другой операции не зависит
• 01# CATALOG_USNOB1.catalogUSNOB1(x/[ra:RAJ2000, de:DEJ2000, name:@'USNO-B1.0', pmRA, pmDE,
B1mag, R1mag, B2mag, R2mag, Imag, B1sg: '@'B1s/g', R1sg: '@'R1s/g', B2sg: '@'B2s/g', R2sg:
@'R2s/g', Isg: '@'Is/g']) – операция класса fc1, зависит только от операции #00
• 02# ProgramableA.formMag(B1mag, 0, 'B1', B1) – операция класса fc6, зависит от атрибута B1mag, а
следовательно, от операции #01
• 03# ProgramableA.formMag(R1mag, 0, 'R1', R1) – операция класса fc6, зависит от атрибута R1mag, а
следовательно, от операции #01
• 04# ProgramableA.formMag(B2mag, 0, 'B2', B2) – операция класса fc6, зависит от атрибута B2mag, а
следовательно, от операции #01
• 05# ProgramableA.formMag(R2mag, 0, 'R2', R2) – операция класса fc6, зависит от атрибута R2mag, а
следовательно, от операции #01
• 06# ProgramableA.formMag(Imag, 0, 'I', I) – операция класса fc6, зависит от атрибута Imag, а
следовательно, от операции #01
• 07# ProgramableA.form0Mags(mags0) – операция класса fc6, не зависит от других операций
• 08# ProgramableA.addMag2Mags(B1, mags0, mags1) – операция класса fc6, зависит от атрибута B1 и
mags0, а следовательно, от операции #02 и #07
• 09# ProgramableA.addMag2Mags(R1, mags1, mags2) – операция класса fc6, зависит от атрибута R1 и
mags1, а следовательно, от операции #03 и #08
• 10# ProgramableA.addMag2Mags(B2, mags2, mags3) – операция класса fc6, зависит от атрибута B2 и
mags2, а следовательно, от операции #04 и #09
• 11# ProgramableA.addMag2Mags(R2, mags3, mags4) – операция класса fc6, зависит от атрибута R2 и
mags3, а следовательно, от операции #05 и #10
• 12# ProgramableA.addMag2Mags(I, mags4, magnitudes) – операция класса fc6, зависит от атрибута I
и mags4, а следовательно, от операции #06 и #11
• 13# ProgramableA.usnob1GetObjectype(B1sg, R1sg, B2sg, R2sg, Isg, objectType) – операция класса
fc6, зависит от атрибута B1sg, R1sg, B2sg, R2sg, I, а следовательно, от операции #01
• 14# ProgramableA.usnob1CheckColorIndex(B1mag, R1mag, B2mag, R2mag, acceptable) – операция
класса fc6, зависит от атрибута B1mag, R1mag, B2mag, R2mag, а следовательно, от операции #01
• 15# ProgramableA.usnob1getProperMotion(pmRA, pmDE, properMotion) – операция класса fc6, зависит
от атрибута pmRA, pmDE, а следовательно, от операции #01
• 16# acceptable = true – операция класс fc7, зависит от атрибута acceptable, а следовательно, от
операции #14
• 17# id(0, quality) – операция класса fc6, не зависит от других операций
• 18# R1mag > 12 – операция класс fc7, зависит от операции #01
• 19# R1mag < 18 – операция класс fc7, зависит от операции #01
• 20# R2mag > 12 – операция класс fc7, зависит от операции #01
```

- 21# $R2mag < 18$ - операция класс fc_7 , зависит от операции #01
- 22# $v_USNOB1_Data(x/[ra, de, name, magnitudes, quality, objectType, properMotion])$ - операция класса fc_1 , зависит от атрибутов $ra, de, name, magnitudes, quality, objectType, properMotion$, а следовательно от операций #01, #12, #17, #13, #15, #16, #18, #19, #20, #21

Для всех операций из выше описанного взгляда, назначение – взгляды. Для функциональных предикатов #02 - #15, язык реализации – ЯП, место реализации – ресурс ProgrammableA. Для функционального предиката #17, язык реализации не задается, место реализации – ресурс USNOB1, назначение не может быть изменено. В соответствии с данным взглядом строится следующая часть модели рассредоточения (рисунок 5.3.).

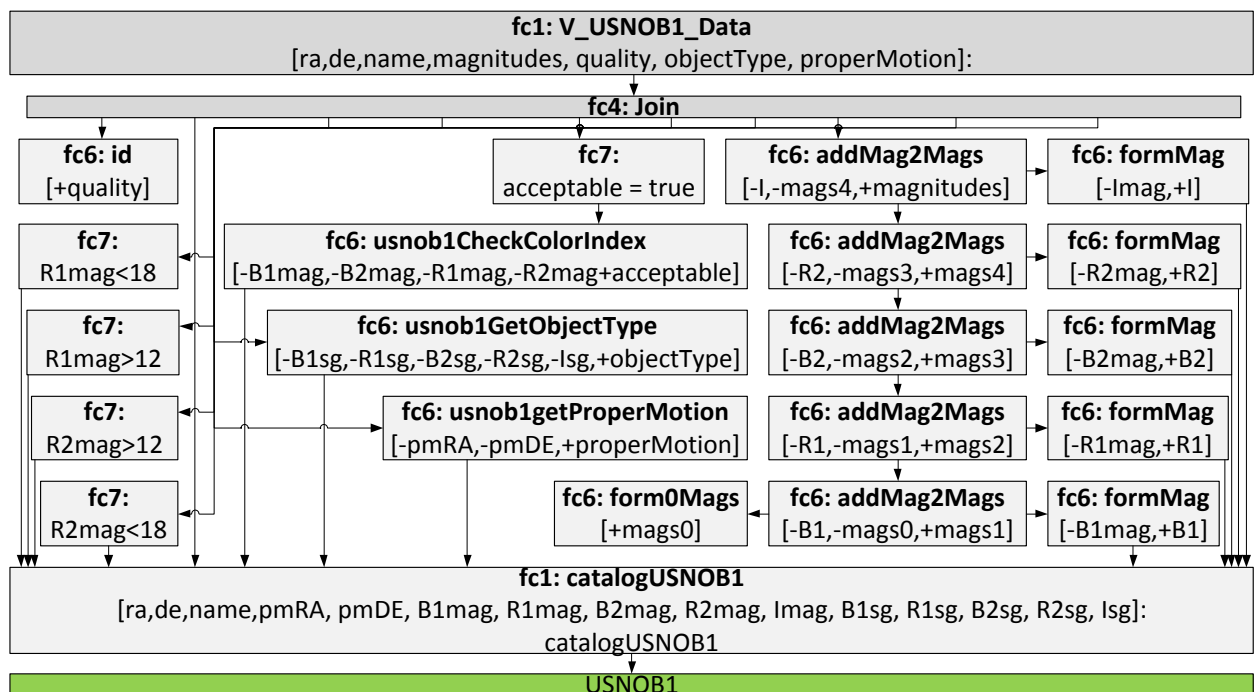


Рисунок 5.3. Фрагмент модели рассредоточения, соответствующий одному взгляду

Аналогичным образом в модель рассредоточения добавляются фрагменты (подграфы), соответствующие и другим взглядам. После того как все взгляды рассмотрены в модель рассредоточения добавляются классы посредника, участвующие в программе на правилах. После этого происходит добавление операций из правил. Правила рассматриваются последовательно. Рассмотрим программу на правилах, и то, как по ней операции добавляются в модель рассредоточения.

Rule 1#

- **00# astronomicalObject** – класс посредника
- **01# astronomicalObject**(x1/[ra: spatialCoord.ra, de: spatialCoord.de, objectType, properMotion, quality, magnitudes]) – операция класса fc_1 , зависит от операции #00
- **02# ra < queryRA + radius** – операция класса fc_7 , зависит от операции #01
- **03# ra > queryRA - radius** – операция класса fc_7 , зависит от операции #01
- **04# de < queryDE + radius** – операция класса fc_7 , зависит от операции #01
- **05# de > queryDE - radius** – операция класса fc_7 , зависит от операции #01
- **06# r(x/[ra, de, magnitudes, objectType, properMotion, quality])** – операция класса fc_1 , зависит от операции #01 - #05

Rule 2#

- **07# combineMagnitudes** (r/AstronomicalObject, r1) – операция класса fc_6 , зависит от операции #06

Rule 3#

- **08# getIsolated**(r1, r2) – операция класса fc_6 , зависит от операции #07

Rule 4#

- **09# r2**(x1/[ra, de, name, objectType, properMotion, quality, magnitudes]) – операция класса fc_1 , зависит от операции #08
- **10# checkType**(ra, de, 'G', nType) – операция класса fc_6 , зависит от операции #09
- **11# nType = false** – операция класса fc_7 , зависит от операции #10
- **12# objectType = Star** – операция класса fc_7 , зависит от операции #09
- **13# properMotion < 0.01** – операция класса fc_7 , зависит от операции #09
- **14# quality < 0.01** – операция класса fc_7 , зависит от операции #00
- **15# r3**(x/[ra, de, name, magnitudes]) – операция класса fc_1 , зависит от операции #09, #11 - #14

Rule 6#

- **16# r3**(x1/[ra, de, name, magnitudes]) – операция класса fc_1 , зависит от операции #15
- **17# isVariablebyMagnitudes**(ra, de, isVar) – операция класса fc_6 , зависит от операции #16
- **18# isVar = true** – операция класса fc_7 , зависит от операции #17
- **19# r4**(x/[ra, de, name]) – операция класса fc_1 , зависит от операции #16, #18

Rule 5#

- **20# variableStar** – класс посредника
- **21# variableStar**(x1/[ra: spatialCoord.ra, de: spatialCoord.de, name]) – операция класса fc_1 , зависит от операции #20
- **22# ra < queryRA + radius** – операция класса fc_7 , зависит от операции #21
- **23# ra > queryRA - radius** – операция класса fc_7 , зависит от операции #21
- **24# de < queryDE + radius** – операция класса fc_7 , зависит от операции #21
- **25# de > queryDE - radius** – операция класса fc_7 , зависит от операции #21
- **26# r4**(x/[ra, de, name]) – операция класса fc_1 , зависит от операции #21 - #25

Rule 7#

- **27# xmatch**(r3, r4, r5) – операция класса fc_6 , зависит от операции #26, #19, #15

Rule 8#

- **28# r5**(x1/[ra, de, name, magnitudes, distance]) – операция класса fc_1 , зависит от операции #27
- **29# distance > 0.01** – операция класса fc_7 , зависит от операции #28
- **30# r6**(x/[ra, de, name magnitudes]) – операция класса fc_1 , зависит от операции #28, #29

Rule 9#

- **31# r6**(x/ra, de, name, magnitudes]) – операция класса fc_1 , зависит от операции #30
- **32# showStadards**(ra, de, radius, magnitudes, im) – операция класса fc_6 , зависит от операции #31
- **33# r7**(im/Image) – операция класса fc_1 , зависит от операции #32

Операции добавляются в модель рассредоточения последовательно. Также добавляются дуги означающие зависимости между операциями. Стоит отметить, что в программе скрыта неявная операция дизъюнкции формул в правилах #5 и #6. В обоих правилах предикат класса назван одинаково “r4”, что

означает, что осуществляется объединение данных полученных в каждом из правил. Этот факт отражен уже в модели рассредоточения (рисунок 5.4.). У всех операций, которые были добавлены в модель рассредоточения из выше описанной программы, назначение – взгляды.

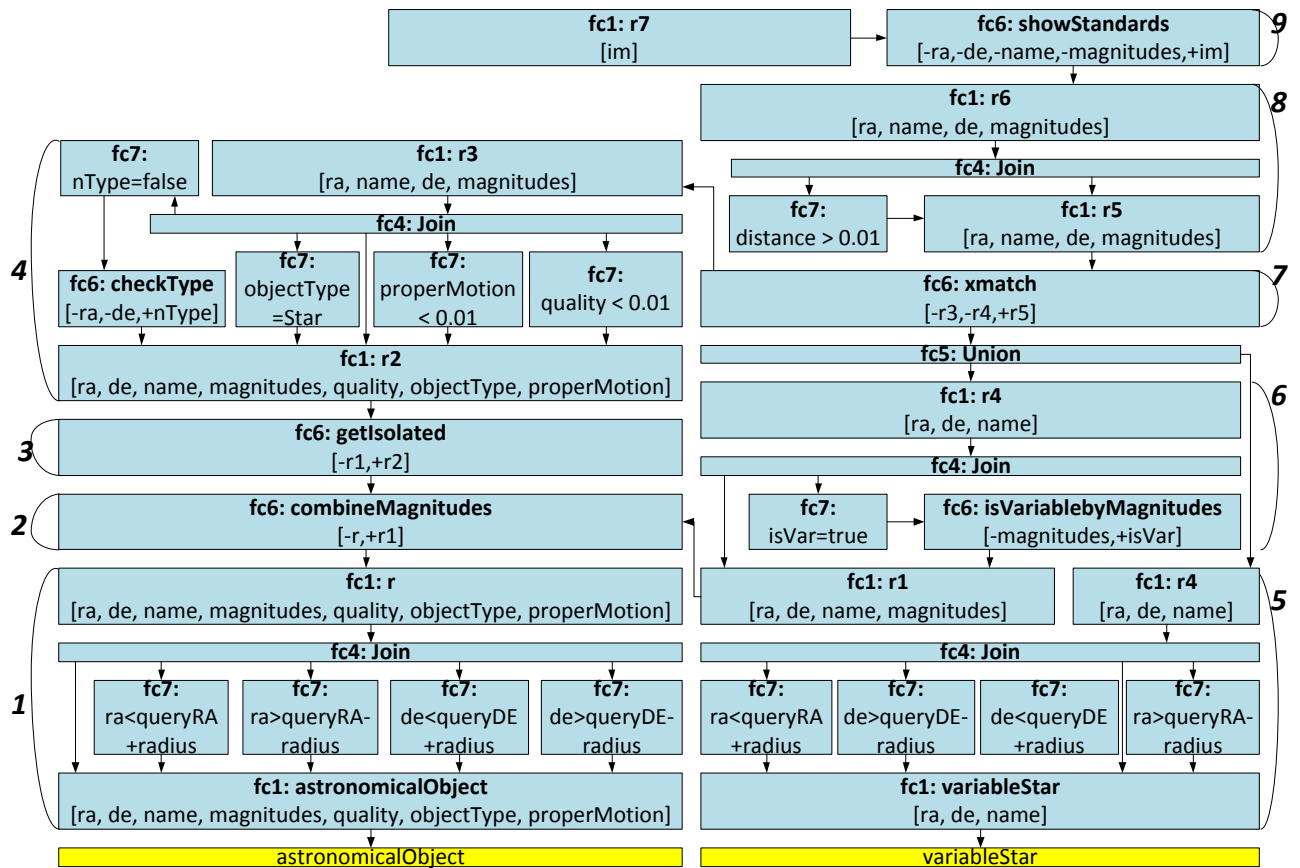


Рисунок 5.4. Фрагмент модели рассредоточения, соответствующий программе на правилах

Общая модель рассредоточения для задачи, включающая в себя все взгляды и программу на правилах, представлена на рисунке 5.2. Рассмотрим теперь автоматические перестановки в модели рассредоточения (применением экспертных правил), а также перестановки осуществляемые экспертом.

Во взгляде для функций $id(quality)$, $usnob1CheckColorIndex(acceptable)$, $usnob1getProperMotion(properMotion)$, $usnob1GetObjectTypes(objectTypes)$ в соответствии с экспертным правилом, описанным в 2.9., было принято решение что функции, реализованные на ЯП в стороннем ресурсе, должны быть

реализованы в адаптере ресурса. Ниже приведено экспертное правило Rule#08, примененное для этой операции.

$$\forall op \in DM ($$

$$FC(op) \in \{fc_6^{PL}, fc_6^{DB}\}$$

$$\& \exists z \in DM (z \neq op \& FC(z) = fc_7 \& (z, op) \in DM)$$

$$\rightarrow FC'(op) = fc_6^{PL-Wrapper})$$

Функциональный предикат реализован на ЯП. В модели рассредоточения присутствуют предикаты условий, зависящие от данного функционального предиката. В этом случае функция реализуется на ЯП в адаптерах (Рисунок 5.5.). Реализуемые в адаптере функции выделены зеленым цветом.

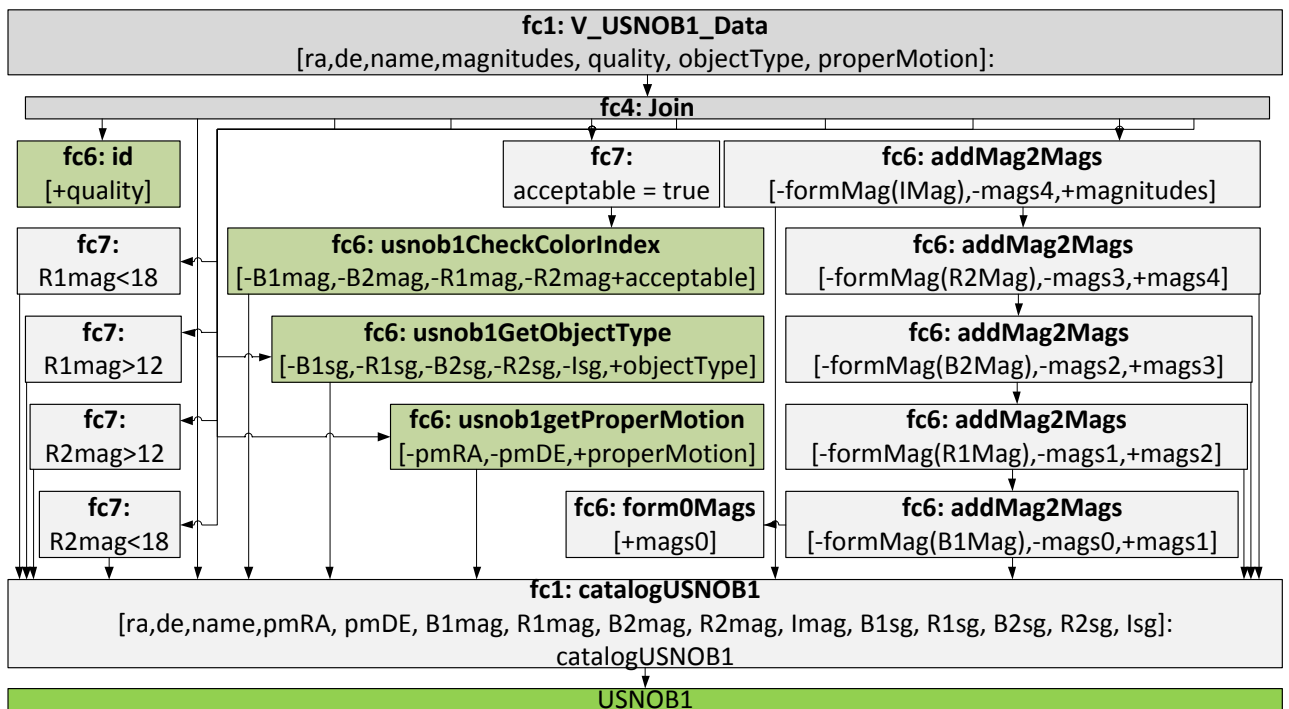


Рисунок 5.5. Перестановка реализации функций, слияние функций

Кроме того было применено правило Rule#13 по свертыванию операций, представленное ниже.

$$\forall op_1, \dots, op_n \in DM ($$

$$(op_1, op_2), \dots, (op_{n-1}, op_n) \in DM$$

$$\& \forall i, j (A(op_i) = A(op_j))$$

$$\& \neg \exists z \in DM (z \neq op_i \& ((z, op_2) \in DM \mid \dots \mid (z, op_n) \in DM))$$

$$\rightarrow op_1 \cup \dots \cup op_n = op)$$

Каскад функций *form0Mags*, *addMag2Mags*, *formMag* влечет накладные расходы. В модели рассредоточения нет операций условий связанных с этими

функциями, а следовательно автоматически операции не переставлялись. Кроме того, от этих функций не зависят другие операции, и все они реализованы на ЯП, а значит, в соответствии с экспертным правилом, они могут быть слиты в одну функцию. В начале сливаются функции `addMag2Mags` и соответствующие им функции `formMag` (рисунок 5.5). После чего каскад функций `addMag2Mags` был слит в одну функцию (Рисунок 5.6.). Кроме того, т.к. в модели рассредоточения присутствует операция `combineMagnitudes`, зависящая от результата выполнения данного каскада функций, то функция может быть выполнена в адаптере или посреднике. Алгоритмом направленного перебора, используя метод прогонки, для этой операции было установлено, что вариант реализации в адаптере предпочтительнее. Операции условий не требуют перестановки, таким образом, у всех операций во взгляде оптимальные назначения.

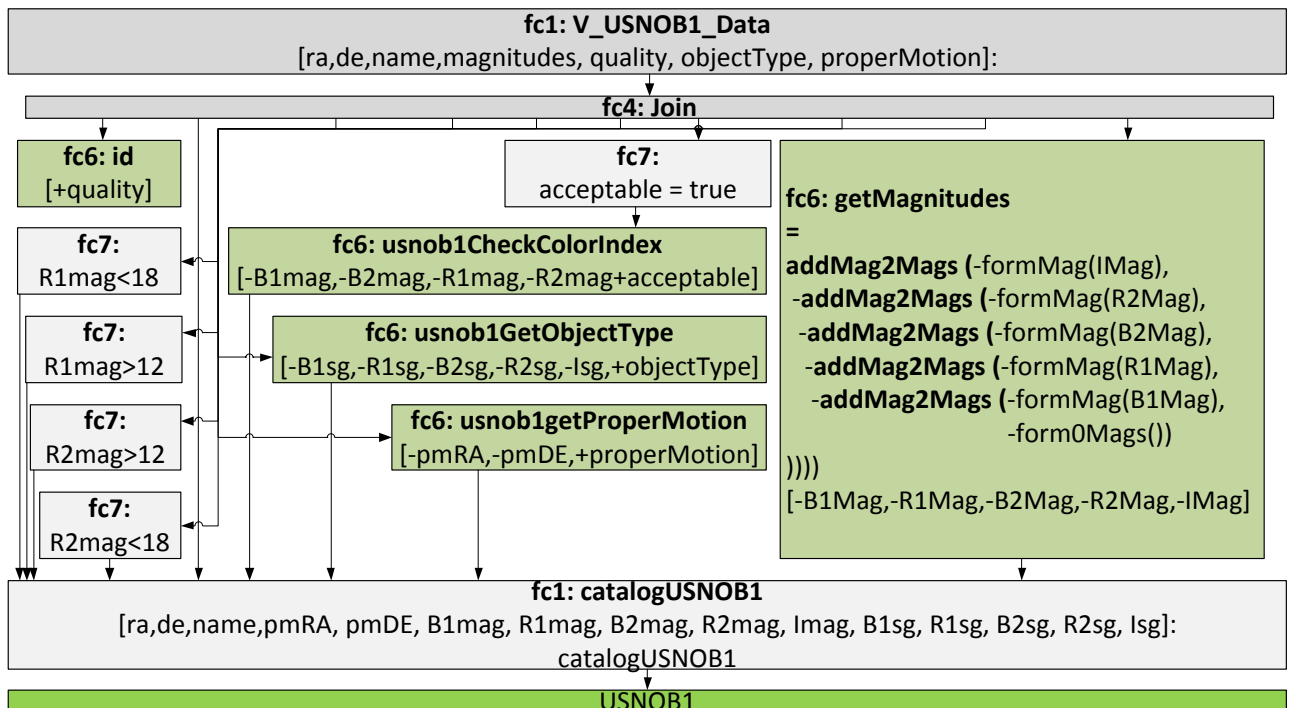


Рисунок 5.6. Объединение каскада функций в одну

Рассмотрим теперь преобразования операций в модели рассредоточения, соответствующие программе посредника. Правила 2, 3 и 7 это сложные вычислительные функции, обрабатывающие все множество объектов целиком,

в отличие от функций во взглядах рассмотренных выше. Применено экспертное правило Rule#09, представленное ниже.

$$\forall op \in DM ($$

$$FC(op) = fc_6^{PL} \ \& \ InputParameterCollection(op) = true$$

$$\rightarrow FC'(op) = fc_6^{PL-SP})$$

В соответствии с правилом, подобные функции выгоднее выполнять в ЯП (рисунок 5.7.) а не в посреднике, нивелировав тем самым лишние накладные расходы, связанные с загрузкой и выгрузкой данных в/из СУБД посредника.

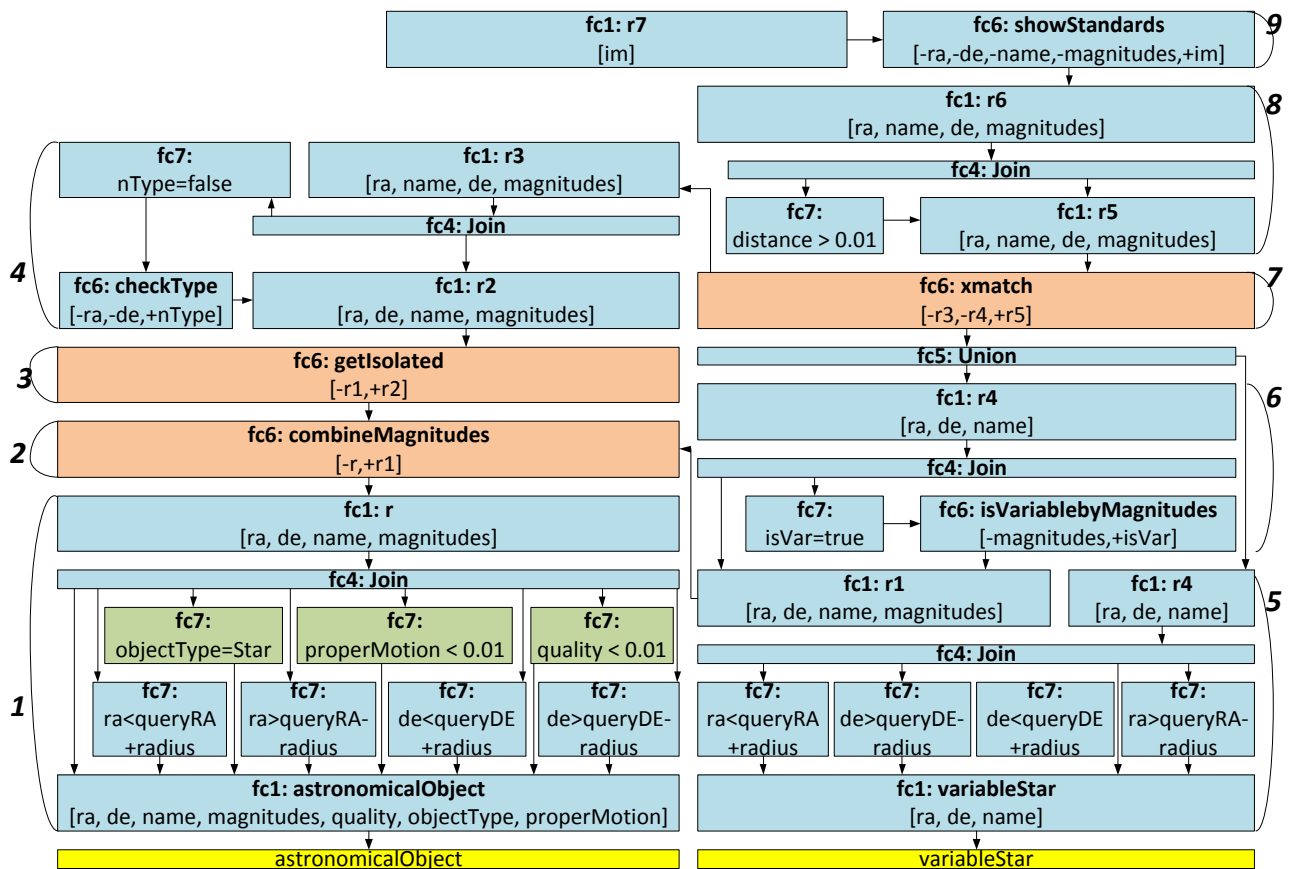


Рисунок 5.7. Перестановка условий, и назначений для функций

В 4ом правиле требуется, чтобы коллекция объектов была в СУБД посредника. Над коллекцией выполняются три условия, и вычисляется функция. Может быть применено экспертное правило Rule#12.

$$\forall op_1, \dots, op_n \in DM ($$

$$FC(op_1) = fc_7 \ \& \ (op_1, op_2)_x, \dots, (op_{n-1}, op_n)_x \in DM$$

$$\rightarrow (op_1, op_n)_x \in DM \ \& \ \neg (op_1, op_2)_x \in DM)$$

В соответствии с этим правилом, операции усечения множества по условию стоит выполнять как можно “раньше”. Таким образом, условия

перемещаются из правила 4го в 1ое правило (рисунок 5.7.). Это уменьшает как общее число объектов, так избавляет от необходимости передавать эти данные следующим операциям, т.к. более нигде эти параметры не используются. Благодаря этой перестановке и тому, что функции, вычисляющие значения для *quality*, *properMotion* и *objectType* реализуются теперь в адаптерах ресурсов, условия, накладываемые на эти параметры, могут быть выполнены на ресурсе.

4ое и 6ое правила представляют собой вызов функции и выполнение условия, накладываемого на результат функции. Выполнение подобного правила требует загрузки коллекции объектов в СУБД посредника, затем выполнение функции для каждого объекта и помещение результата в СУБД посредника. Затем выполняется выборка данных по условию, накладываемому на результат функции, и передача данных следующей операции. Подобное выполнение неэффективно (рисунок 5.8).

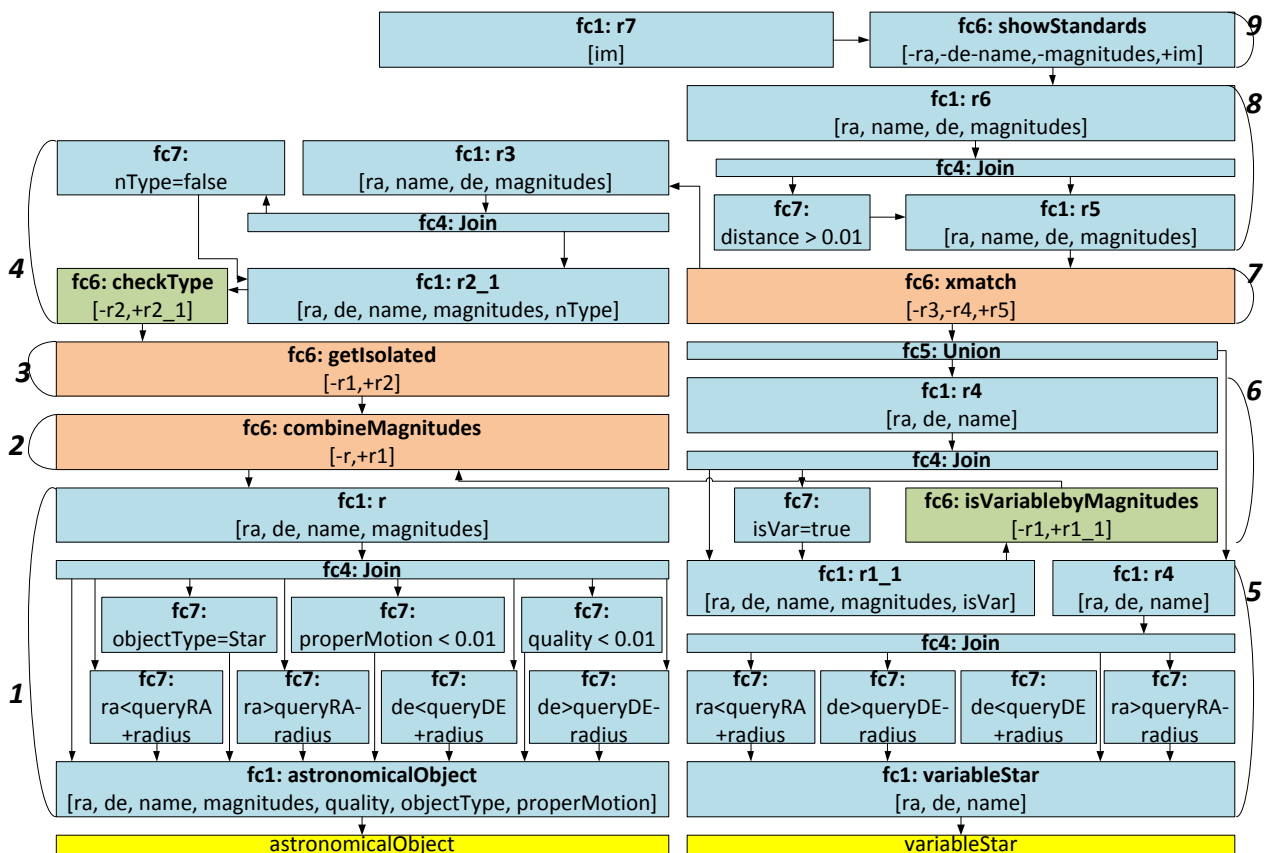


Рисунок 5.8. Изменение реализации функций, для поддержки работы с коллекциями объектов

Может быть применено экспертное правило Rule#07. В 4ом правиле, функция реализована веб-сервисом. В 6ом правиле функция реализована на ЯП, но не умеет принимать коллекции. Повышение производительности достигается реализацией подобных функций в ЯП, не над отдельными объектами, а над коллекцией целиком (рисунок 5.8.).

$\forall op \in DM ($
 $FC(op) \in \{fc_6^{WS}, fc_6^{PL}\} \& A(op) = R$
 $\& InputParameterCollection(op) = false$
 $\rightarrow FC'(op) = fc_6^{PL} \& InputParameterCollection(op) = true)$

Далее для операций 4го, 6го и 8го правил применялся алгоритм направленного перебора, которым было установлено, что оптимальным является выполнение как функций так и операций усечения множества по условию в ЯП (рисунок 5.9.).

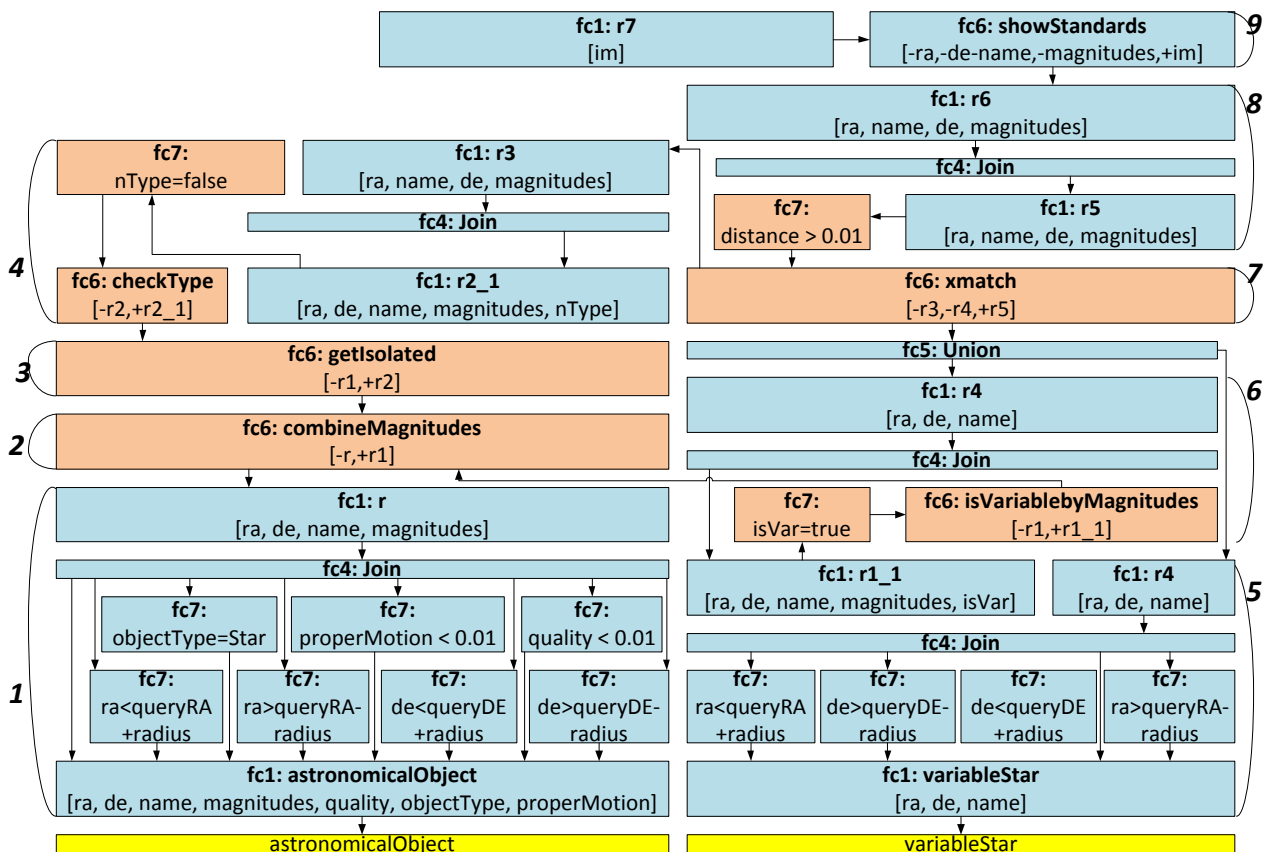


Рисунок 5.9. Перестановка функций и операций усечения множества по условию в ЯП

После этой перестановки в 4ом, 6ом и 8ом правиле, присутствуют каскады операций, которые ничего не делают ($fc1:r2_1 \rightarrow fc4:Join \rightarrow fc1:r3$, $fc1:r1_1 \rightarrow fc4:Join \rightarrow fc1:r4$, $fc1:r5 \rightarrow fc4:Join \rightarrow fc1:r6$). Данные операции были убраны экспертом. Но они могут быть убраны и автоматически, алгоритмом направленного перебора. Кроме того в 4ом, 6ом, 7ом и 8м правиле в соответствии с экспертным правилом, используемым ранее во взглядах, связки функция и условие, объединяются в одну операцию. Полученные функции среди объектов коллекции отбирают те, которые удовлетворяют условиям. Наконец алгоритмом направленного перебора установлено, что оптимальное назначение для операции *Union* – ЯП (рисунок 5.10).

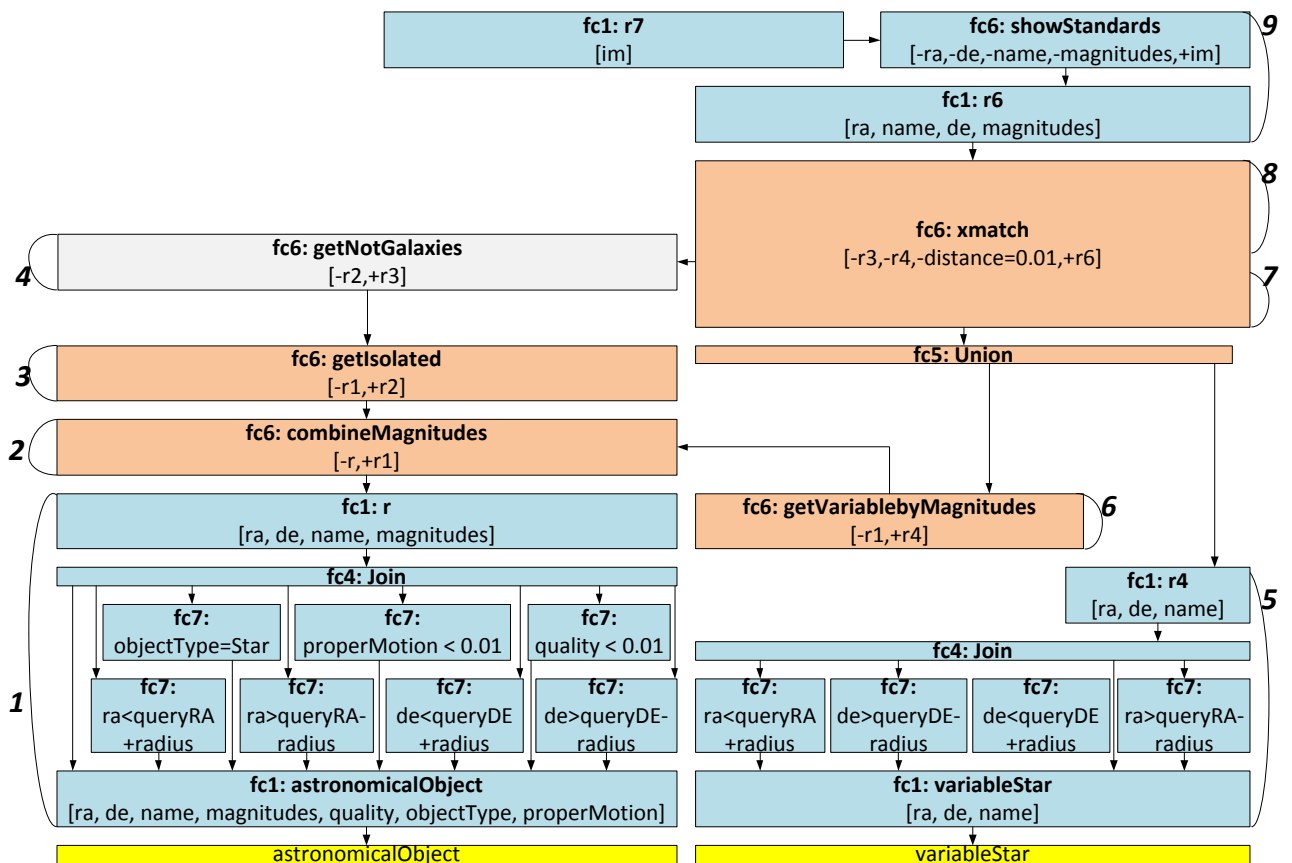


Рисунок 5.10. Оптимизация 4го, 6го, 7го и 8го правил

Методом прогонки установлено, что значительная часть накладных расходов приходится на выполнение 4го и 9го правил. В 4ом правиле проблема заключается в том, что функция для определения является ли объект галактикой, обращается к удаленному веб-сервису NED. Таким образом, для п

объектов, требуется n обращений. Это влечет за собой большие накладные расходы. Решение найдено экспертом. Вместо того чтобы для каждого объекта проверять, является ли он галактикой, можно выбрать все галактики в данной области неба за одно обращение к веб-сервису. После чего при проверке, является ли объект галактикой, осуществляется обращение не к удаленному сервису, а к уже загруженному массиву данных (рисунок 5.11.).

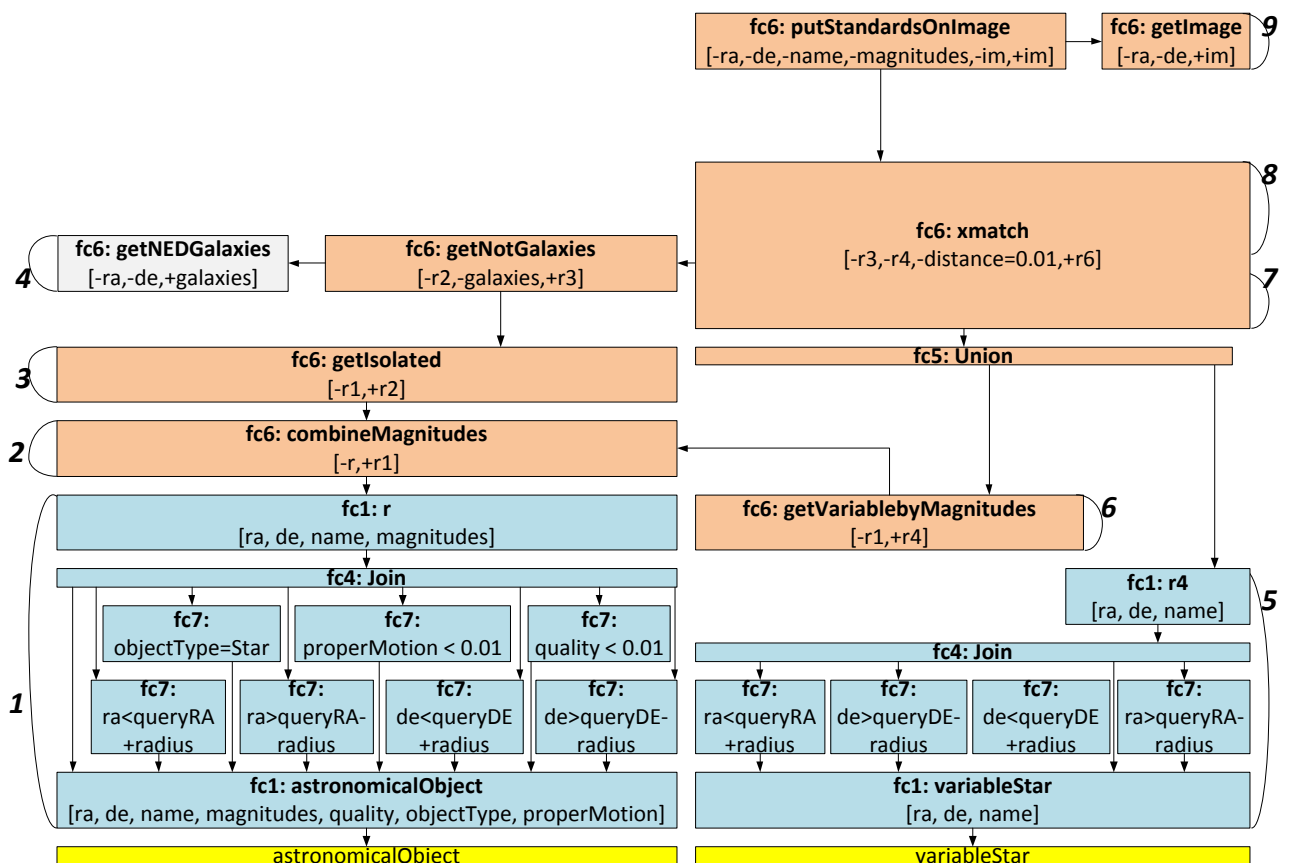


Рисунок 5.11. Результирующая модель рассредоточения

В 9ом правиле проблема заключается в узком канале при загрузке изображения. Функция может быть разбита на две составляющие. Первая – загрузка самого изображения, эта функция не зависит от результата выполнения 7го-8го правил, а зависит только от заранее известных координат. И вторая – наложение на изображение объектов получаемых в 7ом-8ом правиле. Вторая составляющая выполняется быстро и не вносит накладных расходов. Таким образом, результирующая модель рассредоточения

представлена на рисунке 5.11. При генерации кода сначала генерируются взгляды по соответствующим фрагментам модели рассредоточения.

GAV взгляды:

```
// GAV for SDSS
v_SDSS_Data(x/[ra, de, name, magnitudes, quality, objectType, properMotion])
:- CATALOG_SDSS.catalogSDSS(x/[ra:RAJ2000, de:DEJ2000, name:SDSS, umag, gmag, rmag, imag, zmag,
Q, cl])
& CATALOG_SDSS.getMagnitudes(umag, gmag, rmag, imag, zmag, magnitudes)
& CATALOG_SDSS.sdssGetQuality(Q, quality)
& CATALOG_SDSS.sdssGetObjectTypes(cl, objectType)
& CATALOG_SDSS.sdssCheckColorIndex(umag, gmag, rmag, imag, zmag, acceptable)
& acceptable = true
& id(0, properMotion)
& rmag > 15 & rmag < 20

// GAV for 2MASS
v_2MASS_Data(x/[ra, de, name, magnitudes, quality, objectType, properMotion])
:- CATALOG_2MASS.twomass_psc(x/[ra:RAJ2000, de:DEJ2000, name:@'2MASS', Kmag, Jmag, Hmag, Qflg,
Rflg, Bflg, Cflg, Xflg, Aflg])
& CATALOG_2MASS.getMagnitudes(Kmag, Jmag, Hmag, magnitudes)
& CATALOG_2MASS.c2MassGetQuality(Qflg, Rflg, quality)
& CATALOG_2MASS.getUnknownObjectType(objectType)
& Bflg = '111'
& Cflg = '000'
& Xflg = 0
& Aflg = 0
& id(0, properMotion)
& Jmag > 12 & Jmag < 18

// GAV for USNOB1
v_USNOB1_Data(x/[ra, de, name, magnitudes, quality, objectType, properMotion])
:- CATALOG_USNOB1.catalogUSNOB1(x/[ra:RAJ2000, de:DEJ2000, name:@'USNO-B1.0', pmRA, pmDE, B1mag,
R1mag, B2mag, R2mag, Imag, B1sg: '@'B1s/g', R1sg: '@'R1s/g', B2sg: '@'B2s/g', R2sg: '@'R2s/g', Isg:
@'Is/g'])
& CATALOG_USNOB1.getMagnitudes(B1mag, R1mag, B2mag, R2mag, Imag, magnitudes)
& CATALOG_USNOB1.usnob1GetObjectTypes(B1sg, R1sg, B2sg, R2sg, Isg, objectType)
& CATALOG_USNOB1.usnob1CheckColorIndex(B1mag, R1mag, B2mag, R2mag, acceptable)
& CATALOG_USNOB1.usnob1getProperMotion(pmRA, pmDE, properMotion)
& acceptable = true
& id(0, quality)
& R1mag > 12 & R1mag < 18
& R2mag > 12 & R2mag < 18

// GAV for VSX
v_VSX_Data(x/[ra, de, name])
:- CATALOG_VSX.catalogVSX(x/[ra:RAJ2000, de:DEJ2000, name:Name])

// GAV for ASAS
v_ASAS_Data(x/[ra, de, name])
:- CATALOG_ASAS.catalogASAS(x/[ra:_RA, de:_DE, name:ASAS])

// GAV for GSC
v_GSC_Data(x/[ra, de, name, magnitudes, objectType])
:- CATALOG_GSC.catalogGSC(x/[ra:RAJ2000, de:DEJ2000, name:@'GSC2.3', Vmag])
& CATALOG_GSC.getMagnitudes(Vmag, magnitudes)
& CATALOG_GSC.getUnknownObjectType(objectType)

// GAV for UCAC
v_UCAC_Data(x/[ra, de, name, magnitudes, objectType])
:- CATALOG_UCAC.catalogUCAC(x/[ra:RAJ2000, de:DEJ2000, name:@'3UC', fmag: '@'f.mag'])
& CATALOG_UCAC.getMagnitudes(fmag, magnitudes)
& CATALOG_UCAC.getUnknownObjectType(objectType)
```

LAV взгляды остались без изменений.

После того как взгляды построены, соответствующие им операции удаляются из модели рассредоточения. Для каждого связного подграфа с назначением – программа посредника, генерируются соответствующие правила. В данном примере генерируются два правила (1 и 5). После этого каждый подграф, для которого построено правило, заменяется в модели рассредоточения одной операцией (Рисунок 5.12).

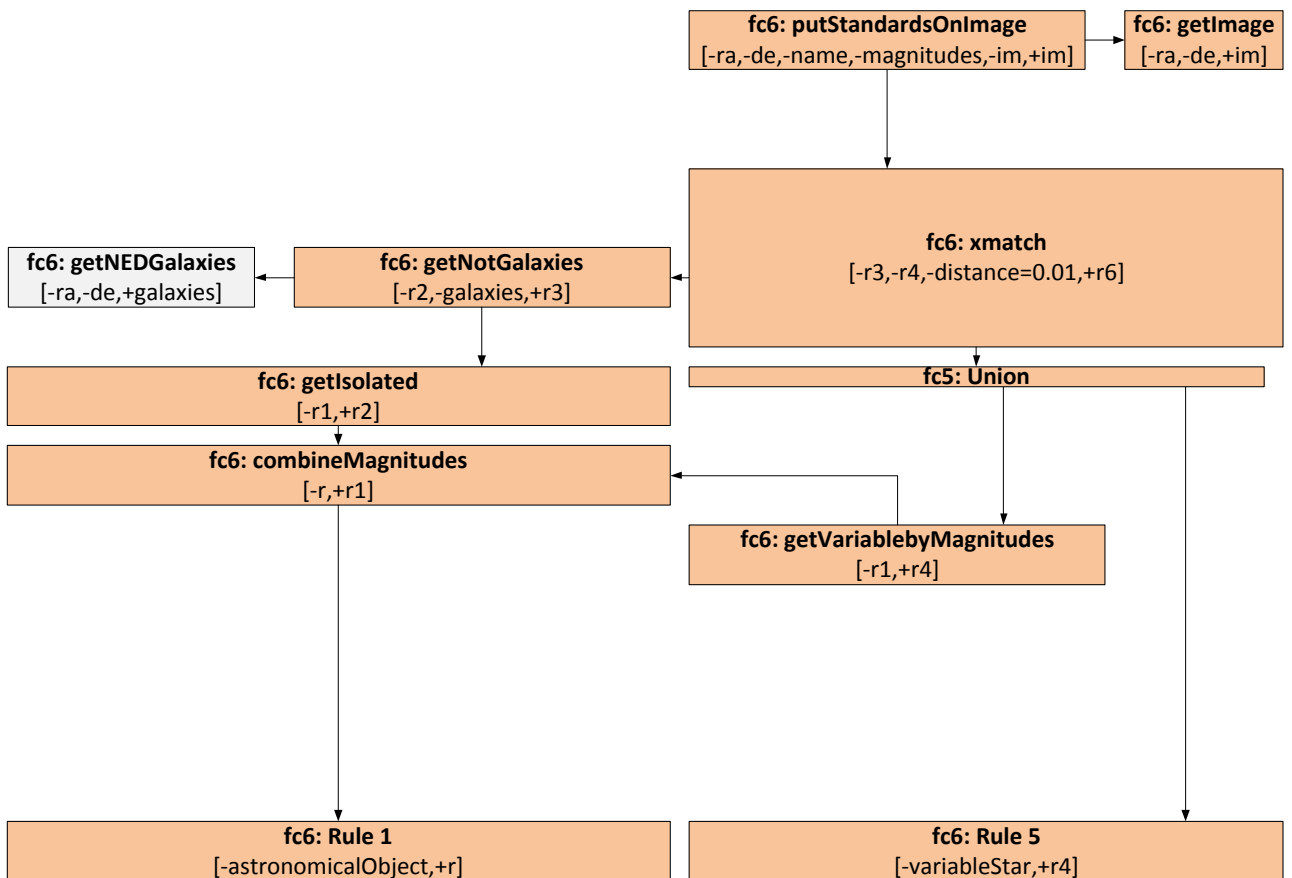


Рисунок 5.12. Модель рассредоточения после генерации кода для правил
Полученный граф определяет выполнение алгоритма решения задачи.
Более наглядно та же самая модель рассредоточения представлена на рисунке 5.13. Операции и связи на рисунке 5.13. полностью совпадают с рисунком 5.12.

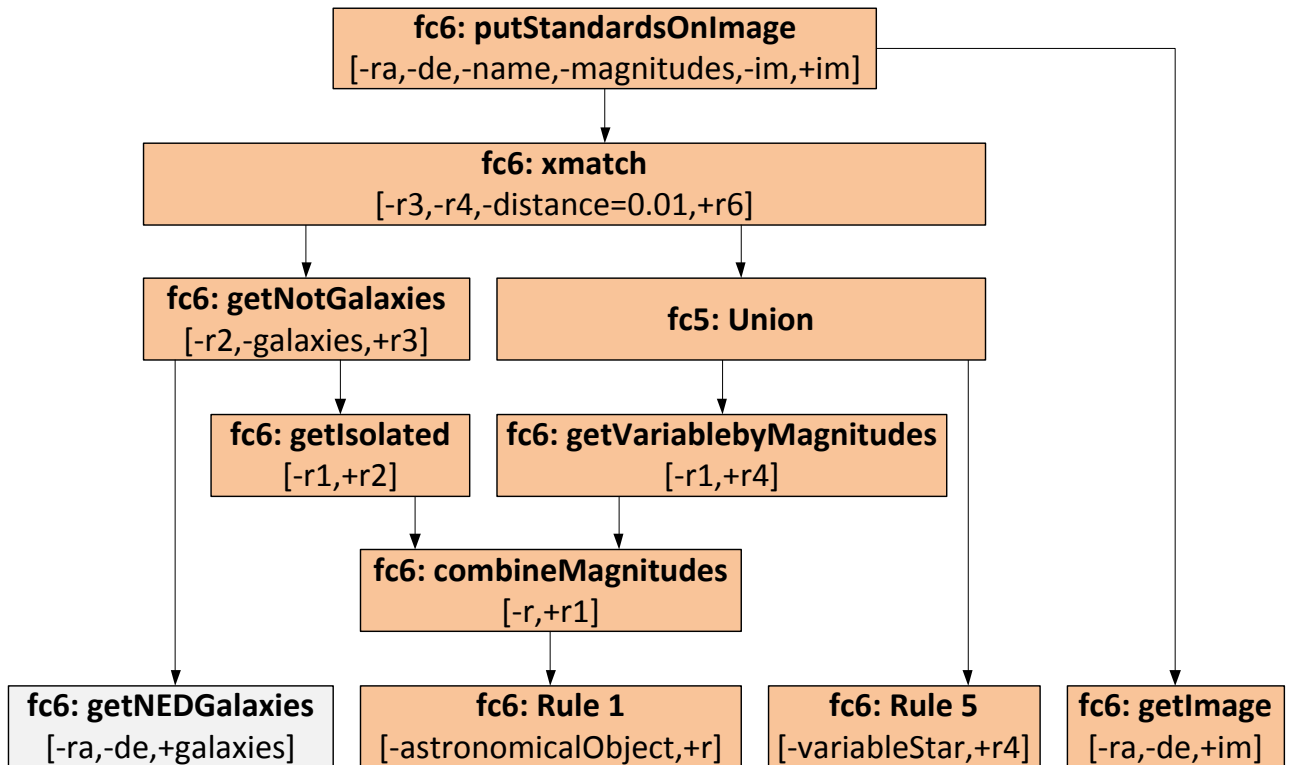


Рисунок 5.13. Граф выполнения задачи

В соответствии с графом выполнения генерируется код на ЯП. Вначале генерируется код для независимых операций (они могут выполняться одновременно). Затем в соответствии с зависимостями генерируются и другие операции. Итоговая программа на ЯП (Java) и языке правил (полученная по модели рассредоточения, изображенной на рисунке 5.12) представлена ниже:

```

class Class_r {
    double      ra;
    double      de;
    String      name;
    Set<Magnitude> magnitudes;
}
class Class_r4 {
    double      ra;
    double      de;
    String      name;
}

double queryRA = ...;
double queryDE = ...;
double radius = 0.01;

1 # Synchronized Parallel
  Aladin.getImage(queryRA, queryDE, radius);

2 # Synchronized Parallel
  Set<String> galaxies = GetNEDGalaxies(queryRA, queryDE, radius);

3 # Synchronized Parallel
  #SYNTHESIS
  
```

```

Class r(x/[ra, de, name, magnitudes])
:- astronomicalObject(x1/[ra: spatialCoord.ra, de: spatialCoord.de, name, objectType,
properMotion, quality, magnitudes])
& ra < queryRA + radius & ra > queryRA - radius
& de < queryDE + radius & de > queryDE - radius;
& objectType = Star
& properMotion < 0.01
& quality < 0.01;
#SYNTHESIS

4 # Synchronized Parallel
#SYNTHESIS
Class r4(x/[ra, de, name])
:- variableStar(x1/[ra: spatialCoord.ra, de: spatialCoord.de, name]);
& ra < queryRA + radius & ra > queryRA - radius
& de < queryDE + radius & de > queryDE - radius;
#SYNTHESIS

5 Set<Class_r> r1 = combineMagnitudes (r);

6 # Synchronized Parallel
Set<Class_r> r2 = getIsolated(r1);
Set<Class_r> r3 = getNotGalaxies(r2, galaxies);

7 # Synchronized Parallel
Set<Class_r4> r5 = getVariablebyMagnitudes(r1);
r5.add(r4);

8 Set<Class_r> r6 = xmatch(r3, r5, 0.01);

9 Aladin.putStandardsOnImage(r6);

```

В соответствии со статическим подходом связывания языков программирования и языка правил предметных посредников (подробное описание представлено в разделе 3.4.1.), для правила №3 и №4 были сформированы типы результата, что позволяет использовать типизированные коллекции, значительно превосходящие в производительности, динамически создаваемые классы.

```

class Class_r {
    double          ra;
    double          de;
    String          name;
    Set<Magnitude> magnitudes;
}
class Class_r4 {
    double          ra;
    double          de;
    String          name;
}

```

Программа реализации алгоритма решения задачи состоит из последовательных и параллельных частей. Вначале, запускаются четыре параллельных потока №1, 2, 3, 4.

Первый поток загружает изображения посредством инструментария Aladin (т.к. этот процесс может занимать ощутимое время). Этот метод соответствует

исходному правилу 9, в котором функция *showStandards* представляет собой загрузку изображения посредством Aladin, после чего на изображение накладываются найденные стандарты. Стоит отметить, что процесс наложения стандартов выполняется быстро, таким образом, подобное преобразование позволило сократить общее время выполнения, на время, затрачиваемое на загрузку изображения. Кроме того в исходной реализации изображение в начале загружалось на сторонний ресурс, откуда уже передавалось в посредник. Эти затраты нивелируются т.к. метод вызывается напрямую из языка программирования.

Во втором потоке загружаются объекты являющиеся галактиками, посредством сервиса NED.

```
Set<String> galaxies = GetNedGalaxies(queryRA, queryDE, radius);
```

Третий поток представляет собой правило №1 исходной программы, в которое, как было сказано выше, добавлены условия из правила №4 исходной программы. В правиле загружаются объекты из указанной области, и удовлетворяющие некоторым условиям.

Четвертый поток представляет собой правило №5 исходной программы, в котором загружаются переменные объекты из указанной области.

Пятая конструкция программы объединяет магнитуды из класса *g*, получаемого в третьем потоке, а следовательно дожидается его завершения. Стоит отметить, что эта операция не блокирует выполнение остальных потоков.

После выполнения конструкции №5, запускаются два потока №6 и №7.

В шестом потоке среди астрономических объектов, удовлетворяющих ряду условий (см. поток №3), отбираются изолированные объекты и не являющиеся галактиками. Для определения принадлежности к галактикам, используется список имен, составленный во втором потоке, а следовательно эта операция дожидается завершения второго потока.

В седьмом потоке среди астрономических объектов, удовлетворяющих ряду условий (см. поток №3), отбираются объекты являющиеся переменными

исходя из условий, накладываемых на магнитуды. После чего, к множеству этих объектов, добавляются объекты, полученные в четвертом потоке. Эта операция дожидается завершения четвертого потока.

В восьмой конструкции из множества кандидатов полученных в шестом потоке, отсеиваются те, которые являются переменными. Переменность определяется с помощью множества объектов составленного в седьмом потоке. Следовательно, эта операция дожидается завершения, как шестого, так и седьмого потоков.

Стоит отметить, что все функции из конструкций программы №5-№8 реализованы на ЯП, и выполняются в памяти, что существенно повышает производительность, по сравнению с их реализацией на стороннем ресурсе.

Наконец, результат накладывается на изображение, полученное в первом потоке, и готовое изображение возвращается пользователю. Подробно оценки производительности приводятся в пятой главе, стоит отметить, что подобное изменение программы позволило улучшить среднее время выполнения почти в 10 раз.

Подводя итог, с помощью системы построения рассредоточений для данной задачи были выполнены следующие действия:

- **автоматически** построена модель рассредоточения по заданному начальному алгоритму решения задачи;
- функции *id(quality)*, *usnob1CheckColorIndex(acceptable)*, *usnob1getProperMotion(properMotion)*, *usnob1GetObjecttype(objectType)* во взглядах назначены для реализации на ЯП в адаптерах **с помощью экспертных правил**;
- **экспертным правилом** каскад функций *form0Mags*, *addMag2Mags*, *formMag* во взглядах слит в одну функцию *getMagnitudes*, назначение для которой определено **автоматически методом направленного перебора**;
- функции *getIsolated*, *combineMAGnitudes*, *xmatch* назначены для реализации в СП **с помощью экспертного правила**;

- операции условий для атрибутов *objectType*, *properMotion*, *quality* перенесены ближе к операциям, продуцирующим данные, с помощью **экспертного правила**;
- реализация функций *checkType* и *isVariablebyMagnitudes* изменена **автоматически с помощью экспертного правила** для возможности обработки коллекций, а не отдельных объектов;
- **алгоритмом направленного перебора** для функций *checkType*, *isVariablebyMagnitude*, операции *Union*, а также для условий накладываемых на атрибуты *nType*, *distance*, *isVar* было определено оптимальное назначение – СП;
- каскады операций *fc1:r2_1->fc4:Join->fc1:r3*, *fc1:r1_1->fc4:Join->fc1:r4*, *fc1:r5->fc4:Join->fc1:r6* были выброшены из модели рассредоточения **автоматически** при генерации кода;
- в соответствие с **экспертным правилом** пары функция-условие **автоматически** слиты в одну функцию (*checkType+nType->getNotGalaxies*, *isVariablebyMagnitude+isVar->getVariablebyMagnitudes*, *xmatch+distance->xmatch*);
- методом прогонки **автоматически** определено, что функции *getNotGalaxies* и *ShowStandards* влекут значительные накладные расходы;
- **экспертом** каждая из этих функций разбита на две составляющие, методом прогонки определено, что подобное разбиение повышает производительность;
- по результирующей модели рассредоточения **автоматически** построена программа, реализующая эффективное выполнение алгоритма решения задачи.

Таким образом, с помощью системы построения рассредоточений, удалось автоматизировать все построение итоговой модели рассредоточения, кроме разбиения двух функций, выполненного экспертом.

5.3. Описание процесса тестирования алгоритмов построения эффективного рассредоточения

Нам известно как устроены алгоритмы построения эффективного рассредоточения, поэтому тестирование проводилось методом «белого ящика». В этом случае, в задачу тестировщика входит проверка тестируемых алгоритмов такими тестами, чтобы проработал каждый блок кода.

Задача разработки тестовых примеров сама по себе неоднозначна, т.к. синтетические примеры не всегда отражают действительность. Поэтому алгоритмы построения эффективного рассредоточения тестировались как на синтетических примерах, так и на реальной задаче, описанной в разделе 1.5.

5.3.1. Описание тестовых примеров

Рассматриваются две тестовые задачи. Первая задача – синтетическая. В задаче определяется посредник, состоящий из трех классов. В каждом классе описано по одной переменной одного типа данных. Посредник описывается тремя классами, в каждый из классов зарегистрировано по одному ресурсу. В программе к посреднику запрашивается конъюнкция всех трех классов посредника, что влечет за собой выполнение двух операций соединения. Каждая такая операция может быть выполнена на каждом из трех ресурсов, либо в посреднике. Если не учитывать порядок выполнения соединения, а ограничиваться учетом только места выполнения операции, то всего вариантов: $(n+1) + (n+1)!/2$. В случае $n = 3$ (число ресурсов), имеем 16 вариантов. Если же учитывать порядок то формула намного сложнее, и вариантов еще больше. Кроме того в посреднике описано 2 функции модуля, а также 3 функции разрешения конфликтов, по каждой на ресурс. Для каждой из функций определены все возможные варианты реализаций. Функции модуля могут быть реализованы веб-сервисом, на ЯП, на PL/SQL, процедурой ресурса. Функции разрешения конфликтов могут быть реализованы на PL/SQL и на ЯП. К посреднику задается программа на языке правил. В программе выбираются все

данные из трех классов, удовлетворяющие простому условию, а также вызываются обе функции. Во взглядах при регистрации используются все три функции разрешения конфликтов. Во взглядах задано по условию на выборку данных из ресурса.

Все выбранные функции – тривиальны, чтобы их выполнение не вносило вклад в общее время, а были наиболее заметны издержки связанные с выполнением функций. Объем данных в ресурсах ограничивается десятью тысячами (10 000) записей. Спецификация тестового примера представлена в Приложении Е.

Вторая задача – реальная научная задача в области астрономии. Задача заключается в определении вторичных стандартов для фотометрической калибровки оптических компонентов космических гамма-всплесков. Подробное описание самой задачи дано в разделе 1.5. После описания схемы посредника были определены астрономические ресурсы, релевантные решаемой задаче. Каталоги SDSS, USNOB-1, 2MASS, GSC, UCAC – основные ресурсы, используемые для извлечения стандартов. Именно среди этих каталогов отбираются все звезды удовлетворяющие параметрам, описанным в ТЗ. Каталоги VSX, ASAS, GCVS, NSVS – используются для проверки факта переменности выбранных стандартов.

5.3.2. *Набор тестов*

В качестве тестов важно проверить ключевые особенности алгоритмов (масштабируемость, устойчивость, производительность). В случае алгоритма полного перебора важно оценить возможность построения минимального рассредоточения. В случае направленного перебора сравнить насколько рассредоточение хуже минимального, аналогичный тест интересен и в случае применения экспертных правил. Таким образом, рассматривались следующие тесты:

Тест производительности алгоритмов построения рассредоточений

Тест заключается в том, что задается начальное рассредоточение, описанное в предыдущем разделе 5.3.1. а также в разделе 1.5., после чего ищется минимальное рассредоточение полным перебором. Т.к. для каждого рассредоточения получается оценка времени выполнения, то можно произвести анализ этих значений. В частности интересно сравнение времени выполнения минимального рассредоточения со временем выполнения худшего рассредоточения (с максимальным временем выполнения), а также с рассредоточениями, получаемыми другими алгоритмами и с начальным рассредоточением.

Тест масштабируемости при поиске рассредоточения

Производительность полного перебора всегда низкая. Тест заключается в проверке масштабируемости полного перебора по числу классов посредника, числу ресурсов и числу функций, при фиксированном алгоритме решения задачи. Также тестируется масштабируемость по числу операций в модели рассредоточения при фиксированных параметрах посредника (классы, ресурсы, функции). Аналогично тестируется и алгоритм направленного перебора.

Тест устойчивости минимального рассредоточения

Тест заключается в том, что варьируется объем данных в ресурсах, и проверяется всегда ли одно и то же рассредоточение является минимальным. Аналогично тестируется и алгоритм направленного перебора.

5.3.3. Результаты тестирования

Тестирование представлено в следующей сводной таблице (таблица 5.1.). Каждое время выполнения определялось как среднее из 5 запусков, цифры в секундах. Зависимость времени поиска рассредоточения во всех случаях считается относительно среднего времени выполнения алгоритма решения задачи.

Таблица 5.1. Результаты тестирования производительности

	Синтетическая задача	Реальная научная задача
Время выполнения худшего рассредоточения	217 секунд	1093 секунды
Время выполнения начального рассредоточения	48 секунд	317 секунд
Время выполнения минимального рассредоточения	11 секунд	34 секунды
Время выполнения эффективного рассредоточения (направленный перебор)	13 секунд	52 секунды
Время выполнения рассредоточения построенного с помощью экспертных правил	13 секунд	274 секунды
Время поиска минимального рассредоточения	3 часа 50 минут	Около 38 часов
Время поиска эффективного рассредоточения (направленным перебором)	15 минут	52 минуты
Время построения рассредоточения применением экспертных правил	Меньше минуты	

Результаты тестирования показывают, что в синтетическом тесте, когда нет накладных расходов на выполнение функций и передачу данных по сети, полным перебором удалось сократить время примерно в 4 раза. Направленный перебор также дал хорошие результаты, время выполнения сократилось в 3.5 раза, но время построения такого рассредоточения в 15 раз меньше, нежели полным перебором. Подобный же результат получен и при применении экспертных правил. Алгоритм построения эффективного рассредоточения с помощью экспертных правил не использует симуляций методом прогонки, поэтому подобное построение всегда занимает мало времени.

В реальной научной задаче алгоритмом полного перебора удалось сократить время выполнения задачи почти в 10 раз. Направленным перебором удалось сократить время выполнения в 6 раз. Это в полтора раза хуже, чем рассредоточение, построенное полным перебором, но время, затраченное на

построение направленным перебором, в 40 раз меньше времени полного перебора. Применение экспертных правил улучшило время незначительно. Это объясняется тем, что невозможно заранее учесть все особенности среды выполнения, именно поэтому важны реальные оценки производительности операций с тем или иным назначением методом прогонки. Учитывая, что полный перебор занимает существенное время, лучший результат можно достичь путем совместного использования алгоритма, основанного на экспертных правилах, и алгоритма направленного перебора.

Также тестировалось поведение алгоритмов при изменении параметров системы. Зависимости представлены в таблице 5.2.

Таблица 5.2. Результаты тестирования масштабируемости и устойчивости

	Синтетическая задача	Реальная научная задача
Зависимость времени поиска рассредоточения при увеличении числа классов посредника	не зависит	
Зависимость времени поиска рассредоточения при увеличении числа ресурсов	экспоненциальная	линейная
Зависимость времени поиска рассредоточения при увеличении числа функций	степенная	
Зависимость времени поиска рассредоточения при увеличении числа операций в модели рассредоточения	3^n – полный перебор $3*n$ – направленный перебор	
Устойчивость минимального рассредоточения	Одно-и-тоже	Различное

Линейная зависимость времени поиска рассредоточения от числа ресурсов объясняется тем, что в задаче используются только данные из одного класса посредника, и не требуется выполнение операции соединения. Зависимости времени поиска рассредоточения от числа операций в обеих задачах объясняются характером перебора в алгоритмах. В случае полного перебора мы перебираем 3^n вариантов, в случае направленного перебора – $3*n$. В случае реальной задачи минимальное рассредоточения не всегда одно и то же. Это связано с тем, что с течением времени может по-разному меняться загрузка каналов передачи данных.

5.4. Выводы по главе

В главе описана программная реализация системы построения рассредоточений. Также в главе представлен пример применения системы построения рассредоточений для реальной научной задачи. Пример демонстрирует, что значительная часть построения эффективного рассредоточения выполняется автоматически (применением экспертных правил и алгоритмом направленного перебора). Вторая часть главы посвящена тестированию основных свойств алгоритмов построения эффективного рассредоточения. В качестве тестовых примеров были выбраны две задачи, одна синтетическая, определенная специально для тестов, вторая реальная задача, описанная в разделе 1.5. Тесты показали эффективность применения подхода по построению рассредоточения. Результаты представлены в сводной таблице. При полном переборе время выполнения удалось улучшить в 4.3 и 9.3 раз для синтетической и реальной задачи соответственно. При направленном переборе время удалось улучшить в 3.7 и в 6 раз. При этом применение экспертных правил показало отличный результат в синтетической задаче, такой же, как направленным перебором, но время построения разительно меньше, т.к. не проводится симуляций. В случае реальной задачи экспертные правила дали не большое улучшение. Производительность алгоритмов на тестах совпала с ожидаемыми оценками. Устойчивость минимального рассредоточения в синтетической задаче, и ее отсутствие в реальной задаче, объясняется тем, что в реальной задаче невозможно учесть всех факторов, влияющих на производительность. Таким образом, учитывая что полный перебор занимает существенное время, лучший результат можно достичь путем совместного использования алгоритма, основанного на экспертных правилах, или алгоритма направленного перебора, или их совместного использования. Это и продемонстрировано на примере в разделе 5.2.

Заключение

В диссертационной работе исследовалась проблема построения эффективного рассредоточения для алгоритма решения задач в рамках среды предметных посредников.

В диссертационной работе получены следующие основные результаты:

- разработаны методы и средства представления рассредоточений и манипулирования ими;
- разработаны методы и средства оценки эффективности рассредоточений;
- разработан подход к построению эффективного рассредоточения;
- разработаны и реализованы алгоритмы построения эффективного рассредоточения – алгоритм полного перебора, алгоритм направленного перебора, алгоритм, основанный на экспертных правилах;
- разработан подход к сопряжению систем программирования с декларативным языком предметных посредников, адекватный задаче рассредоточения и решающий проблемы несоответствия импеданса;
- реализованы программные инструментарии поддержки разработанных в работе методов и подходов;
- разработана архитектура программируемых адаптеров, обеспечивающая эффективное выполнение рассредоточения над ресурсами, а также методы их конструирования;
- разработаны адаптеры для конкретных классов информационных ресурсов;
- проведено тестирование средств построения эффективного рассредоточения, демонстрирующее эффективность разработанных подходов.

Разработанные алгоритмы могут использоваться и в других задачах. В частности, различные адаптеры использовались для сопряжения среды

предметных посредников с системой АстроГрид, в рамках проекта по созданию семантического ГРИДа [102-104]. Результаты, полученные при сопряжении среды предметных посредников с языками программирования, могут использоваться в любых системах доступа к базам данных из ЯП. Разработанные принципы и характеристики могут рассматриваться как требования к подобным системам. Подход построения рассредоточения приложений может использоваться при проектировании решения научных задач в различных инфраструктурах (ГРИДах, облачных инфраструктурах, виртуальных организациях и посредниках).

Литература

1. Брюхов Д.О. Вовченко А.Е. Захаров В.Н. Желенкова О.П. Калиниченко Л.А. Мартынов Д.О. Скворцов Н.А. Ступников С.А. Архитектура промежуточного слоя предметных посредников для решения задач над множеством интегрируемых неоднородных распределенных информационных ресурсов в гибридной грид-инфраструктуре виртуальных обсерваторий. // Информатика и ее применения. – 2008.—Т. 2, вып. 1. – С. 2 – 34
2. Захаров В.Н., Калиниченко Л.А., Соколов И.А., Ступников С.А.. Конструирование Канонических Информационных Моделей для Интегрированных Информационных Систем. // Информатика и ее Применения, 2007. т.1, вып.2, с.15-39.
3. Рябухин О.В. Брюхов Д.О. Калиниченко Л.А. Формирование выражений взглядов в задаче регистрации ресурсов в предметных посредниках. // RCDL'2009, Петрозаводск, Россия, 2009.
4. Alon Y. Halevy. Answering Queries Using Views: A Survey. // VLDB Journal, 10(4), 2001.
5. Jeffrey D. Ullman. Information Integration Using Logical Views. // In Proc. of the 6th Int. Conf. on Database Theory (ICDT'97), 1997.
6. P.J. McBrien, A. Poulouvasilis. Data Integration by Bi-Directional Schema Transformation Rules. // In Proceedings of ICDE03, IEEE, Pages 227-238, 2003.
7. Briukhov D.O., Kalinichenko L.A., Martynov D.O. Source Registration and Query Rewriting Applying LAV/GLAV Techniques in a Typed Subject Mediator. // Proc. of the Ninth Russian Conference on Digital Libraries RCDL'2007.

8. Kalinichenko L. A., Stupnikov S. A., Martynov D. O. SYNTHESIS: A language for canonical information modeling and mediator definition for problem solving in heterogeneous information resource environments. — // M.: IPI RAS, 2007. 171 p.
9. Briukhov D.O., Kalinichenko L.A., Martynov D.O., Skvortsov N.A., Stupnikov S.A. Mediation Framework for Enterprise Information System Infrastructures. // Proc. of the 9th International Conference on Enterprise Information Systems ICEIS 2007. -- Funchal, 2007. -- Volume Databases and Information Systems Integration. -- P. 246--251.
10. Ioana Manolescu, Daniela Florescu, Donald Kossmann. Answering XML Queries over Heterogeneous Data Sources. // Proc. of the Int'l. Conf. on Very Large Databases (VLDB) 2001, Roma, Italy.
11. Ioana Manolescu, Daniela Florescu, Donald Kossmann, Florian Xhumari, Dan Olteanu. Agora: Living with XML and Relational. // Proc. of the Int'l Conf. on Very Large Databases (VLDB) 2000, Cairo, Egypt.
12. Michael R. Genesereth, Arthur M. Keller, Oliver Duschka. Infomaster: An Information Integration System. // In proceedings of 1997 ACM SIGMOD Conference, May 1997.
13. Oliver M. Duschka, Michael R. Genesereth. Infomaster - An Information Integration Tool. // In proceedings of the International Workshop "Intelligent Information Integration" during the 21st German Annual Conference on Artificial Intelligence, KI-97. Freiburg, Germany, September 1997.
14. Patrick Ziegler. Evaluation of SIRUP with the SIRUP Classification of Data Integration Conflicts. // Technical Report ifi-2007.07, Department of Informatics, University of Zurich, 2007.

15. Patrick Ziegler, Christoph Sturm, Klaus R. Dittrich. Unified Querying of Ontology Languages with the SIRUP Ontology Query API. // In 11. GI-Fachtagung für Datenbanksysteme in Business, Technologie und Web (BTW 2005), volume P-65 of Lecture Notes in Informatics, pages 325-344, Karlsruhe, Germany, March 2-4, 2005.
16. Patrick Ziegler, Klaus R. Dittrich. User-Specific Semantic Integration of Heterogeneous Data: The SIRUP Approach. // In Mokrane Bouzeghoub, Carole Goble, Vipul Kashyap, and Stefano Spaccapietra, editors, First International IFIP Conference on Semantics of a Networked World (ICSNW 2004). volume 3226 of Lecture Notes in Computer Science, pages 44-64, Paris, France, June 17-19, 2004. Springer.
17. Chantal Reynaud, Gloria Giraldo. An application of the mediator approach to services over the Web. // Concurrent Engineering, 2003.
18. Chantal Reynaud. Building scalable mediator systems. // Topical Day in Semantic Integration of Heterogeneous Data, IFIP World Computer Congress, 2004.
19. Marie-Christine Rousset, Chantal Reynaud. Pícel and Xyleme: two illustrative information integration agents. // Book chapter in Intelligent Information Agents Research and Development in Europe, Springer-Verlag, 2003.
20. Alon Levy, Anand Rajaraman, Joann Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. // Proceedings of the Twenty-second International Conference on VLDB 1996.
21. Yannis Papakonstantinou, Hector Garcia-Molina, Jeffrey Ullman. MedMaker: A Mediation System Based on Declarative Specifications. // Proceedings of the 12th International Conference on Data Engineering, 1995.
22. D. Beneventano, S. Bergamaschi. The MOMIS Methodology for Integrating Heterogeneous Data Sources. // IFIP World Computer Congress. Toulouse France, 22-27 August 2004.

- 23.D. Beneventano, S. Bergamaschi, F. Guerra, M. Vincini. Building a Tourism Information Provider with the MOMIS System. // *Information Technology & Tourism Journal*, 2005.
- 24.S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, J. Widom. The TSIMMIS Project: Integration of Heterogeneous Information Sources. // In *Proceedings of IPSJ Conference*, pp. 7-18, Tokyo, Japan, October 1994.
- 25.H. Garcia-Molina , Y. Papakonstantinou , D. Quass , A. Rajaraman , Y. Sagiv , J. Ullman , V. Vassalos , J. Widom. The TSIMMIS approach to mediation: Data models and Languages. // In *Journal of Intelligent Information Systems*, 1997.
- 26.M. Boyd, P.J. McBrien. Comparing and Transforming Between Data Models via an Intermediate Hypergraph Data Model. // *Journal on Data Semantics IV*, Pages 69-109, Springer-Verlag, 2005.
- 27.M. Boyd, S. Kittivoravithkul, C. Lazanitis, P.J. McBrien, N. Rizopoulos. AutoMed: A BAV Data Integration System for Heterogeneous Data Sources. // In *Proceedings of CAiSE04*, Springer Verlag LNCS Vol 3084, Pages 82-97, 2004.
- 28.L. Zamboulis, A. Poulouvasilis. Using AutoMed for XML Data Transformation and Integration. // In *Proceedings of DIWeb'04, CAiSE Workshop Proceedings Volume 3*, Pages 58-69.
- 29.L. Zamboulis, N. Martin, A. Poulouvasilis. A Uniform Approach to Workflow and Data Integration. // In *Proceedings of U.K. e-Science All Hands Conference*, September, 2007.
- 30.Patrick Ziegler. User-Specific Semantic Integration of Heterogeneous Data: What Remains to be Done? // *Technical Report ifi-2004.01*, Department of Informatics, University of Zurich, 2004.
- 31.Patrick Ziegler. Data Integration Projects World-Wide. <http://hi.baidu.com/wwcs/blog/item/b158563d2aae29ef3d6d97b9.html>

32. Patrick Ziegler, Klaus R. Dittrich. Data Integration — Problems, Approaches, and Perspectives. // In John Krogstie, Andreas L. Opdahl, and Sjaak Brinkkemper, editors, Conceptual Modelling in Information Systems Engineering, pages 39–58. Springer, Berlin, 2007.
33. Abrial J.-R. B-technology: Technical overview. — B-Core (UK) Ltd., 1993.
34. Abrial J.-R. The B-book: Assigning programs to meanings. — Cambridge: Cambridge University Press, 1996.
35. The B-toolkit online documentation.
<http://www.bcore.com/ONLINEDOC/BToolkit.html>.
36. Briukhov D. O., Kalinichenko L. A. Component-based information systems development tool supporting the SYNTHESIS design method // 2nd East-European Conference “Advances in Databases and Information Systems” Proceedings.—Berlin-Heidelberg:Springer-Verlag, 1998. P. 305–327.
37. Briukhov D.O., Kalinichenko L. A., Skvortsov N. A. Information sources registration at a subject mediator as compositional development // East-European Conference “Advances in Databases and Information Systems” Proceedings. Lithuania, Vilnius, Springer, LNCS No. 2151, 2001.
38. Вовченко А.Е. Автоматизация создания адаптеров для сред неоднородных распределенных информационных источников. // Сборник тезисов XIV Международной научной конференции студентов, аспирантов и молодых ученых «Ломоносов». МГУ, 2007, стр. 14.
39. Briukhov D., Kalinichenko L., Martynov D., Skvortsov N., Stupnikov S., Vovchenko A., Zakharov V., Zhelenkova O. Application driven mediation middleware of the Russian virtual observatory for scientific problem solving over multiple heterogeneous distributed information resources. // Scientific Information for Society – from Today to the Future: Proc. of the 21st CODATA Conference. -- 2009. -- P. 80-85.

40. Вовченко А.Е., Калиниченко Л.А., Малков О.Ю., Мамардашвили Н.А., Патракова М.Е. Встраивание средств Data Mining в инфраструктуру виртуальной обсерватории. // Труды 9ой Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» - RCDL'2007, Переславль-Залесский, Россия, 2007.
41. Kalinichenko L.A., Stupnikov S.A., Vovchenko A.E., Zakharov V.N., Zhelenkova O.P. Russian Virtual Observatory Community Centre for Scientific Problems Solving over Multiple Distributed Information Sources. // Proc. of the Eighth Russian Conference on Digital Libraries RCDL'2006, Suzdal. -- Yaroslavl: P. G. Demidov Yaroslavl State University, 2006. -- P. 120--129.
42. Вовченко А.Е., Вольнова А.А., Денисенко Д.В., Калиниченко Л.А., Куприянов В.В., Позаненко А.С., Скворцов Н.А., Ступников С.А. Применение средств виртуальной обсерватории для выбора вторичных стандартов поля при фотометрии оптического послесвечения гамма-всплесков. // Труды Всероссийской астрономической конференции ВАК-2010 «От эпохи Галилея до наших дней». – САО РАН: Нижний Архыз. – 2010.
43. Вовченко А.Е., Захаров В.Н., Калиниченко Л.А., Ковалёв Д.Ю., Рябухин О.В., Скворцов Н.А., Ступников С.А. Многоуровневые спецификации в концептуальном и онтологическом моделировании. // Труды 13-й Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» RCDL'2011. – Воронеж: Воронежский государственный университет, 2011. – С. 35-43.

44. Вовченко А.Е., Захаров В.Н., Калиниченко Л.А., Ковалёв Д.Ю., Рябухин О.В., Скворцов Н.А., Ступников С.А. От спецификаций требований к концептуальной схеме. // Труды 12ой Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» - RCDL'2010, Казань, Россия, 2010.
45. Вовченко А.Е. Рассредоточение для реализации приложений в распределенной среде предметных посредников. // Труды 13-й Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» RCDL-2011. – Воронеж: Воронежский государственный университет, 2011. – С. 285-292.
46. Вовченко А.Е. Рассредоточение для реализации алгоритма решения задач над неоднородными распределенными ресурсами. // Вестник Воронежского государственного университета, серия Системный анализ и информационные технологии. - №2. - Воронеж: Воронежский Государственный Университет, 2011, с.117-122.
47. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. // Addison-Wesley, Reading (1995)
48. Bernt Kullbach, Andreas Winter, Peter Dahm, Jürgen Ebert. Program Comprehension in Multi-Language Systems. // Proceeding WCRE '98 Proceedings of the Working Conference on Reverse Engineering (WCRE'98), 1998.
49. Rajiv Dewan, Abraham Seidmann, Zhiping Walter. Workflow Optimization through Task Redesign in Business Information Processes. // Proceeding HICSS '98 Proceedings of the Thirty-First Annual Hawaii International Conference on System Sciences - Volume 1, IEEE Computer Society Washington, DC, USA, 1998.

50. Вовченко А.Е., Крупа А.В. Планирование запросов над множеством неоднородных распределенных информационных ресурсов в архитектуре средств поддержки предметных посредников. // Труды 11ой Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» - RCDL'2009, Петрозаводск, Россия, 2009. с. 335-342.
51. Surajit Chaudhuri. An Overview of Query Optimization in Relational Systems. // Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems. 1998.
52. José Luis Ambite, Craig A. Knoblock. Flexible and scalable cost-based query planning in mediators: a transformational approach. // Journal: Artificial Intelligence - Special issue on Intelligent internet systems. Volume 118 Issue 1i2, April 2000.
53. J.L. Ambite, C.A. Knoblock, Planning by rewriting: Efficiently generating high-quality plans. // in: Proc. AAAI-97, Providence, RI, 1997.
54. Ramana Yerneni, Chen Li, Jeffrey D. Ullman, Hector Garcia-Molina. Optimizing Large Join Queries in Mediation Systems. // Proceeding ICDT '99 Proceedings of the 7th International Conference on Database Theory, 1999.
55. S. Adali, K. S. Candan, Y. Papakonstantinou, V. S. Subrahmanian. Query caching and optimization in distributed mediator systems. // Proceedings of the 1996 ACM SIGMOD international conference on Management of data. New York, NY, USA, 1996.
56. J. Ebert, A. Winter, P. Dahm, A. Franzke, and R. Süttenbach. Graph Based Modeling and Implementation with EER/GRAL. // In B. Thalheim, editor, 15th International Conference on Conceptual Modeling (ER'96), Proceedings, number 1157 in LNCS, pages 163–178. Springer, Berlin, 1996.
57. J. Sowa. Conceptual Structures. Information, Processing in Mind and Machine. The Systems Programming Series. // Addison-Wesley, Reading, 1984.

58. Vovchenko A.E. Binding of Programming Languages with Subject Mediators for Scientific Problems Solving. // Advances in Databases and Information Systems: Proc. II of the 15th East-European Conference. - Vienna: Austrian Computer Society, 2011. -- P. 272-279.
59. Cattell R.G.G., Barry D.K. et al. The object data Standard: ODMG 3.0. – Morgan Kaufmann Publishers, San Francisco, California.
60. OCCI User Guide. – http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28390/toc.htm
61. White S., Hapner M. JDBC 2.1 API, November 30, 1999. – http://www.informatik.uni-frankfurt.de/java/JDK/JDK_doku/jdk1.3.1/docs/guide/jdbc/spec2/jdbc2.1.frame.html
62. Melton J. (ISO-ANSI Working Draft) Object Language Bindings (SQL/OLB), American National Standard, Information technology – Database languages – SQL – Part 10: Object Language Bindings (SQL/OLB), August 2003.
63. Eisenberg A., Melton J. SQLJ – Part 1: SQL Routines using the Java TM Programming Language// ACM SIGMOD Record. – December 1999. – V. 28, No4.
64. JDO documentation. – <http://java.sun.com/jdo/>.
65. LINQ to SQL User Guide. – <http://msdn.microsoft.com/ru-ru/library/bb386976.aspx>.
66. Subieta K. Impedance mismatch. – http://www.ipipan.waw.pl/~subieta/SBA_SBQL/Topics/ImpedanceMismatch.html
67. Фаулер М., Бек К., Брант Д., Робертс Д., Алдаик У. Рефакторинг: улучшение существующего кода (2000). — Спб: Символ-Плюс, 2009
68. Kalinichenko L.A. Methods and tools for equivalent data model mapping construction// Proc. of the Int. Conf. on Extending Database Technology EDBT'90. LNCS 416. – Berlin – Heidelberg: Springer-Verlag, 1990. – P. 92-119.

69. Kalinichenko L.A., Stupnikov S.A. Constructing of mappings of heterogeneous information models into the canonical models of integrated information systems// Advances in Databases and Information Systems (ADBIS): Proc. of the 12th East-European Conf. – Pori: Tampere University of Technology, 2008. – P. 106-122.
70. Millstein T., Halevy A., Friedman M. Query containment for data integration systems// J. of Computer and System Sciences. – 2003. – V. 66, Issue 1. – P. 20-39.
71. Sipelstein, J.M., Bletloch, G.E. Collection-oriented languages. // In Proceedings of the IEEE, Volume: 79, Issue:4, Apr 1991, p: 504 – 523.
72. David S. Frankel. Model Driven Architecture: Applying MDA to Enterprise Computing. John Wiley & Sons
73. ATL Project, subproject of Eclipse. <http://www.eclipse.org/atl/>
74. Dave Steinberg, Frank Budinsky, Marcelo Paternostro, Ed Merks. EMF: Eclipse Modeling Framework, 2nd Edition. Dec 16, 2008 by Addison-Wesley Professional.
75. Hibernate Project. Relational Persistence for Java and .NET. <http://hibernate.org/>
76. William R. Cook, Ali H. Ibrahim. Integrating Programming Languages & Databases: What's the Problem? www.cs.utexas.edu/~wcook/Drafts/2005/PLDBProblem.pdf
77. Jian Chen , Qiming Huang. Eliminating the Impedance Mismatch Between Relational Systems and Object-Oriented Programming Languages. // in Proce. the 6th International Hong Kong Database Workshop, 1995.
78. Joseph (Yossi) Gil, Keren Lenz. Eliminating Impedance Mismatch in C++. // VLDB, 2007.
79. Michał Lentner, Kazimierz Subieta. ODRA: A Next Generation Object-Oriented Environment for Rapid Database Application Development. // ADBIS 2007, p. 130-140.

80. Markus Kirchberg. Integration of Database Programming and Query Languages for Distributed Object Bases. // PhD thesis for the degree of Doctor of Philosophy in Information Systems at Massey University, 2007.
81. Котляров Ю.В., Подколодный Н.Л. Подключение баз молекулярно-генетических данных к посреднику среды создания интегрированных электронных библиотек. // Вторая Всероссийская конференция “Электронные библиотеки”. 26-28 сентября 2000.
82. Котляров Ю.В. Интеграция баз молекулярно-генетических данных в предметном посреднике. // ВМК, МГУ. 2000. Магистерская диссертация.
83. Осипов М.А., Мачульский О.Л., Калиниченко Л.А. Отображение модели данных XML в объектную модель. // Первая Всероссийская конференция “Электронные библиотеки”. 1999.
84. Осипов М.А., Калиниченко Л.А. Интеграция XML-коллекций данных в посреднике неоднородных коллекций электронных библиотек. // Вторая Всероссийская конференция “Электронные библиотеки”. 26-28 сентября 2000.
85. Осипов М.А. Подход к полуавтоматической генерации адаптеров в посреднике неоднородных коллекций электронных библиотек. // ВМК, МГУ. 2001. Дипломная работа.
86. Yannis Papakonstantinou, Ashish Gupta, Hector Garcia-Molina, Jeffrey Ullman. A Query Translation Scheme for Rapid Implementation of Wrappers. // Deductive and Object-Oriented Databases (DOOD). 1995.
87. Yannis Papakonstantinou, Ashish Gupta, Laura Haas. Capabilities-Based Query Rewriting in Mediator Systems. // Parallel and Distributed Information Systems (PDIS). 1996. Selected in the "Best of PDIS".
88. Vasilis Vassalos, Yannis Papakonstantinou. Describing and Using Query Capabilities of Heterogeneous Sources. // Proceeded on VeryLarge DataBases (VLDB). 1997.

89. Meta Object Facility (MOF) 2.0 Core Specification. <http://www.omg.org/cgi-bin/apps/doc?ptc/03-10-04.pdf>, 2003.
90. OMG/OCL Object Constraint Language (OCL) 2.0. OMG Final Adopted Specification. <http://www.omg.org/spec/OCL/2.0/>, 2003.
91. Budinsky, F., Steinberg, D., Ellersick, R., Grose, T. Eclipse Modeling Framework, Chapter 5 "Ecore Modeling Concepts". - Addison Wesley Professional, 2004.
92. Astrogrid Project. Virtual Observatory Software for Astronomers. <http://www.astrogrid.org/>
93. Astrogrid DSA Catalog Overview. <http://www.astrogrid.org/maven/docs/HEAD/pal/index.html>
94. The VizieR Catalogue Service. <http://vizier.u-strasbg.fr/viz-bin/VizieR>
95. The Sloan Digital Sky Survey. <http://www.sdss.org/>
96. Вовченко А.Е., Калиниченко Л.А., Костюков М.Ю. Методы и средства доступа к потоковым данным из предметных посредников. // Труды 12ой Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» - RCDL'2010, Казань, Россия, 2010.
97. Jean-Robert Gruser, Louiqa Raschid, Mar'ia Esther Vidal, Laura Bright. Wrapper Generation for Web Accessible Data Sources. // Proceeding COOPIS '98 Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems.
98. Alberto Pan, Juan Raposo, Manuel Álvarez, Justo Hidalgo and Ángel Viña. Semi-Automatic Wrapper Generation for Commercial Web Sources. Proceedings of the IFIP TC8 // WG8.1 Working Conference on Engineering Information Systems in the Internet Context, 2002.
99. Kai-Uwe Sattler, Michael H'oding. Adapter Generation for Extracting and Querying Data from Web Sources. // In Proc. 2nd ACM SIGMOD Workshop WebDB'99, 1999.

100. Stefan Kuhlins and Ross Tredwell. Toolkits for Generating Wrappers. // Proceeding NODe '02 Revised Papers from the International Conference NetObjectDays on Objects, Components, Architectures, Services, and Applications for a Networked World, 2003.
101. Philippe Thiran , Jean-Luc Hainaut , Geert-Jan Houben. Database Wrappers Development: Towards Automatic Generation. // Proceedings of the Ninth European Conference on Software Maintenance and Reengineering, p.207-216, March 21-23, 2005.
102. Вовченко А.Е., Калиниченко Л.А., Ступников С.А. Семантический грид, основанный на концепции предметных посредников. // Труды четвертой международной конференция "Распределённые вычисления и Грид-технологии в науке и образовании" Grid2010, Дубна, ОИЯИ, 2010. – с. 309-318.
103. Вовченко А.Е., Захаров В.Н., Калиниченко Л.А., Ступников С.А., Скворцов Н.А. Предметные посредники в гибридной грид-инфраструктуре виртуальных обсерваторий для решения задач над неоднородными распределенными информационными ресурсами. // Труды IV-ой Международной научно-практической конференции «Современные информационные технологии и ИТ-образование», Москва, МГУ имени М.В. Ломоносова, 2009. - с. 762-770.
104. Брюхов Д.О., Вовченко А.Е., Желенкова О.П., Захаров В.Н., Калиниченко Л.А., Мартынов Д.О., Скворцов Н.А., Ступников С.А. Грид-инфраструктура предметных посредников, движимых приложениями, для решения задач над множеством неоднородных распределенных информационных ресурсов. // Труды третьей международной конференция "Распределённые вычисления и Грид-технологии в науке и образовании" Grid2008, Дубна, ОИЯИ, 2008.

Приложение А Грамматика декларативного языка правил предметных посредников (язык Syfs)

Синтаксис языка Syfs

```

< logic program > ::= {{< program >}}
< program > ::= < rule list >
< rule list > ::= < rule > [, < rule >]...
< rule > ::= < head > : - < body >.
< head > ::= < atom >
< body > ::= < formula >
< atom > ::= < predicate name > (< term list >)
< term > ::= < typed variable > | < atom > | < arithmetic expression > | < value >
< typed variable > ::= < variable > [ / (< type expression > ) ]
< type expression > ::= < compositional term > |
< type expression > < compositional operation > < type term >
< compositional operation > ::= " | " | &
< type term > ::= < type variable identifier > | < type designator > | (< type expression > ) | <
function designator >
< constant > ::= < nonconstant value > [ / < type designator > ] | < collection constant >
< type designator > ::= < type name > | < attribute name > | < type name > [ "." < type name > ] ".inst" | <
reduct > | < simple built in type >
< simple built in type > ::= real | string
< built-in type > ::= < set type > | < bag type > | < simple built in type >
< set type > ::= { set; type of element : < type > }
< bag type > ::= { bag; type of element : < type > }
< type name > ::= < type identifier > | < class name > [:nonobject]
< type identifier > ::= [ < module identifier > : ] < identifier >
< class name > ::= < class identifier > | < type variable identifier >
< class identifier > ::= [ < module identifier > : ] < identifier >
< type variable identifier > ::= [ < module identifier > : ] < identifier >
< attribute name > ::= < attribute identifier > | < attribute name > "." < attribute identifier >
< attribute identifier > ::= < identifier >
< reduct > ::= < type name > ["reduct element list > "]
< reduct element list > ::= < reduct element > [ ; < reduct element > ]...
< reduct element > ::= < attribute identifier > [ / < type expression > ] [ : < path > ]
< path > ::= < attribute identifier > [ . < attribute identifier > ]
< variable > ::= < identifier >
< formula > ::= < predicate > | < formula > & < formula > |
group by (< grouping list > , " < " < atom - collection > " > " , < formula > )
< predicate > ::= < atom > | < condition >
< condition > ::= < function designator > | < arithmetic expression > < sigma > < arithmetic expression >
| < collection value > < set predicate sign > < collection value >
< collection value > ::= < typed variable > | < function designator > | < collection constant >
< set predicate sign > ::= < | < = > | > >
< sigma > ::= " = " | " < " | " < = " | " > = " | " > > "
< arithmetic expression > ::= < additive > | < additive operation > < additive > |
< arithmetic expression > < additive operation > < additive >
< additive operation > ::= + | -
< additive > ::= < multiplier > | < additive > < multiplicative operation > < multiplier >
< multiplicative operation > ::= * | /
< multiplier > ::= < variable > | < number > | (< arithmetic expression > )
< function designator > ::= < function identifier > [ (< actual parameter list > ) ]
< actual parameter list > ::= < term list >
< predicate name > ::= < function name > | < collection name >
< function name > ::= < function identifier > | < built in type operation name >
< built in type operation name > ::= union | intersect | differ | is_empty | is_in | is_equal

< frame > ::= { [ < frame identifier > ; ] [ < frame body > ] }
< frame body > ::= [ < slot > : [ < value > ] ; ]...
< frame identifier > ::= < identifier >

```

```

< slot >::=< slot identifier >
< slot identifier >::=< identifier >|< variable >
< value >::= none |< constant >|< frame >
< set function >::= min | max | count | total | average
< collection constant >::= { < term >, <term> ... } | {“ [“< term >, <term> ... “]” ... }

```

Antlr грамматика языка Syfs

```

header {
package queryprogram.parser;
}

class QueryParser extends Parser("QueryParserBase");

options {
    k = 3;
    buildAST = true;
    defaultErrorHandler = false;
}

tokens {
    ID<AST=queryprogram.parser.TokenAST>;
    PLUS<AST=queryprogram.parser.TokenAST>;
    MINUS<AST=queryprogram.parser.TokenAST>;
    TIMES<AST=queryprogram.parser.TokenAST>;
    SLASH<AST=queryprogram.parser.TokenAST>;
    "nonobject"<AST=queryprogram.parser.TokenAST>;

    LOGICPROGRAM; RULE; F2FIR; ALLATTRS; BODY; PREDICATE; CONDITION; TERMLIST; TERM;
    NOTCONSTPATH; NUMBERCONST; BOOLEANCONST; APREDICATE; METHOD; TYPEMEET; TYPEJOIN;
    NONOBJECT; TYPETERM; BUILTINTYPE; CLASSINST; TYPEID; REDUCT; REDUCTELEM; AEXPRESS; ABINOP;
AUNOP;

    ID2; PLAN; ATOM; SELECT; PROJECT; MOVE; JOIN; UNION; CJOIN; APPEND; RENAME;
}

logic_program : LBRACE! LBRACE! rule_or_f2fir ( COMMA! rule_or_f2fir )* RBRACE! RBRACE!
    { #logic_program = #[LOGICPROGRAM], #logic_program); }
    ;

rule_or_f2fir
    : (ID SLASH)=> f2fir
    | rule
    ;

rule : head COLON! MINUS! body { #rule = #[RULE], #rule); } ;

f2fir: all_attrs predicate MINUS! GT! predicate { #f2fir = #[F2FIR], #f2fir); } ;

all_attrs: all_attr (COMMA! all_attr)* { #all_attrs = #[ALLATTRS], #all_attrs); } ;

all_attr: ID SLASH! te: type_expression! { #all_attr = #[all_attr, te); } ;

head : predicate ;

body : c_predicate ( CONJUNCTION! c_predicate )* { #body = #[BODY], #body); } ;

predicate { TokenAST ast = new TokenAST(PREDICATE, LT(1)); }
    :
    ID ( DOT! ID )? LPAREN! term_list RPAREN!
    { #predicate = #(ast, #predicate); }
    ;

term_list : term ( COMMA! term )* { #term_list = #[TERMLIST], #term_list); } ;

```

```

opt_term_list : ( term ( COMMA! term )* )? { #opt_term_list = #[TERMLIST], #opt_term_list); } ;

term { TokenAST ast = new TokenAST(TERM, LT(1)); }
    :
      ( ( constant )=> constant
      | not_const_path
      | LPAREN! a_expr RPAREN! )
      ( SLASH! type_expression )?
      { #term = #(ast, #term); }
    ;

constant
    : number_const
      | { LT(1).getText().equals("true") || LT(1).getText().equals("false") }? id: ID {
#id.setType(BOOLEANCONST); }
      | string_const
//   | t = time_const
    ;

// TODO: add support of exponent
number_const! { String s = ""; } :
    ( PLUS | MINUS { s = "-"; } )? i1: INTEGER { s += i1.getText(); }
    ( DOT i2: INTEGER { s += "." + i2.getText(); } )?
    { #number_const = #[NUMBERCONST, s]; }
    ;

string_const : PRIMESTRING ;

not_const_path : ( ID | method ) ( DOT! ( ID | method ) )* { #not_const_path = #[NOTCONSTPATH],
#not_const_path); } ;

method : ID LPAREN! opt_term_list RPAREN! { #method = #[METHOD], #method); } ;

condition : c_predicate ( CONJUNCTION! c_predicate )* { #condition = #[CONDITION], #condition);
} ;

c_predicate :
    ( a_predicate )=> a_predicate
    | predicate
    ;

a_predicate { TokenAST ast = new TokenAST(APREDICATE, LT(1)); }
    : a_expr_as_term
      ( EQUALS! { ast.setText("="); }
      | LT! { ast.setText("<"); }
      | GT! { ast.setText(">"); }
      | LTE! { ast.setText("<="); }
      | GTE! { ast.setText(">="); } )
      a_expr_as_term
      { #a_predicate = #(ast, #a_predicate); }
    ;

a_expr_as_term { TokenAST ast = new TokenAST(TERM, LT(1)); }
    : a_expr { #a_expr_as_term = #(ast, #a_expr_as_term); }
    ;

a_expr : a_expr1 ( ( PLUS^ | MINUS^ ) { #a_expr.setType(ABINOP); } a_expr1 )* ;

a_expr1 : a_expr2 ( ( TIMES^ | SLASH^ ) { #a_expr1.setType(ABINOP); } a_expr2 )* ;

a_expr2
    : ( PLUS | MINUS )=> op: a_un_op! a_term { #a_expr2 = #(#op, #a_expr2); }
      | a_term
    ;

```

```

a_un_op : ( PLUS | MINUS ) { #a_un_op.setType(AUNOP); } ;

a_term { TokenAST ast = new TokenAST(TERM, LT(1)); }
      :
      ( ( constant )=> constant
      | not_const_path
      | LPAREN! a_expr RPAREN! )
      { #a_term = #(ast, #a_term); }
      ;

type_expression : type_meet ;

type_meet // type_meet
      { TokenAST ast = new TokenAST(TYPEMEET, LT(1)); }
      : type_join ( CONJUNCTION! type_join )*
      { #type_meet = #(ast, #type_meet); }
      ;

type_join
      { TokenAST ast = new TokenAST(TYPEJOIN, LT(1)); }
      : type_term ( BAR! type_term )*
      { #type_join = #(ast, #type_join); }
      ;

type_term
      { TokenAST ast = new TokenAST(TYPETERM, LT(1)); }
      :
      (
      ( LPAREN! type_expression RPAREN!
      | ( builtin_type )=> builtin_type
      | ( class_inst )=> class_inst
      | type_id ) ( reduct )?
      | reduct
      ) ( DOT! no: "nonobject" { #no.setType(NONOBJECT); } )?
      { #type_term = #(ast, #type_term); }
      ;

type_id : ( ( ID DOT! ID )=> ID DOT! ID | ID ) { #type_id = #([TYPEID], #type_id); } ;

class_inst : ID ( DOT! ID )? DOT! "inst"! { #class_inst = #([CLASSINST], #class_inst); } ;

reduct
      { TokenAST ast = new TokenAST(REDUCT, LT(1)); }
      : LBRACKET! reduct_element ( COMMA! reduct_element )* RBRACKET!
      { #reduct = #(ast, #reduct); }
      ;

reduct_element : ID ( SLASH! type_expression )? ( COLON! ID ( DOT! ID )* )? { #reduct_element =
#([REDUCTELEM], #reduct_element); } ;

builtin_type : ( set_type | other_builtin_type ) { #builtin_type = #([BUILTINTYPE],
#builtin_type); } ;

set_type
      : { LT(2).getText().equals("set") }? LBRACE! ID! SEMI!
      { LT(1).getText().equals("type_of_element") }? ID! COLON! type_expression SEMI! RBRACE!
      ;

other_builtin_type
      : { isBuiltinTypeID(LT(1).getText()) }? ID
      ;

class QueryLexer extends Lexer;

```

```

options {
    testLiterals = false;
    k = 3;
}

WS
: ( ' ' | '\t' | '\f' | '\n' | '\r' { newline(); } )
{ $setType(Token.SKIP); }
;

COMMENT
: "//" ( ~( '\n' | '\r' ) )*
{ $setType(Token.SKIP); }
;

ML_COMMENT
: "/*" ( ( '*' ~( '/' ) => '*' | ('\n') { newline(); } | ('\r') | ~('*' | '\n' | '\r') )* '*'
 '/'
{ $setType(Token.SKIP); }
;

protected DIGIT      : '0'..'9' ;
protected LETTER     : 'a'..'z' | 'A'..'Z' ;
protected PRIME      : '\'' ;
protected UNDERSCORE: '_' ;

PRIMESTRING
: PRIME! ( ~( '"' | '\n' | '\'' ) | '\n' { newline(); } )* PRIME!
;

INTEGER      : (DIGIT)+ ;

DOT          : '.' ;
COLON        : ':' ;
SEMI         : ';' ;
COMMA        : ',' ;
SLASH        : '/' ;

LPAREN       : '(' ;
RPAREN       : ')' ;
LBRACE       : '{' ;
RBRACE       : '}' ;
LBRACKET     : '[' ;
RBRACKET     : ']' ;
LT           : '<' ;
GT           : '>' ;
NOT_EQUALS   : "<>" ;
LTE          : "<=" ;
GTE          : ">=" ;

EQUALS       : '=' ;
PLUS         : '+' ;
MINUS        : '-' ;
TIMES        : '*' ;

CONJUNCTION  : '&' ;
BAR          : '|' ;

ID options { testLiterals = true; }
: ( LETTER | UNDERSCORE ) ( LETTER | UNDERSCORE | DIGIT )*
| "@"! ( ~'\'' )* '\!'
| "@\"! ( ~'\"' )* '\"!'
;

```

Приложение Б Алгебраическая форма языка правил предметных посредников – язык *Asyfs*

Язык *Asyfs* (Algebraic Synthesis Formulae language Subset) является алгебраической версией языка *Syfs*, и предназначен для выражения плана запроса над информационным источником. План определяет способ и порядок выполнения запроса. Планом называется формула языка *Asyfs*. Формула представляет собой алгебраическое выражение, которое образуются из операций над коллекциями. В качестве констант выступает операция *Atom*. В синтаксисе языка используется инфиксная форма записи выражений. Синтаксис языка *Asyfs* в грамматике EBNF приведен ниже. Все операции над коллекциями языка *Asyfs* описываются в последующих пунктах.

Операция *Atom*

Операция *Atom* не имеет операндов, и фактически играет роль констант в алгебраических формулах языка *Asyfs*. В соответствии с синтаксисом языка *Asyfs* операция *Atom* имеет следующий вид:

$$p(x)$$

где p – указатель коллекции, x – идентификатор переменной.

Синтаксически операция *Atom* полностью совпадает с предикатов коллекции в языке *Syfs*, в котором не используется конструкция проекции $/T$. Операция *Atom* возвращает коллекцию, которая совпадает с результирующим множеством формулы из одного предиката коллекции $p(x)$ в языке *Syfs*. Отличие состоит в том, что переменная x является обыкновенным атрибутом, и на этот атрибут не распространяются исключения, который определяются для переменных в языке *Syfs*. Таким образом, переменную (атрибут) x необходимо явно включать в проекцию после операции *Atom*, а также отслеживать, чтобы имена переменных не совпадали, и они не участвовали в соединениях

Результирующее множество RS , которое возвращает операция $Atom$, получается из коллекции p присоединением переменной x . Последнее означает, что меняется тип экземпляров коллекции, к которому добавляется атрибут x . Типом атрибута x является объектный тип экземпляров класса p : $p.inst.object$. Формальное построение результирующего множества RS описывается следующим образом:

$$RS = \{extend(o, x) \mid p(o)\}$$

Функция $extend(o, x)$ возвращает экземпляр, который получается из o добавлением атрибута x , который принимает o в качестве своего значения, т.е. для $o = \langle a_1=v_1, \dots, a_n=v_n \rangle$:

$$extend(o, x) = \langle a_1=v_1, \dots, a_n=v_n, x=o \rangle$$

При этом считается, что функция преобразования ρ_T определяется так, как будто в редукте, соответствующему указателю АТД T , присутствует элемент x : x . Последнее означает, что переменная x входит в результат преобразования, и преобразование не изменяет значения переменной x .

Операция Project

Операция $Project$ является унарной операцией в языке $Asyfs$, которая осуществляет преобразование коллекции в соответствии с АТД (редуктом в частном случае), который является параметром этой операции. В соответствии с синтаксисом языка $Asyfs$ операция $Project$ имеет следующий вид:

$PROJECT\{R\} \text{ plan}$

где R – указатель АТД, $plan$ – план, возвращающий коллекцию, которая является операндом операции $Project$. Коллекция, которую возвращает коллекция, определяется формулой:

$$\rho_R(p)$$

где ρ_R – функция преобразования коллекции.

Операция Join

Операция Join является бинарной операцией в языке Asyfs, которая осуществляет соединение двух коллекций. В соответствии с синтаксисом языка Asyfs операция Join имеет следующий вид:

JOIN $plan_1$ $plan_2$

где $plan_1$ и $plan_2$ – планы, возвращающие коллекции, которые являются операндами операции Join. Операция Join семантически полностью соответствует операции соединения коллекций в языке Syfs. Результирующая коллекция операции Join равна соединению операндов операции Join.

Операция Union

Операция Union является бинарной операцией в языке Asyfs, которая осуществляет объединение двух коллекций как множеств. В соответствии с синтаксисом языка Asyfs операция Union имеет следующий вид:

UNION $plan_1$ $plan_2$

где $plan_1$ и $plan_2$ – планы, возвращающие коллекции, которые являются операндами операции Union. Результирующая коллекция операции Union равна объединению операндов операции как множеств:

$$c_1 \cup c_2$$

где c_1 и c_2 – коллекции, которые возвращают планы $plan_1$ и $plan_2$ соответственно.

Операция Select

Операция Select является унарной операцией в языке Asyfs, с помощью которой осуществляется фильтрация элементов в коллекции согласно логическому условию, которое является параметром операции. В соответствии с синтаксисом языка Asyfs операция Select имеет следующий вид:

SELECT{condition} plan

где $plan$ – план, возвращающий коллекцию, которая является операндом операции Select, $condition$ – логическое условие, которое в языке Asyfs полностью совпадает с условием в языке Syfs. Результирующая коллекция

операции `Select` является подмножеством операнда операции, которое составляют элементы, для которых выполняется условие операции `Select`:

$$\{ x \mid x \in c, \text{condition}(x) \}$$

где c – коллекция, которую возвращает план $plan$.

Операция Append

Операция `Append` является унарной операцией в языке `Asyfs`, которая над элементами коллекции осуществляет вычисление функции/метода, параметра операции `Append`, и присоединение атрибутов с результатами. В соответствии с синтаксисом языка `Asyfs` операция `Append` имеет следующий вид:

`APPEND{predicate} plan`

где $plan$ – план, возвращающий коллекцию, которая является операндом операции `Append`, $predicate$ – предикат-функция, который в языке `Asyfs` полностью совпадает с предикатом-функцией в языке `Syfs`.

Операция Move

Операция `Move` является специальной унарной операцией в языке `Asyfs`, которая всегда является корнем дерева плана. Операция `Move` создает класс, который содержит все элементы коллекции, операнда операции `Move`. Указатель созданного класса определяется параметром операции `Move`. В соответствии с синтаксисом языка `Asyfs` операция `Move` имеет следующий вид:

`MOVE{class} plan`

где $plan$ – план, возвращающий коллекцию, которая является операндом операции `Move`, $class$ – сокращенный указатель класса, который создает операция `Move`. Тип экземпляров созданного класса совпадает с типом элементов коллекции. Созданный класс определяется следующей формулой:

$$\{ o \mid o \in c \}$$

где c – коллекция, которую возвращает план $plan$.

Синтаксис языка Asyfs

Приведенная ниже грамматика языка `Asyfs` наследует правила из грамматики языка `Syfs`, которые определяют нетерминальные символы

(помечаются подчеркнутой линией), которые используются в правилах грамматики языка Asyfs.

Asyfs:

```

<plan>
 ::= <atom>
    | <project>
    | <select>
    | <move>
    | <join>
    | <union>
    | <append>

<atom>
 ::= <class designator> '(' <variable> ')

<project>
 ::= 'PROJECT{' <adt designator> '}' <plan>

<select>
 ::= 'SELECT{' <condition> '}' <plan>

<move>
 ::= 'MOVE{' <identifier> '}' <plan>

<join>
 ::= 'JOIN' <plan> <plan>

<union>
 ::= 'UNION' <plan> <plan>

<append>
 ::= 'APPEND{' <function predicate> '}' <plan>

```

Syfs:

```

<class predicate>
 ::= <class designator> '(' (<variable identifier> [ '/' <adt designator> ] ')'

<module entity designator>
 ::= <identifier> [ '.' <identifier> ]

<class designator>
 ::= <module entity designator>

<type designator>
 ::= <module entity designator>

<adt designator>
 ::= <module entity designator>
    | <reduct>
    | <class designator> '.inst'

<function type designator>
 ::= <module entity designator>

<reduct>
 ::= [ <adt designator> ] '[' <reduct element> { ',' <reduct element> } '['

<reduct element>
 ::= <attribute identifier> [ '/' <adt designator> ] [ ':' <path> ]

```

```

<path>
 ::= <attribute identifier> {',' <attribute identifier>}

<attribute identifier>
 ::= <identifier>

<variable identifier>
 ::= <identifier>

<condition>
 ::= <c-predicate> {'&' <c-predicate>}

<c-predicate>
 ::= <arithmetic predicate> | <module function c-predicate> | <adt c-predicate>

<arithmetic predicate>
 ::= <term> <relational operation> <term>

<relational operation>
 ::= '<'
    | '<='
    | '='
    | '>'
    | '>='

<module function c-predicate>
 ::= <function type designator> '(' (<input arguments> ')'

<adt c-predicate>
 ::= <attribute identifier> '(' (<self term> [',' <input arguments>] ')'

<input arguments>
 ::= <term> {',' <term>}

<term>
 ::= <attribute>
    | <get attribute>
    | <function designator>
    | <arithmetic expression>
    | <variable>
    | <constant>

<self term>
 ::= <attribute>
    | <get attribute>
    | <function designator>
    | <variable>

<variable>
 ::= <variable identifier>

<attribute>
 ::= <attribute identifier>

<get attribute>
 ::= <term> '.' <attribute identifier>

<arithmetic expression>
 ::= '(' <arithmetic expression> ')'
    | <term> {'+' <term>}
    | <term> {'-' <term>}
    | <term> {'*' <term>}
    | <term> {'/' <term>}

<function designator>
 ::= <module function designator>

```

```
| <adt function designator>

<module function designator>
::= <function type designator> '(' <input arguments> ')

<adt function designator>
::= <function type designator> '(' <input arguments> ')

<function predicate>
::= <adt predicate>
  | <module function predicate>

<module function predicate>
::= <function type designator> '(' <input arguments> ',' <output arguments> ')

<adt predicate>
::= <attribute identifier> '(' (<self term> [',' <input arguments>] ')

<output arguments>
::= <output argument> {',' <output argument>}

<output argument>
::= <attribute identifier> [ '/' <adt designator> ]
```

Приложение В Статическое связывание языка СИНТЕЗ и языка программирования Java

Назовем отображение M

Synthesis	Java	Комментарий
Простые типы (отображение один-в-один)		
integer	long	Отображение типа
real	double	Отображение типа
string	String	Отображение типа
Boolean	boolean	Отображение типа
Сложные типы		
{ScienceData; in: type; flux: {set; type_of_element: Flux;}}	<pre>public class ScienceData { Set<Flux> flux; }</pre>	Множество отображается в стандартное множество ЯП
{ScienceData; in: type; psType: { enum: {enum_list: {broad, intermediate, narrow}};}}	<pre>public class ScienceData { public static enum Enum_psType { broad, intermediate, narrow}; Enum_psType psType; }</pre>	Для неименованных перечислений заводится тип перечисления Enum_psType в ЯП, а для слота psType, соответствующий ему атрибут типа Enum_psType.
Операции		
Арифметические: +, -, *, /	+ , - , * , /	Отображение операций
Строковые: +, in, нужно бы уж раписать с аргументами подробно substring, like, cardinal	<pre>+ str.contains(...) str.substring(...) Pattern p = Pattern.compile(regular expression(like)); Matcher m = p.matcher(str(строка)); boolean b = m.matches(); это в одно выражение str.length()</pre>	Отображение строковых операций
same(x,y) x = y (для объектов) x = y (для простых типов, не строк, и для перечислений) x = y (для строк)	<pre>x.isEqual(y) x.isEqual(y) x == y (x.compareTo(y) == 0)</pre>	Сравнение объектов
^	!	Операция отрицания

&,	&&,	Логические операции
Встроенные функции: sqrt, abs	Math.sqrt, Math.Abs	Математические функции
C1 => C2	result = true; if (M(C1)) result = result && M(C2);	Логическое следование
all x/Star (C1)(для типа)	result = M(C1);	Квантор всеобщности для типов
all x/Star (is_in(x, clsName) & C1) (для класса)	result = true; for (Star x: Cls_clsName.getSet()) {result = result && M(C1)}	Квантор всеобщности для классов
exist x/Star (is_in(x, clsName) & C1) (для класса)	result = false; for (Star x: Cls_clsName.getSet()) {result = result M(C1)}	Квантор существования для классов
Множества: x < y (x, y - множества), x <= y (x, y - множества), x > y (x, y - множества), x >= y (x, y - множества), is_in(x,y) (x элемент, y - множество) is_in(x,y) --!!!	y.containsAll(x) && (x.size() < y.size()) , y.containsAll(x), x.containsAll(y) && (x.size() > y.size()), x.containsAll(y), y.contains(x) В отдельных случаях, если is_in является предпосылкой импликации в пред-пост условиях функции, при этом y - это входной или выходной параметр (множеств) то is_in разворачивается в for: for (SetType x: y) { ... }	Операции над множествами
Переменные		
x (параметр функции)	x	
x (подкванторная переменная в инварианте)	this	
Пути		
x.a.b	x.getA().getB()	В ЯП используются get-методы
Константы		
123	123	
123.123	123.123	
'text'	"text"	
true	True	
point (перечислимый)	AstronomicalObject.Enum_morphology.point	
{3,4,5}	public class SynthesisHelper { public static <Type> Set<Type> set(Type[] input); } SynthesisHelper.<Integer>set(new Integer[]{Integer.valueOf(3),Integer.valueOf	Константа множество отображается в объект множество Set<Type>

<pre>{x,y,z} (x,y,z - типа CoordEQJ)</pre>	<pre>(4),Integer. valueOf (5)); SynthesisHelper.<CoordEQJ>set(new CoordEQJ[] {M(x),M(y),M(z)});</pre>	<p>соответствующего типа. Для этого используется generic функция.</p>
АТД		
<pre>{ScienceData; in: type; name: string; spatialCoord: CoordEQJ; },</pre>	<pre>public class ScienceData { String name; CoordEQJ spatialCoord; public boolean checkInvariant() { //ivariant methods call return result; } }</pre>	<p>Тип отображается в класс, для каждого типа образуется стандартный метод checkInvariant(); Даже если инварианты не описаны, метод всегда существует, просто всегда возвращает истинное значение.</p>
<pre>{CatalogData; in: type; supertype: ScienceData; ... },</pre>	<pre>public class CatalogData extends ScienceData { ... public boolean checkInvariant () { boolean result = super.checkInvariant(); //ivariant methods call return result; } }</pre>	<p>Наследование типов отображается в наследование классов ЯП. При этом наследуются также инварианты. Метод checkInvariant() переопределяется в наследнике, но вначале стоит вызов проверки инварианта у родителя: super.checkInvariant()</p>
Классы		
<pre>{scienceData; in: class; instance_section: ScienceData; },</pre>	<pre>public class Cls_scienceData { private class ScienceDataSet extends HashSet<ScienceData> implements Set<ScienceData> { @Override public boolean add(ScienceData e) { boolean result = super.add(e); if (!Cls_scienceData.checkInvariant()) throw new RuntimeException("ClassInvariantFailure"); return result; } ... @Override</pre>	<p>Класс отображается в класс ЯП, все методы и поля этого класса статические. В классе ЯП определяется множество, ScienceDataSet в котором переопределяются</p>

	<pre> public boolean remove(Object o) { boolean result = super.remove(o); if (!Cls_scienceData.checkInvariant()) throw new RuntimeException("ClassInvariantFailure"); return result; } @Override public boolean addAll(Collection<? extends ScienceData> c) { // Unsupported Operations исключение return false; } @Override public void clear() { // Unsupported Operations исключение } @Override public boolean removeAll(Collection<?> c) { // Unsupported Operations return false; } @Override public boolean retainAll(Collection<?> c) { // Unsupported Operations return false; } } private static ScienceDataSet scienceData; public static Set<ScienceData> getSet() { return this.scienceData; } public static boolean checkInvariant() { //Invariant Methods Call ... } } </pre>	<p>стандартные методы работы с множествами. В методах добавлена проверка инвариантов. В частности до и после операции вызывается проверка инварианта класса. Кроме того при добавлении объекта, объект добавляется только если инварианты его типа все выполняются.</p> <p>Также в классе, аналогично с типами определяется функция <code>checkInvariant()</code>; которая есть всегда, независимо от того определены ли инварианты.</p> <p>Также запрещаются операции addAll, clear, removeAll, retainAll</p>
<pre> { star; in: class; superclass: astronomicalObject; instance_section: Star; }, </pre>	<pre> public class Cls_star { private class StarSet extends HashSet<Star> implements Set<Star> { @Override public boolean add(Star e) { boolean result = super.add(e) && Cls_astronomicalObject.getSet().add(e); //Class CheckInvariant() return result; } ... @Override </pre>	<p>Наследование классов</p> <p>Наследуемые классы знают о своих родителях. Таким образом, если объект добавляется в класс, то он</p>

	<pre> public boolean remove(Object o) { boolean result = super.remove(o); откат if (result) result = Cls_astronomicalObject.getSet().remove(o); //Class CheckInvariant() return result; } private static StarSet star; public static Set<Star> getSet() ; ... } public class Cls_astronomicalObject { private class AstronomicalObjectSet extends HashSet<AstronomicalObject> implements Set<AstronomicalObject> { @Override public boolean add(AstronomicalObject e) { boolean toAdd = e.checkInvariant(); //Class CheckInvariant() if (toAdd) boolean result = super.add(e); } //Class CheckInvariant() return result; } ... @Override public boolean remove(Object o) { //Class CheckInvariant() boolean result = super.remove(o); if (result) result = Cls_star.getSet().remove(o); //Class CheckInvariant() return result; } private static AstronomicalObjectSet astronomicalObject; public static Set<AstronomicalObject> getSet() ; ... } </pre>	<p>добавляется и в родительский класс. Если объект удаляется из класса, то он удаляется и из родительского. Если объект добавляется в родительский класс, то очевидно в наследуемый он не попадает. Если объект удаляется из родительского класса, то он также удаляется и из наследуемого класса.</p>
<pre> { star; in: class; superclass: astronomicalObject; instance_section: Star; } , { intrestingStars; in: class; superclass: astronomicalObject; instance_section: Star; } </pre>	<pre> public class Cls_star { private class StarSet extends HashSet<Star> implements Set<Star> { @Override public boolean add(Star e) { boolean toAdd = e.checkInvariant(); //Class CheckInvariant() if (toAdd) { boolean result = super.add(e) && Cls_astronomicalObject.getSet().add(e); } } //Class CheckInvariant() return result; } } </pre>	<p>Рассмотрим пример, когда от одного родителя наследуется несколько классов. Один объект может быть одновременно в нескольких классах. Поэтому нужны дополнительные</p>

	<pre> ... @Override public boolean remove(Object o) { //Class CheckInvariant() boolean result = super.remove(o); if (result && !Cls_intrestingStars.getSet().contains(o)) result = Cls_astronomicalObject.getSet().remove(o); //Class CheckInvariant() return result; } private static StarSet star; public static Set<Star> getSet() ; ... } public class Cls_astronomicalObject { private class AstronomicalObjectSet etends HashSet<AstronomicalObject> implements Set<AstronomicalObject> { @Override public boolean add(AstronomicalObject e) { boolean toAdd = e.checkInvariant(); //Class CheckInvariant() if (toAdd) boolean result = super.add(e); } //Class CheckInvariant() return result; } } ... @Override public boolean remove(Object o) { //Class CheckInvariant() boolean result = super.remove(o); if (result) result = Cls_star.getSet().remove(o) && Cls_intrestingStars.getSet().remove(o); //Class CheckInvariant() return result; } private static AstronomicalObjectSet astronomicalObject; public static Set<AstronomicalObject> getSet() ; ... } </pre>	<p>проверки при удалении. Т.е. если объект удаляем из класса, то прежде чем удалять из родителя, необходимо удостовериться что в других наследниках того же самого типа нет этого объекта. Если он там есть, то в родители удалять не надо.</p> <p>Если же объект удаляется в родительском классе, то он должен быть удален во всех наследниках.</p>
Функции и методы		
<pre> { CoordEQJ; in: type; matchObjects: { in: function; params: {+o2/CoordEQJ, +rad1/real, +rad2/real, - returns/boolean}; </pre>	<pre> public class CoordEQJ { public boolean matchObjects(CoordEQJ o2, double rad1, double rad2); } </pre>	<p>спецификация метода</p>

<pre>}; }, { CoordEQJ; in: type; matchObjects: { in: function; params: {+o2/CoordEQJ, +rad1/real, +rad2/real, - result/boolean, - distance/real}; }; },</pre>	<pre>public class CoordEQJ { public class matchObjectsResult { public boolean result; public double distance; } public matchObjectsResult matchObjects(CoordEQJ o2, double rad1, double rad2); }</pre>	<p>Если возвращаемых переменных несколько, то для них заводится составной класс, который содержит результирующие атрибуты.</p>
<pre>{XMATCH; in: module, local, can_accept_data; function: {xmatch; in: function; params: { +ra1/real, +de1/real, +ra2/real, +de2/real, +rad/real, - returns/boolean }; }; }</pre>	<pre>public class XMATCH{ public static boolean xmatch(double ra1, double de1, double ra2, double de2, double rad); }</pre>	<p>Функции модуля отображаются в статическую функцию класса соответствующего модулю.</p>
Инварианты		
<pre>{ Star; in: type; supertype: AstronomicalObject; starInvariant: { in: predicate, invariant; { all x/Star (x.objectType = star & x.morphology = point) }; },</pre>	<pre>public class Star extends AstronomicalObject { private boolean starInvariant() { return (this.getObjectType() == AstronomicalObject.Enum_objectType.star) && (this.getMorphology() == AstronomicalObject.Enum_morphology.point); } public boolean checkInvariant() { boolean result = super.checkInvariant(); result = result & this.starInvariant(); return result; } }</pre>	<p>Инвариант для типа отображается в приватный метод, проверяющий истинность утверждения. В функции <code>checkInvariant</code> осуществляется вызов этого приватного метода.</p>
<pre>{ star; in: class; superclass: astronomicalObject; instance_section: Star; starInvariant: { in: predicate, invariant; { all x/Star (is_in(x, star) & x.objectType = star) } }; },</pre>	<pre>public class Cls_star { //StarSet Defenition ... private static StarSet star; public static Set<Star> getSet(); private static boolean starInvariant() { boolean result = true; for (Star x: Cls_star.getSet()){ result = result && (x.getObjectType() == AstronomicalObject.Enum_objectType.star); } return result; } public static boolean checkInvariant() { boolean result =</pre>	<p>Аналогично типам, инвариант класса отображается в приватный метод проверяющий истинность инварианты. Вызов инварианта осуществляется методом</p>

	<pre>Cls_astronomicalObject.checkInvariant(); result = result & this.starInvariant(); return result; } }</pre>	checkInvariant()
<pre>{ star; in: class; superclass: astronomicalObject; instance_section: Star; starInvariant: { in: predicate, invariant; { exist x/Star (is_in(x, star) & x.objectType = star) } }; },</pre>	<pre>public class Cls_star { //StarSet Defenition ... private static StarSet star; public static Set<Star> getSet(); private boolean starInvariant() { boolean result = false; for (Star x: Cls_star.getSet()){ result = result (this.getObjectType() == AstronomicalObject.Enum_objectType.star); } } return result; } public static boolean checkInvariant() { boolean result = Cls_astronomicalObject.checkInvariant(); result = result & this.starInvariant(); return result; } }</pre>	Для квантора существования, все аналогично квантору всеобщности
<pre>{ standard; in: class; superclass: star; instance_section: Standard; isolatedInvariant: { in: predicate, invariant; {{ all x/Standard y/AstronomicalObject (is_in(x, standard) & is_in(y, astronomicalObject) &^same(x,y) => (sqrt ((x.spatialCoord.ra - y.spatialCoord.ra) * (x.spatialCoord.ra - y.spatialCoord.ra) + (x.spatialCoord.de - y.spatialCoord.de) * (x.spatialCoord.de - y.spatialCoord.de)) > 30))}} };</pre>	<pre>public class Cls_standard { //StandardSet definition ... private static StandardSet standard; public static Set<Standard> getSet(); private static boolean isolatedInvariant(){ boolean result = true; for (Standard x: Cls_standard.getSet()) { for (AstronomicalObject y: Cls_astronomicalObject.getSet()) { if (!x.iseEqual(y)) { result = result && (Math.sqrt((x.getSpatialCoord().getRa()- y.getSpatialCoord().getRa()) * (x.getSpatialCoord().getRa()- y.getSpatialCoord().getRa()) + (x.getSpatialCoord().getDe()- y.getSpatialCoord().getDe()) * (x.getSpatialCoord().getDe()- y.getSpatialCoord().getDe())) > 30); } } } }</pre>	Условие инвариантов может быть связано с несколькими классами, тогда в реализации метода инварианта присутствует несколько вложенных циклов.
Функции		
<pre>colorExcess: { in: function; params: { +firstPB/Passband, +secondPB/Passband, - returns/real} ; { predicative; {</pre>	<pre>private double private_colorExcess(Passband firstPB, Passband secondPB); public double colorExcess(Passband firstPB, Passband secondPB) { If (!(firstPB.getInPhotometry() == secondPB.inPhotometry()))</pre>	Если для функции заданы пред-пост условия, то каждое условие проверяется до или после

<pre> firstPB.inPhotometry = secondPB.inPhotometry} }; </pre>	<pre> throw new RuntimeException("FunctionPreconditionFailure"); return this.private_colorExcess(fristPB, secondPB); } </pre>	<p>выполнения метода <code>private_colorExcess</code> реализующего само выполнение функции. В данном примере условие проверяется до вызова.</p>
<pre> getCorrectMagnitudes: { in: function; params: { +values/{ set; type_of_element: Magnitude;} , -magnitudes/{ set; type_of_element: Magnitude;} } ; { predicative; { all mag1/Magnitude, mag2/Magnitude (is_in(mag1, magnitudes) & is_in(mag2, magnitudes) & mag1.passband = mag2.passband => mag1 = mag2)) } }; }; } </pre>	<pre> private Set<Magnitude> private_ getCorrectMagnitudes(Set<Magnitude> values); public Set<Magnitude> getCorrectMagnitudes(Set<Magnitude> values) { Set<Magnitude> magnitudes = this. private_getCorrectMagnitudes(values); boolean result = true; for (Magnitude mag1: magnitudes) { for (Magnitude mag2: magnitudes) { if (mag1.getPassband() == mag2.getPassband()) if (!(mag1.isEqual(mag2))) result = false; } } if (!result) throw new RuntimeException("FunctionPredicateFailure"); return magnitudes; } </pre>	<p>В данном примере условие проверяется после выполнения функции.</p>
<pre> { Star; in: type; supertype: AstronomicalObject; starInvariant: { in: predicate, invariant; { all x/Star (x.objectType = star & x.morphology = point) } }; colorExcess: { in: function; params: { +firstPB/Passband, +secondPB/Passband, - returns/real} ; }; }, </pre>	<pre> public class Star extends AstronomicalObject { private boolean starInvariant() { return (this.getObjectType() == AstronomicalObject.Enum_objectType.star) && (this.getMorphology() == AstronomicalObject.Enum_morphology.point); } public boolean checkInvariant() { boolean result = super.checkInvariant(); result = result & this.starInveriant(); return result; } private double private_colorExcess(Passband firstPB, Passband secondPB); public double colorExcess(Passband firstPB, Passband secondPB) { if (!this.checkInvariant()) throw new RuntimeException("TypeInvariantFailure"); double result = this.private_colorExcess(fristPB, secondPB); if (!this.checkInvariant()) throw new RuntimeException("TypeInvariantFailure"); return result; } </pre>	<p>В данном примере продемонстрирован механизм вызова инвариантов типов в функциях. Для типа звезды определен инвариант, ему соответствует приватная функция проверяющая истинность условия, и метод <code>checkInvariant()</code> проверяющий все инварианты. При вызове функции инварианты проверяются до выполнения функции и после</p>

	}	выполнения.
<pre>{ Star; in: type; supertype: AstronomicalObject; colorExcess: { in: function; params: { +firstPB/Passband, +secondPB/Passband, - returns/real} ; }; }, { star; in: class; superclass: astronomicalObject; instance_section: Star; } , { intrestingStars; in: class; superclass: astronomicalObject; instance_section: Star; } } Пример реализации метода</pre>	<pre>public class Star extends AstronomicalObject { public boolean checkInvariant() { boolean result = super.checkInvariant(); return result; } private double private_colorExcess(Passband firstPB, Passband secondPB); public double colorExcess(Passband firstPB, Passband secondPB) { //Type Invariant Check double result = this.private_colorExcess(fristPB, secondPB); //Type Invariant Check if (!Cls_Star.checkInvariant()) thrown new RuntimeException("ClassInvariantFailure"); return result; } } public class Cls_Star extends Cls_AstronomicalObject { public static boolean checkInvariant() { boolean result = super.checkInvariant(); result = result && сначала проверка входит ли объект в класс Cls_star.checkInvariant() && Cls_interestingStars.checkInvariant(); return result; } }</pre>	<p>Рассмотрим пример вызова инвариантов классов. Когда изменяется объект, должны вызываться не только инварианты типа, но и инвариант классов в которых объект находится.</p> <p>Т.к. изменения объектов происходит только в функциях, поэтому проверка происходит в функциях. Кроме того, проверка происходит после работы функции, т.е. после того как объект изменен.</p> <p>Для каждого типа определяется класс Cls_Star (по имени типа) в котором есть статический метод <code>invarintCheck()</code>, который уже вызывает соответствующие методы соответствующих классов.</p>

Приложение Г Динамическое связывание языка СИНТЕЗ и языка программирования Java

Структуры языка программирования Java для представления элементов языка СИНТЕЗ при динамическом связывании

Представление схемы языка СИНТЕЗ

```
class Schema {
    String name;
    Module[] modules;
    static BooleanType BOOLEAN_TYPE = new BooleanType();
    static IntegerType INTEGER_TYPE = new IntegerType();
    static NoneType NONE_TYPE = new NoneType();
    static RealType REAL_TYPE = new RealType();
    static StringType STRING_TYPE = new StringType();
    static TimeType TIME_TYPE = new TimeType();
    public static SetType getSetType(Type elementType) {
        return elementType.isScalarBuiltinType()
            ? builtinTypeSets.get(elementType) : new SetType(elementType);
    }
}
```

Представление модуля языка СИНТЕЗ

```
class Module {
    Schema inSchema;
    String name;
    String[] metaClasses;
    Module[] imports;
    Class[] classes;
    Type[] types;
}
```

Представление класса языка СИНТЕЗ

```
class Class {
    Module inModule;
    String name;
    String[] metaClasses;
    Class[] superClasses;
    ADT instanceType;
}
```

Представление типа языка СИНТЕЗ

```
class Type {}
class BuiltinType extends Type {}
class BooleanType extends BuiltinType {}
class IntegerType extends BuiltinType {}
class RealType extends BuiltinType {}
class StringType extends BuiltinType {}
class TimeType extends BuiltinType {}
class NoneType extends BuiltinType {}
class SetType extends BuiltinType {
    Type elementType;
}
```

Представление функций языка СИНТЕЗ

```
class FunctionType extends Type {
    Module inModule;
}
```

```

String name;
String[] metaClasses;
public FunctionParameter[] functionParameters;
public String functionSpecification;
public String functionImplementation;
}
class FunctionParameter {
    static int IN = 1;
    static int OUT = 2;
    static int INOUT = 3;
    String name;
    Type type;
    int kind;
}

```

Представление абстрактных типов языка СИНТЕЗ

```

class ADT extends Type {
    Module inModule;
    String name;
    String[] metaClasses;
    ADT[] superTypes;
    Attribute[] attributes;
    Invariant[] invariants;
}
class Attribute{
    String name;
    Type type;
}
class Invariant {
    String name;
    ADT inADT;
    String logicalFormula;
}

```

Представление для составных типов языка СИНТЕЗ (соединение, пересечение)

```

class CompositionalType extends ADT {
    public ADT[] operands;
    public ADT actualType;
}
class JoinType extends CompositionalType {}
class MeetType extends CompositionalType {}

```

Представление редукта языка СИНТЕЗ

```

class Reduct extends ADT {
    Module inModule;
    ADT reductOf;
    ReductElement[] reductElements;
    ADT body;
}
class ReductElement {}
class ReductAttributeElement extends ReductElement {
    public Attribute from;
}
class ReductFromElementBase extends ReductElement {
    public String newName;
    public Type prjType;
}
class ReductFromAttrElement extends ReductFromElementBase {
    public Attribute from;
}

```

Структуры языка программирования Java для представления коллекций объектов языка СИНТЕЗ

```
class SynthClass {
    Schema schema;
    Set<SynthClassElemADT> elements;
}
```

```
class SynthClassElem{
    Type type;
    Object value;
}
```

```
class SynthClassElemADT extends SynthClassElem {
    SynthClassElemADT(ADT type, Map<String, SynthClassElem> elements){
        this.type = type;
        this.value = elements;
    }
}
```

```
class SynthClassElemBoolean extends SynthClassElem {
    SynthClassElemBoolean(boolean value){
        this.type = Schema.BOOLEAN_TYPE;
        this.value = Boolean.valueOf(value);
    }
}
```

```
class SynthClassElemDate extends SynthClassElem {
    SynthClassElemDate(long value){
        this.type = Schema.TIME_TYPE;
        this.value = new Date(value);
    }
}
```

```
class SynthClassElemInteger extends SynthClassElem {
    SynthClassElemInteger(long value){
        this.type = Schema.INTEGER_TYPE;
        this.value = Long.valueOf(value);
    }
}
```

```
class SynthClassElemReal extends SynthClassElem {
    SynthClassElemReal(double value){
        this.type = Schema.REAL_TYPE;
        this.value = Double.valueOf(value);
    }
}
```

```
class SynthClassElemString extends SynthClassElem {
    SynthClassElemString(String value){
        this.type = Schema.STRING_TYPE;
        this.value = value;
    }
}
```

```

class SynthClassElemSet extends SynthClassElem {
  SynthClassElemSet(Type instanceType, Set<SynthClassElem> elements){
    this.type = Schema.getSetType(instanceType);
    this.value = elements;
  }
  SynthClassElemSet(Type instanceType, SynthClassElem[] elements){
    this.type = Schema.getSetType(instanceType);
    for (SynthClassElem elem: elements)
      value.add(elem);
  }
}

```

Синтаксис инвариантов языка СИНТЕЗ, отображаемых в ЯП

```

< invariant > ::=
  < invariant identifier > : { in: predicate, invariant;
    {predicative; {
      all <typed variable list> (
        [ is_in(<collection identifier>, <variable>)
          [& is_in(<collection identifier>, <variable>)]... ->]
        < condition composition > )
      |
      exists <typed variable list> (
        [ is_in(<collection identifier>, <variable>)
          [& is_in(<collection identifier>, <variable>)]... &]
        < condition composition > )
    }}}
}

< condition conjunction > ::= < condition > | < condition > [& < condition >]...
< condition composition > ::= < condition > | (< condition composition >) |
  < condition composition > & < condition composition > |
  < condition composition > | < condition composition >
< condition > ::=
  < arithmetic expression > < sigma > < arithmetic expression > |
  < collection value > < set predicate sign > < collection value >
< collection value > ::= < typed variable > | < collection constant >
< collection constant > ::= { < term > [, < term >]... } | {“[< term > [, < term >]... “]”}
< set predicate sign > ::= < | < = > | > = > | >
< sigma > ::= = | < | < = > | > = > | >
< arithmetic expression > ::= < additive > | < additive operation > < additive > |
< arithmetic expression > < additive operation > < additive >
< additive operation > ::= + | -
< additive > ::= < multiplier > | < additive > < multiplicative operation > < multiplier >
< multiplicative operation > ::= * | /
< multiplier > ::= < variable > | < number > | (< arithmetic expression >)

< term > ::= < typed variable > | < arithmetic expression > | < value >
< typed variable > ::= < variable > /< type name >
< value > ::= none | < constant >
< constant > ::= < collection constant > | < string > | < number > | < Boolean constant >

```

Синтаксис предикативных спецификаций методов АД языка СИНТЕЗ, отображаемых в ЯП

```

< method > ::=
  < method identifier >: {in: function; { predicative; {< pre-post condition conjunction >}}}
< pre-post condition conjunction > ::= < pre-post condition > [ & < pre-post condition > ]...
< pre-post condition > ::= < condition composition > |
  (< condition composition >) => < condition conjunction > |
  all <typed variable list> (
    is_in(<variable identifier>, <term>) [& is_in(<variable identifier>, <term>)]...
    [& (< condition composition >)] =>
    < condition conjunction >))

```

Алгоритм интерпретации инвариантов и спецификаций методов при динамическом связывании

Инварианты

С точки зрения интерпретации различаются инварианты типов и классов.

В случае инвариантов типа вида $\text{all } x/T (F)$, проверка осуществляется для конкретных объектов, поэтому кванторы всеобщности игнорируются, и осуществляется интерпретация формулы F специальной функцией `checkConditionComposition()`.

В случае инвариантов класса, каждый квантор разворачивается в цикл (инвариант проверяется для всех существующих в данный момент объектов класса). Если кванторов несколько, то они разворачиваются во вложенные циклы. При интерпретации квантора всеобщности $\text{all } x/T (\text{is_in}(C, x) \rightarrow F)$ проверяется, верно ли условие `checkCondition(F)` для всех итераций цикла (объектов класса). При интерпретации квантора существования проверяется, выполняется ли условие хотя бы на одной итерации цикла.

Пример. Инвариант класса с квантором всеобщности

```
{ star; in: class;
  instance_section: Star;
  starInvariant:
  { in: predicate, invariant;
    { all x/Star (is_in(star, x) -> F) }
  };
},
```

интерпретируется следующим образом:

```
result:= true;
foreach(x: star){
  result := result && checkConditionComposition(F);
}
if(result = true) then return true;
```

Пример. Инвариант класса с квантором существования

```
{ star; in: class;
  instance_section: Star;
  starInvariant:
  { in: predicate, invariant;
    { exist x/Star (is_in(star, x) & F) }
  };
},
```

интерпретируется следующим образом:

```
result:= false;
foreach(x: star){
  result := result || checkConditionComposition(F);
}
```

```

}
if(result = true) then return true;

```

Опишем теперь алгоритм интерпретации условий

checkConditionComposition(F). Без ограничения общности можно считать, что представляет собой дизъюнкцию конъюнкций условий C_{ij} вида $\langle \text{condition} \rangle$, поэтому функция checkConditionComposition представляет собой условия:

```

if(D11 & ... & D1n1 | D21 & ... & D2n2 | ... | Dm1 & ... & Dmnm)
  result:= true;
else
  result:= false;

```

Здесь $D_{ij} = \text{checkCondition}(C_{ij})$.

Функция checkCondition(C) выглядит следующим образом:

```

if(C = A1 sigma A2, где Ai ∈ <arithmetic expression>, sigma ∈ <sigma>)
then
  вычислить e(A1) - значение A1;
  вычислить e(A2) - значение A2;
  return истинность условия e(A1) sigma e(A2);
if(C = B1 sigma B2, где Bi ∈ <collection value>, sigma ∈ <set predicate sign>)
  вычислить e(B1) - значение B1;
  вычислить e(B2) - значение B2;
  return истинность условия e(B1) sigma e(B2)

```

Методы

Спецификация метода представляет собой конъюнкцию формул вида $\langle \text{pre-post condition} \rangle$. Интерпретация всех таких конъюнктов осуществляется независимо: функция интерпретации checkPrePostCondition должна возвращать истину на каждом конъюнкте. При этом все конъюнкты делятся на две группы: предусловие (конъюнкты, оперирующие только входными переменными) и постусловие (все остальные конъюнкты).

Истинность интерпретирующей функции на конъюнктах предусловия проверяется до вызова метода, на конъюнктах постусловия – после завершения метода.

Функция checkPrePostCondition(P) выглядит следующим образом:

```

if(P ∈ < condition composition >)
then return checkConditionComposition(P)
else if(P = C1 -> C2)
then
  return (not checkConditionComposition(C1) or checkConditionConjunction(C2))
else
if( P = all x1/T1, ... , xn/Tn(is_in(c1, x1) & ... & is_in(c1, x1) & C1 -> C2) )
then
  result = true;
  foreach(x1: c1)
  ...
  foreach(xn: cn)

```

```
result = result &&  
    (not checkConditionComposition(C1) ||  
     checkConditionConjunction(C2)  
    )  
return result;
```

Функция `checkConditionConjunction(C1 & ... & Cn)` выглядит следующим

образом:

```
if(checkCondition(C1) && ... && checkCondition(C2)) return true;  
else return false;
```

Приложение Д Спецификация адаптера

Спецификация интерфейса адаптера

Информационные ресурсы неоднородны, но адаптеры предоставляют унифицированный доступ к ним. Интерфейс взаимодействия адаптеров был разработан, чтобы обеспечить возможность эффективного адаптивного планирования. Интерфейс был ранее кратко описан в разделе 4.3. Здесь приводится более подробное описание методов.

```
public interface RemoteAdapter {
    boolean isSameResource(RemoteAdapter ra);
    void echoRequest() throws ConnectionException;
    void remoteEchoRequest(RemoteAdapter adapter) throws ConnectionException,
    InternalErrorException, NotImplementedException;
    long getAdapterCapability(AdapterCapability capability) throws ConnectionException;
    boolean adapterSelectExtendedCapability(boolean opOnSchemaData, String op,boolean
    left_op_const,String left_op_type,boolean right_op_const, String right_op_type) throws
    ConnectionException, NotImplementedException;
    Module getAdapterSchemeModule();
    Module getAdapterSessionModule(int sessionID);
    int openSession() throws ConnectionException, OverloadException;
    void closeSession(int sessionID)throws ConnectionException, InvalidArgumentException;
    void keepAlive(int sessionID)throws ConnectionException, InvalidArgumentException;
    long getSessionCapability(int sessionID,SessionCapability capability) throws
    ConnectionException, InvalidArgumentException;
    int registerQuery(int sessionID,Plan plan)throws ConnectionException,
    InvalidArgumentException;
    void closeQuery(int sessionID,int queryID)throws ConnectionException,
    InvalidArgumentException;
    int openReaderOnQuery(int sessionID, int queryID,String rowIdAttributeName, long startNumber)
    throws ConnectionException, InvalidArgumentException;
    int materializeReader(int sessionID,int readerID) throws
    ConnectionException,InvalidArgumentException,InternalErrorException,NotImplementedException,LowSp
    aceException;
    int materializeReader(int sessionID,int readerID,long maxBytes,long maxRows) throws
    ConnectionException,InvalidArgumentException,InternalErrorException,NotImplementedException,LowSp
    aceException;
    int LoadRemoteData(int sessionID,RemoteAdapter adapter,int remoteSessionID,int
    remoteReaderID) throws
    ConnectionException,InvalidArgumentException,NotImplementedException,InternalErrorException,LowSp
    aceException;
    int LoadRemoteData(int sessionID,RemoteAdapter adapter,int remoteSessionID,int
    remoteReaderID,long maxBytes,long maxRows) throws ConnectionException,InvalidArgumentException,
    NotImplementedException, InternalErrorException, LowSpaceException;
    void removeData(int sessionID,int dataSetID) throws ConnectionException,
    InvalidArgumentException,NotImplementedException;
    long freeSpaceAvailable(int sessionID) throws ConnectionException, InvalidArgumentException,
    NotImplementedException;
    String getData(int sessionID, int readerID, long maxBytes, long maxRows) throws
    ConnectionException, InvalidArgumentException;
    long dataAvailable(int sessionID, int readerID) throws ConnectionException,
    InvalidArgumentException;
    long rowsSent(int sessionID,int readerID) throws ConnectionException,
    InvalidArgumentException;
    long bytesSent(int sessionID,int readerID) throws
    ConnectionException,InvalidArgumentException;
```

```

void closeReader(int sessionID, int readerID) throws ConnectionException,
InvalidArgumentException;

/*Query Estimating Functions*/
long estimateRows(int sessionID,int queryID) throws ConnectionException,
InvalidArgumentException;
long estimateDataSize(int sessionID,int queryID) throws ConnectionException,
InvalidArgumentException;
long averageRowSize(int sessionID,int queryID) throws ConnectionException,
InvalidArgumentException;
int getSampleDataReader(int sessionID, int queryID,long maxBytes, long maxRows, String
rowIdAttributeName, long startNumber) throws ConnectionException, InvalidArgumentException;

/*Control*/
ArrayList<ReaderTaskStats> getReaderTasks(int sessionID,int readerID)throws
ConnectionException;
ArrayList<DataLoadTaskStats> getDataLoadTasks(int sessionID) throws ConnectionException;
}

```

Интерфейс *RemoteAdapter* имеет несколько реализаций. Выбор реализации и создание их экземпляров зависит от конкретного используемого адаптера.

Все методы интерфейса должны допускать одновременное использование из нескольких потоков (*thread safe*). Также, все методы интерфейса синхронны, то есть блокируют выполнение вызывающего потока до завершения запрошенной операции. При реализации адаптеров особое внимание уделялось чтобы избегать ситуаций, потенциально вызывающих взаимоблокировки при синхронизации потоков.

Все методы адаптера являются блокирующими *synchronized*. Исключение составляют методы: получения данных – *getData*, получения статистики – *rowsSent*, *bytesSent*, *getDataLoadTasks*, *getReaderTasks*, загрузки данных с других адаптеров – *loadRemoteData*.

Ниже представлено описание методов в составе интерфейса.

boolean isSameResource(RemoteAdapter ra)

Этот метод должен возвращать истинное значение, если и только если объект *RemoteAdapter*, для которого вызван метод (*this*) и параметр *ra* взаимодействуют с одним и тем же адаптером одного и того же ресурса. Этот метод необходим, т.к. отдельный экземпляр класса, реализующего данный интерфейс, будет создаваться планировщиком на каждый модуль в схеме запроса. Если на адаптере объявлено несколько модулей, потребуется

определить, являются ли экземпляры ссылками на один и тот же адаптер. Фактически, при наличии этого метода станет возможным объявлять несколько модулей в одном адаптере. Все наборы адаптеров для модулей, все пары из которых возвращают истинное значение на данном методе, будут считаться при планировании единым источником. А лишние экземпляры этого интерфейса будут удалены. В качестве реализации, допустимо и предпочтительно сравнение на равенство URI адаптеров источников. В случае, если сравнение на равенство URI произвести невозможно, должно возвращаться значение ложь.

void echoRequest() throws ConnectionException

Этот метод проверяет, есть ли связь с адаптером, в случае отсутствия таковой выбрасывается исключение.

void remoteEchoRequest(RemoteAdapter adapter)

Этот метод заставляет адаптер this отправить echoRequest адаптеру adapter. В случае отсутствия связи посредника с адаптером this, выбрасывается ConnectionException. В случае отсутствия связи адаптера this с адаптером adapter выбрасывается InternalErrorException.

Данный метод может быть не реализован, что означает, что некоторые адаптеры будут всегда при его вызове выбрасывать NotImplementedException.

long getAdapterCapability(AdapterCapability capability)

Опрос свойств адаптера. Подробно различные возможности описаны в разделе 6.1.2. В текущей реализации все свойства кодируются целыми числами. Параметр AdapterCapability представляет собой целое число. Набор его возможных значений определен в соответствующем типе. Адаптер, встретив незнакомое ему значение параметра, обязан вернуть 0.

boolean adapterSelectExtendedCapability(boolean opOnSchemaData, String op, boolean left_op_const, String left_op_type, boolean right_op_const, String right_op_type)

Данный метод может быть не реализован. Если данный метод реализован, то он должен возвращать истинное значение, если и только если операция

Select с аргументами, описанными в параметрах, выполняема на данном адаптере. Здесь под «типом» аргумента подразумевается имя встроенного типа в языке СИНТЕЗ. Под «операцией» подразумевается любая допустимая операция над данной парой типов в языке Asyfs.

Module getAdapterSchemeModule()

Данный метод возвращает модуль, в котором содержатся классы адаптера.

Module getAdapterSessionModule(int sessionID)

Данный метод возвращает модуль сессии адаптера, создаваемый адаптером на каждую сессию. В этом модуле содержатся как текущие классы адаптера, так и временные классы, соответствующие данным полученным от других адаптеров.

int openSession()

Данный метод открывает новую сессию и возвращает её идентификатор. Если для открытия сессии недостаточно ресурсов, то выбрасывается соответствующее исключение. В случае получения данного исключения, планировщик попытается снова открыть сессию через случайный промежуток времени от 0 до k миллисекунд, после каждой неудачной попытки k будет умножаться на 2. Предпринимается не более n попыток.

Сессия – единица управления ресурсами адаптера. Адаптер волен решать, какой объем ресурсов выделить пользователю в момент открытия сессии, но выделив, обязан резервировать их за данной сессией вплоть до её закрытия.

void closeSession(int sessionID)

Данный метод закрывает сессию и уничтожает все связанные с ней ресурсы. Все текущие операции, открытые на данной сессии должны быть прерваны, и все ресурсы, выделенные под сессию, освобождены к моменту возврата из данного метода.

void keepAlive(int sessionID)

Данный метод поддерживает сессию в открытом состоянии. Сессия существует с момента открытия до момента закрытия соответствующим методом. Тем не менее должен существовать механизм, для того чтобы ресурсы выделенные для сессии освобождались, если выполнение запроса прервано не штатно. Таковым механизмом является таймаут. Если к данной сессии не было обращений в течение *n* минут, сессия закрывается принудительно. Если планировщику необходимо поддерживать сессию открытой, несмотря на то, что она не используется, он должен использовать этот метод.

long getSessionCapability(int sessionID, SessionCapability capability)

Этот метод используется для опроса свойств характерных для конкретной сессии. К таким свойствам относятся объемы ресурсов, выделенных на данную сессию.

int registerQuery(int sessionID, Plan plan)

Метод регистрирует новый запрос и возвращает его номер (уникальный в пределах сессии). Если в параметрах есть какая-то ошибка, возвращает соответствующее исключение.

void closeQuery(int sessionID, int queryID)

Метод уничтожает ранее зарегистрированный запрос и освобождает выделенные ресурсы и память. Если в данный момент этот запрос каким-то образом используется, например, открыты ридеры на этот запрос, то они уничтожаются по цепочке.

int openReaderOnQuery(int sessionID, int queryID, String rowIdAttributeName)

Метод отрывает ридер на запрос и возвращает его номер. Этот номер должен быть уникальным в пределах сессии. Ридер в дальнейшем может быть использован для получения данных – результата выполнения запроса. Если *rowIdAttributeName* не null, то к результату должен быть присоединен

идентификатор строки. Имя атрибута, содержащего идентификатор, должно равняться указанному.

int materializeReader(int sessionID,int readerID)

Этот метод может быть не реализован. Подробнее см. в описании AdapterCapability. Метод создает временную коллекцию максимального возможного объема из данных, которые возвращает ридер, указанный в параметре readerID, но не обязательно из всех данных. Возвращается идентификатор созданной коллекции.

int materializeReader(int sessionID,int readerID, long maxBytes, long maxRows)

Аналогичен по семантике предыдущему методу, но, дополнительно, позволяет ограничить размер создаваемой коллекции, как в байтах, так и в количестве объектов, экземпляров в классе-результате

int loadRemoteData(int sessionID,RemoteAdapter adapter,int remoteSessionID,int remoteReaderID)

Данный метод создает коллекцию максимального возможного размера и возвращает её идентификатор. В коллекцию помещаются данные, находящиеся в адаптере adapter, на котором должна существовать открытая сессия remoteSessionID и в котором должен существовать открытый ридер с именем remoteReaderID. В случае возникновения ошибки в процессе обращения адаптера this к адаптеру adapter, выбрасывается исключение InternalErrorException с правильно заполненным полем inner_exception. Этот метод может быть не реализован.

int loadRemoteData(int sessionID,RemoteAdapter adapter,int remoteSessionID,int remoteReaderID, long maxBytes, long maxRows)

Этот метод аналогичен предыдущему методу, но, дополнительно, позволяет ограничить размер создаваемой коллекции, как в байтах, так и в кортежах. Этот метод может быть не реализован.

void removeData(int sessionID,int dataSetID)

Данный метод удаляет коллекцию с идентификатором dataSetID в сессии sessionID. Рекурсивно должны быть удалены все запросы, использующие эту коллекцию и рекурсивно все открытые ридеры на этих запросах. Все ресурсы должны быть удалены и освобождены до возврата из данного метода. Этот метод может быть не реализован.

long freeSpaceAvailable(int sessionID)

Возвращает объем свободного места, гарантированно доступного для создания временных коллекций в данный момент (в байтах).

String getData(int sessionID, int readerID, long maxBytes, long maxRows)

Этот метод используется для получения данных из открытого в данный момент ридера. Метод должен вернуть SynthClass, содержащий не более maxRows объектов, экземпляров в классе-результате и не более maxBytes байт. Можно вернуть пустой SynthClass, если текущий кортеж не укладывается в ограничения. Каждое обращение сдвигает положение ридера. То есть на одном и том же ридере невозможно прочитать одни и те же данные дважды, если это не пустой SynthClass.

long dataAvailable(int sessionID, int readerID)

Метод возвращает размер в байтах объекта, на который указывает ридер. Если такого объекта не существует, то возвращается 0, что должно означать, что весь ридер прочитан.

long rowsSent(int sessionID,int readerID)

Метод возвращает количество объектов, которые уже были прочитаны через данный ридер. В случае, если в данный момент выполняется передача данных, то есть активен вызов getData, возвращаемое число должно быть на отрезке от его значения до начала передачи до его значения в момент окончания. Последовательные обращения к этому методу должны генерировать неубывающую последовательность.

long bytesSent(int sessionID,int readerID)

Метод возвращает количество байт, которые уже были прочитаны через данный ридер. Требования к значению на промежуточных этапах во время передачи данных такие же, как и у предыдущего метода.

void closeReader(int sessionID, int readerID)

Закрывает ридер и освобождает все связанные с ним ресурсы. Запрос, на котором он открыт, не уничтожается и может быть использован для повторного открытия ридера.

long estimateRows(int sessionID,int queryID)

Оценочная функция, обязательна к реализации всеми адаптерами. Возвращает приблизительное ожидаемое количество объектов, экземпляров в классе-результате.

long estimateDataSize(int sessionID,int queryID)

Оценочная функция, обязательна к реализации всеми адаптерами. Возвращает приблизительный ожидаемый объем данных в результате запроса в байтах.

long averageRowSize(int sessionID,int queryID)

По сути своей является отношением результата estimateDataSize к estimateRows. Все замечания, относящиеся к этим методам, справедливы и для данного метода.

int getSampleDataReader(int sessionID, int queryID,long maxBytes, long maxRows, String rowIdAttributeName)

Функция строит тестовую выборку и возвращает идентификатор ридера, через который можно считать эту выборку. Полученная выборка должна содержать не более maxBytes байт и не более maxRows объектов, экземпляров в классе-результате. Семантика rowIdAttributeName такая же как в методе openReaderOnQuery.

`ArrayList<ReaderTaskStats> getReaderTasks(int sessionID, int readerID)`

Данный метод возвращает статистику вызовов метода `getData` над некоторым ридером. Параметром является номер сессии и ридер, статистику вызовов метода `getData` которого мы хотим получить. Статистка описана классом:

```
public class ReaderTaskStats
{
    /**
     * Номер задания. Увеличивающийся счетчик обращений к функции, уникален в пределах сессии.
     */
    public long readerTaskID;
    /**
     * 0 - в очереди, 1 - выполняется, 2 - завершена
     */
    public int status;
    /**
     * идентификатор Ридера.
     */
    public long readerID;
    /**
     * Время постановки задания в очередь, если очереди нет - оно совпадает со временем
     * начала исполнения, т.к. данная структура генерируется отдельно для каждого
     * вызова getData; очевидно, что временем начала выполнения считается момент,
     * начала работы этой функции.
     */
    public Date enqueued;
    /**
     * Время начала исполнения задания.
     * Время, когда метод getData был вызван.
     */
    public Date beginexec;
    /**
     * Время завершения, к этому момента уже должен быть построен SynthClass
     * Это поле необходимо заполнять непосредственно перед вызовом return
     * (как можно ближе) на стороне адаптера при выходе из метода getData
     * Даты могут быть любыми (имеется в виду часовой пояс), главное чтобы
     * система исчисления времени была общей для всего ридера, желательно
     * GMT, или локальное время
     */
    public Date complete;
    /**
     * объем данных согласно стандартному алгоритму определения объема, измеряющему
     * размеры коллекций и свободное место
     */
    public long dataBytesSent;
    /**
     * данные, переданные по сети, фактически - длина строки результата
     */
    public long networkBytesSent;
}
```

`ArrayList<DataLoadTaskStats> getDataLoadTasks(int sessionID)`

Данный метод возвращает статистику вызовов метода `loadRemoteData`. Параметром является номер сессии, статистика вызовов метода `loadRemoteData`, которую мы хотим получить. Статистка описана классом:

```

public class DataLoadTaskStats
{
    /**
     * уникальный в пределах сессии идентификатор
     */
    public long dataSetTaskID;
    /**
     * 0 - в очереди, 1 - выполняется, 2 - завершена
     */
    public int status;
    /**
     * Идентификатор новой созданной коллекции, если уже известен. 0 иначе
     */
    public int dataSetID;
    /**
     * Время начала исполнения задания.
     * Время, когда метод getData был вызван.
     */
    public Date enqueued;
    /**
     * Время завершения, к этому момента уже должен быть построен SynthClass
     * Это поле необходимо заполнять непосредственно перед вызовом return
     * (как можно ближе) на стороне адаптера при выходе из метода getData
     * Даты могут быть любыми (имеется в виду часовой пояс), главное чтобы
     * система исчисления времени была общей для всего ридера, желательно
     * GMT, или локальное время
     */
    public Date beginexec;
    /**
     * Количество вызовов getData на удалённом адаптере, которое потребовалось,
     * чтобы загрузить данные (если они загружались маленькими порциями)
     */
    public long getDataCalls;
    /**
     * Инициализируется нулем. Каждый раз, вызывая на удаленном адаптере getData
     * добавляем разницу времени между началом и завершением вызова.
     */
    public long totalTimeOnRemoteCalls;
    /**
     * Время, потраченное на разбор SynthClass и помещение данных в хранилище.
     */
    public long totalTimeStoringData;
    /**
     * Суммарное время, потраченное на ожидание освобождения ресурсов (в цикле),
     * например, если количество потоков, которые могут одновременно писать, ограничено,
     * а фактически заданий больше.
     */
    public long totalTimeWaitingForResouces;
    /**
     * объем данных согласно стандартному алгоритму определения объема, определяющему
     * размеры коллекций и свободное место
     */
    public Date complete;
    /**
     * объем данных согласно стандартному алгоритму определения объема,
     * определяющему размеры коллекций и свободное место, равняется
     * размеру новосозданной коллекции.
     */
    public long totalDataBytesRecv;
    /**
     * данные, переданные по сети, фактически - сумма длин строкового
     * представления SynthClass
     */
    public long totalNetworkBytesRecv;
}

```

Очень важно, чтобы во всех функциях всех адаптеров одни и те же данные имели одинаковый измеримый размер в байтах. К сожалению, длина в байтах, представленная в виде SynthClass, этому критерию не отвечает, потому ниже будет описан алгоритм определения размера кортежа в байтах. Этим алгоритмом должны измеряться размеры передаваемых блоков данных. Также коллекции, которые ограничены по размеру, через ограничения сессии, должны уметь принимать любую допустимую коллекцию, измерение объема которой данным алгоритмом не превышает указанного ограничения. То же справедливо в плане общего ограничения на объем принимаемых данных.

Фиксируем размер стандартных типов данных.

Integer – 8 байт

Real – 8 байт

Date(time) – 8 байт

Для всех типов с переменной длиной справедливо следующее представление.

<объем данных, описывающих экземпляр, 8 байт><собственно данные>

Так, для конкретной строки объем будет равен 8+<длина UTF-8 представления этой строки>. Для пустого множества объем в байтах будет равен 8.

Если составной тип имеет атрибут с переменной длиной, то сам этот тип тоже переменной длины. Так, например, экземпляр типа описываемого в виде {int a; string s;} будет иметь длину равную 8(содержит длину всего экземпляра)+8(длина типа int)+8(содержит длину строки в UTF-8)+<количество байт в UTF-8 представлении строки>.

Спецификация ограниченных возможностей адаптеров

Планировщик выполняет запрос на информационных ресурсах. Различные информационные ресурсы и их адаптеры обладают различными ограничениями на выполнение тех или иных операций. Эти ограничения ресурсы сообщают планировщику посредством адаптеров через

соответствующий интерфейс. Формулируя запросы к ресурсам, планировщик обязан соблюдать ограничения. К таким ограничениям могут относиться запреты на выполнение:

- Union;
- Join;
- Append{id()};
- некоторых бинарных операций над некоторыми типами.

Ключевой характеристикой адаптеров является наличие или отсутствие возможности принимать данные от других адаптеров.

Все вышеупомянутые ограничения формулируются отдельно как для данных, которые предоставляет информационный ресурс (локальных), так и для данных, которые приняты от других адаптеров (внешних). Интерфейс для описания возможностей адаптеров представлен ниже:

```
public class AdapterCapability {
    public final int Value;
    public static final AdapterCapability UserDataSetsSupport = new AdapterCapability(0);
    public static final AdapterCapability UserDataSetsComplexQuery = new AdapterCapability(1);
    public static final AdapterCapability RemoteUserDataLoad = new AdapterCapability(2);
    public static final AdapterCapability CanExecuteSchemaJoin = new AdapterCapability(4);
    public static final AdapterCapability CanExecuteTempJoin = new AdapterCapability(5);
    public static final AdapterCapability CanExecuteMixedJoin = new AdapterCapability(6);
    public static final AdapterCapability CanExecuteSchemaUnion = new AdapterCapability(7);
    public static final AdapterCapability CanExecuteTempUnion = new AdapterCapability(8);
    public static final AdapterCapability CanExecuteMixedUnion = new AdapterCapability(9);
    public static final AdapterCapability CanExecuteSchemaSelect = new AdapterCapability(10);
    public static final AdapterCapability CanExecuteTempSelect = new AdapterCapability(11);
    public static final AdapterCapability SupportsExtendedSelectCapability = new
AdapterCapability(18);
    public static final AdapterCapability CanExecuteSchemaSetTypeOperations = new
AdapterCapability(12);
    public static final AdapterCapability CanExecuteSetTypeOperationsOnTempData = new
AdapterCapability(13);
    public static final AdapterCapability CanAcceptTempSetTypeData = new AdapterCapability(14);
    public static final AdapterCapability AllowSchemaMethodsOnTempData = new
AdapterCapability(15);
    public static final AdapterCapability CanExecuteTempAppendID = new AdapterCapability(16);
    public static final AdapterCapability CanExecuteSchemaAppendID = new AdapterCapability(17);
    public AdapterCapability(int v)
    {
        Value=v;
    }
}
```

Рассмотрим по порядку все значения *AdapterCapability*. На все незнакомые значения *getAdapterCapability* должна возвращать 0.

UserDataSetsSupport – Если в результате вызова *getAdapterCapability* с этим параметром получено значение отличное от нуля, это значит, что адаптер поддерживает временные коллекции.

UserDataSetsComplexQuery – *getAdapterCapability* может, но не обязан возвращать отличное от нуля значение при вызове с этим параметром, если и только если возвращается отличное от нуля значение при вызове с параметром *UserDataSetsSupport*. Если возвращается значение равное нулю, это означает, что использование временных коллекций в запросах сильно ограничено. Единственным допустимым запросом с использованием временных коллекций при этом будет запрос, который запрашивает, без каких либо изменений, все данные, содержащиеся в одной конкретной коллекции.

RemoteUserDataLoad – *getAdapterCapability* может, но не обязан, возвращать отличное от нуля значение при вызове с этим параметром, если и только если возвращается отличное от нуля значение при вызове с параметром *UserDataSetsSupport*. Фактически, это означает, что адаптер поддерживает загрузку данных с других адаптеров.

CanExecuteSchemaJoin – *getAdapterCapability* должна возвращать отличное от нуля значение, если поддерживается операция *Join* над двумя и более коллекциями, объявленными в схеме источника (не временными).

CanExecuteTempJoin – *getAdapterCapability* может, но не обязан возвращать отличное от нуля значение при вызове с этим параметром, если и только если возвращается отличное от нуля значение при вызове с параметром *UserDataSetsComplexQuery*. Если возвращается отличное от нуля значение, это означает, что на адаптере допустимо выполнение операции *Join* над двумя или более временными коллекциями.

CanExecuteMixedJoin – *getAdapterCapability* может, но не обязан возвращать отличное от нуля значение при вызове с этим параметром, если и только если возвращается отличное от нуля значение при вызове с параметром *UserDataSetsComplexQuery*. Если возвращается отличное от нуля значение, это

означает, что адаптер способен выполнить операцию Join над парой операндов, один из которых является результатом выполнения каких-либо допустимых на этом адаптере операций над коллекциями, объявленными в схеме, а второй над временными коллекциями. Если данный вид операции не поддерживается самим ресурсом, то её можно реализовать, материализовав результат выполнения запроса-операнда во временной коллекции, и выполнить эту операцию уже над парой временных коллекций, если такая операция поддерживается (это будет уметь выяснять планировщик). Если данный флаг установлен, то адаптер обязан уметь присоединять к результату подобной операции любое количество коллекций, объявленных в схеме источника, и, если дополнительно установлен флаг, разрешающий подобную операцию над парой временных коллекций, то может присоединять и любое количество временных коллекций. Над результатом выполнения подобной операции должны быть выполнимы любые операции, которые разрешено выполнять над временной коллекцией.

CanExecuteSchemaUnion – *getAdapterCapability* должна возвращать отличное от нуля значение, если поддерживается операция Union над двумя и более коллекциями объявленными в схеме ресурса (не временными).

CanExecuteTempUnion – *getAdapterCapability* может, но не обязан возвращать отличное от нуля значение при вызове с этим параметром, если и только если возвращается отличное от нуля значение при вызове с параметром *UserDataSetsComplexQuery*. Если возвращается отличное от нуля значение, это означает, что на адаптере допустимо выполнение операции Union над двумя или более временными коллекциями.

CanExecuteMixedUnion – *getAdapterCapability* может, но не обязан возвращать отличное от нуля значение при вызове с этим параметром, если и только если возвращается отличное от нуля значение при вызове с параметром *UserDataSetsComplexQuery*. Если возвращается отличное от нуля значение, это означает, что адаптер способен выполнить операцию Union над парой

операндов, один из которых является результатом выполнения каких-либо допустимых на этом адаптере операций над коллекциями, объявленными в схеме, а второй – над временными коллекциями. Если данный вид операции не поддерживается самим источником, то её можно реализовать, материализовав результат выполнения запроса-операнда во временной коллекции, и выполнить эту операцию уже над парой временных коллекций, если такая операция поддерживается (это будет уметь выяснять планировщик). Если данный флаг установлен, то адаптер обязан уметь присоединять к результату подобной операции любое количество коллекций, объявленных в схеме ресурса, и, если дополнительно установлен флаг, разрешающий подобную операцию над парой временных коллекций, то может присоединять и любое количество временных коллекций. Над результатом выполнения подобной операции должны быть выполнимы любые операции, которые разрешено выполнять над временной коллекцией.

CanExecuteSchemaSelect – *getAdapterCapability* должна возвращать отличное от нуля значение при вызове с этим параметром, если и только если поддерживается операция *Select* над данными в составе схемы (и любыми данными, которые можно получить в результате допустимых операций над данными в схеме) в полном объеме.

CanExecuteTempSelect – *getAdapterCapability* должна возвращать отличное от нуля значение при вызове с этим параметром, если и только если поддерживается операция *Select* над временными данными в составе схемы (и любыми данными, которые можно получить в результате допустимых операций над временными данными) в полном объеме.

SupportsExtendedSelectCapability – *getAdapterCapability* может, но не обязан возвращать отличное от нуля значение при вызове с этим параметром, если и только если она возвращает нулевое значение при вызове с параметрами *CanExecuteSchemaSelect* и *CanExecuteTempSelect*.

CanExecuteSchemaSetTypeOperations – *getAdapterCapability* должна возвращать отличное от нуля значение при вызове с этим параметром, если поддерживаются операции с множествами над данными, объявленными в схеме источника. Если в схеме источника содержатся множества, то операции над множествами обязаны поддерживаться.

CanAcceptTempSetTypeData – *getAdapterCapability* может, но не обязан возвращать отличное от нуля значение при вызове с этим параметром, если и только если возвращается отличное от нуля значение при вызове с параметром *UserDataSetsSupport*. Означает, что во временных коллекциях могут содержаться атрибуты – множества.

CanExecuteSetTypeOperationsOnTempData – *getAdapterCapability* может, но не обязан возвращать отличное от нуля значение при вызове с этим параметром, если и только если возвращается отличное от нуля значение при вызове с параметром *CanAcceptTempSetTypeData*. Если возвращается отличное от нуля значение, допустимо использование операций над множествами над временными данными.

AllowSchemaMethodsOnTempData – *getAdapterCapability* может, но не обязан возвращать отличное от нуля значение при вызове с этим параметром, если и только если возвращается отличное от нуля значение при вызове с параметром *UserDataSetsSupport*. Ненулевой результат означает, что все методы, объявленные в схеме источника, могут быть использованы, в том числе и над временными данными.

CanExecuteSchemaAppendID – *getAdapterCapability* должна возвращать ненулевое значение при вызове с этим параметром, если адаптер может выполнять операции *Append* с *Id* предикатом над данными, объявленными в схеме источника. В виде *Id* предикатов будут представлены все операции преобразования типов атрибутов, а также арифметические операции (или логические операции над булевыми аргументами).

CanExecuteTempAppendID – *getAdapterCapability* должна возвращать ненулевое значение при вызове с этим параметром, если адаптер может выполнять операции Append с Id предикатом над временными данными. В виде Id предикатов будут представлены все операции преобразования типов атрибутов, а также арифметические операции (или логические операции над булевыми аргументами).

Приложение Е Спецификация синтетического тестовой задачи

Схема тестового посредника представлена ниже:

```
{Mediator; in: module, s2o_some;
  type:
  { MediatorType1; in: type;
    t1_var1: string;
    t1_var2: integer;
    t1_var3: real;
    t1_var4: date;
    t1_var5: {set; type_of_element: integer;}};
  },
  { MediatorType2; in: type;
    t2_var1: string;
    t2_var2: integer;
    t2_var3: real;
    t2_var4: date;
    t2_var5: {set; type_of_element: real;}};
  },
  { MediatorType3; in: type;
    t3_var1: string;
    t3_var2: integer;
    t3_var3: real;
    t3_var4: date;
    t3_var5: {set; type_of_element: string;}};
  };

function:
{getSum; in: function;
  params: {+r1/real, +r2/real, -returns/real};
},
{concat; in: function;
  params: {+r1/string, +r2/string, -returns/string};
};

class_specification:
{ mediatorClass1; in: class;
  instance_section: MediatorType1;
},
{ mediatorClass2; in: class;
  instance_section: MediatorType2;
},
{ mediatorClass3; in: class;
  instance_section: MediatorType3;
};
}
}
{I_FUNC; in: module, intermediate, s2o_all; import: Mediator;
  function:
  {getStr; in: function;
    params: {+r/real, -returns/string};
  },
  {getDate; in: function;
    params: {+r/real, -returns/date};
  },
  },
  {getSet; in: function;
    params: {+r1/integer, +r2/integer, +r3/integer, -returns/{set; type_of_element:
integer;}};
  },
  },
}
```

Спецификации ресурсов выглядят следующим образом:

```
{CATALOG1; in: module, local;
  type:
    {Catalog1; in: type;
      var1: string;
      var2: integer;
      var3: real;
      var4: date;
      var51: integer;
      var52: integer;
      var53: integer;
    };
  class_specification:
    {catalog1; in: class;
      instance_section: Catalog1;
    };
}
{CATALOG2; in: module, local;
  type:
    {Catalog2; in: type;
      var1: string;
      var2: integer;
      var3: real;
      var4: real;
      var5: {set; type_of_element: real;};
    };
  class_specification:
    {catalog2; in: class;
      instance_section: Catalog2;
    };
}
{CATALOG3; in: module, local;
  type:
    {Catalog3; in: type;
      var1: real;
      var2: integer;
      var3: real;
      var4: date;
      var5: {set; type_of_element: string;};
    };
  class_specification:
    {catalog3; in: class;
      instance_section: Catalog3;
    };
}
```

Взгляды выглядят следующим образом:

```
// GAV for Catalog1
v_Catalog1_Data(x/[var1, var2, var3, var4, var5])
:- CATALOG1.catalog1 (x/[var1, var2, var3, var4, var51, var52, var53])
& I_FUNC.getSet (var51, var52, var53, var5)
& var2 > 10

// GAV for Catalog2
v_Catalog2_Data(x/[var1, var2, var3, var4, var5])
:- CATALOG2.catalog2 (x/[var1, var2, var3, temp: var4, var5])
& I_FUNC.getDate (temp, var4)
& var3 > 10

// GAV for Catalog3
v_Catalog3_Data(x/[var1, var2, var3, var4, var5])
:- CATALOG3.catalog3 (x/[temp: var1, var2, var3, var4, var5])
& I_FUNC.getStr (temp, var1)
& var2 > 10
```

```
//LAVs
Views.v_Catalog1_Data(x/[var1, var2, var3, var4, var5])
  :- mediatorClass1(x/[var1:t1_var1, var2:t1_var2, var3:t1_var3, var4:t1_var4, var5:t1_var5])

Views.v_Catalog2_Data(x/[var1, var2, var3, var4, var5])
  :- mediatorClass2(x/[var1:t2_var1, var2:t2_var2, var3:t2_var3, var4:t2_var4, var5:t2_var5])

Views.v_Catalog3_Data(x/[var1, var2, var3, var4, var5])
  :- mediatorClass3(x/[var1:t3_var1, var2:t3_var2, var3:t3_var3, var4:t3_var4, var5:t3_var5])
```

Программа на языке правил посредника выглядит следующим образом:

```
r(x/[ t1_var1, common, t1_var3, t1_var3, t1_var5, t2_var1, t2_var3, t2_var4, t2_var5, t3_var1,
t3_var3, t3_var4, t3_var4, sum, concat])
:- mediatorClass1(x/[t1_var1, common:t1_var2, t1_var3, t1_var4, t1_var5])
& mediatorClass2(y/[t2_var1, common:t2_var2, t2_var3, t2_var4, t2_var5])
& mediatorClass3(z/[t3_var1, common:t3_var2, t3_var3, t3_var4, t3_var5])
& getSum(t1_var3, t2_var3)
& concat(t1_var1, t3_var1)
```